Universiteit Leiden

Master Computer Science

Evaluation of sequential machine learning approaches for multi-label classification and transition identification of driving scenario categories

Name:	Kartikey Ahlawat
Student ID:	4178262
Date:	31/08/2001
Specialisation:	Data Science
1st supervisor:	Dr. Hari H.H.S. Subraveti
2nd supervisor:	Dr. Mitra Baratchi

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Acknowledgements

This thesis was produced in collaboration with TNO's Integrated Vehicle Safety department and supervised by the Leiden Institute of Advanced Computer Science (LIACS) at Leiden University, made possible using the computational resources of the Kaggle environment. I want to thank my supervisors, Hari H.H.S. Subraveti and Mitra Baratchi, for their invaluable guidance, support, and encouragement throughout the writing of this thesis. I truly appreciate our meetings and email exchanges, which have been very inspiring and have given me a lot of confidence. I also thank my colleagues at TNO for their help throughout the project, from whom I learnt a great deal in both research and engineering. Special thanks go to Erwin de Gelder for his valuable feedback. Completing this thesis would not have been possible without all of you, and I am sincerely grateful.

Abstract

Advancements in automated vehicle (AV) technology underscore the need to assess automated driving systems (ADS) across various driving scenarios to ensure they cover their Target Operational Domain (TOD). Currently, rule-based methods dominate the generation of scenario-based test cases by classifying vehicle trajectory data into corresponding scenario categories. Recent work involves applying transfer learning to a pre-trained transformer-based model; an additional undertaking involves pre-training from scratch (TrajPT). However, work on transformer-based architectures has focused on trajectory prediction rather than scenario category classification or transition identification. Additionally, these models are based on either encoder-decoder or decoder-only architectures, making them suitable for autoregressive tasks but not for discriminative tasks such as classification. This motivated evaluating an encoder-based classifier and comparing its performance with a conventional sequential model. This thesis proposes a new Transformers (BERT) model for the multi-label classification of driving scenario categories and the identification of their transitions. It evaluates it against two sequential machine learning baseline models, namely Long Short-Term Memory (LSTM) and Bidirectional Long Short-Term Memory (BiLSTM), using the highD dataset. The thesis defines a set of comprehensive evaluation metrics: macro F1-score, hamming loss, $\text{NMABE}_{average}$, $\text{MMR}_{average}$, $\text{MMR}_{ST_{average}}$; evaluating the model from both classification and transition identification perspectives. An extensive ablation study on the pre-training and fine-tuning setup of BERT highlighted the impact of hyperparameter values on model training and performance. The study finds that the BERT model pre-trained for $101,568$ training steps in a curriculum-styled, three-stage learning approach and fine-tuned using LoRA performs better on 4 of 5 evaluation metrics compared to (Bi)LSTM. Moreover, BERT fine-tuning took $\approx 94\%$ less time and $\approx 68\%$ fewer trainable parameters. This shows the efficiency of transfer learning over the conventional training method. For supervised learning of (Bi)LSTM and BERT fine-tuning, a rule-based method (TNO's StreetWise tool) was used to label/classify the highD, highlighting the necessity of this approach for creating labelled data for ML model training. This shows that it will remain an essential component of scenario-based testing, regardless of the final classification method selected.

Contents

List of abbreviations

Table 1: Commonly used acronyms in Automated Vehicle [1] and Machine Learning domains.

Acronym	Full form
Automated Vehicle Domain	
AV	Automated Vehicle
ADS	Automated Driving System
ACC	Adaptive Cruise Control
CAN	Controller Area Network
TTC	Time-To-Collision
PET	Post-Encroachment Time
DRAC	Deceleration Rate to Avoid Crash
THW	Time Headway
DHW	Distance Headway
ODD	Operational Design Domain
TOD	Target Operational Domain
LC	Lane change
Machine Learning	
RF	Random Forest
SVM	Support Vector Machine
NN	Neural Network
SLP	Single Layer Perceptron
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
LSTM	Long Short Term Memory
BiLSTM	Bidirectional Long Short Term Memory
GRU	Gated Recurrent Unit
RNN	Recurrent Neural Network
LLM	Large Language Model
MLM	Masked Language Model
MFM	Masked Frame Modelling
NMABE	Normalised Mean Absolute Boundary Error
MMR	Mean Miss Rate

Chapter 1

Introduction

As highlighted in Oliver Wyman's Mobility Investment Radar[1], funding in mobility solutions and technologies surged by US$10 billion in 2025 from 2024, reaching a global investment total of US$54 billion. The 'autonomous and electric vehicle' sector was pivotal in drawing attention within the mobility domain. The rise in driving automation aimed at achieving full autonomy (level 5, as noted in Table 2.2) has underscored the need to assess automated driving systems (ADS) across various scenarios to ensure they cover their target operational domain (TOD). Due to its ability to effectively capture the dynamics and intricacies of driving behaviour without actually driving the test vehicle, scenario-based testing is extensively used. This testing relies on real-world vehicle trajectory data to generate test cases. The goal is to segment these trajectories in relation to an ego vehicle into potential scenarios, classifying them into pertinent categories such as lane change, car following, cut-in, cut-out, etc. Numerous methods have been employed for this purpose, with rule-based strategies often being central [3]. These rule-based methods are either used on their own or coupled with other data-driven algorithms. When used independently, they segment and label the data into respective categories. When combined with other algorithms, they may create weak labels for supervised learning or produce segmentations for clustering. Their popularity stems from their efficiency, as they reduce the manual effort required for labelling and provide labelled data for classifier development.

Researchers have recently applied various machine learning techniques to classify driving scenario categories. Techniques ranging from unsupervised methods, such as clustering, to supervised methods such as RF, SVM, 1D-CNN, LSTM, GRU, and eventually transformer-based architectures [4, 5, 6, 7, 8, 9]. Although most studies do not directly target scenario category classification, they are related and hold potential as relevant supporting work for this research task. With the rise and significance of transformers, reliance on rule-based approaches diminishes during pre-training[2] due to self-supervised learning. However, labels generated by these methods can be used for fine-tuning downstream classification tasks.

Recent research in vehicle trajectory prediction is moving towards testing transformer-based architectures. However, the majority of the research study focused on implementing LLM for vehicle trajectory prediction rather than scenario category classification or transition identification. In this thesis, the BERT model is selected to model the trajectory sequence, since vehicle activity sequences define the vehicle trajectory and have been observed to exhibit interdependence. The current activities could influence the future activities. Given BERT's capabilities, this motivated the implementation and experimentation with it for the multi-label classification of the driving scenario category task, which has not yet been done. A rule-based method (TNO's StreetWise tool) was used to frame-level label highD [10] across 12 scenario categories. This data was then used to fine-tune the model for the downstream task.

The thesis involves pre-training the BERT model from scratch. During the literature review, papers on pre-trained LLMs trained on a vehicle trajectory dataset for trajectory prediction/forecasting were encountered. However, most are not publicly available as a foundation model. Additionally, these models have a decoder-based architecture, i.e., they are suitable for generative tasks but not for discriminative tasks such as classification. Although if trained on large enough data, it could generalise and exhibit emergent classifier abilities. However, by nature, they were developed for next-token prediction (using masked self-attention) in

[1] An annual survey tracking the investments of 20,000 companies in mobility to uncover promising technologies [2].
[2] Training Transformer-inspired architectures from scratch is termed pre-training.

an autoregressive task. Encoder-based architectures, such as BERT, utilise bidirectional self-attention with masked language modelling, which aligns with the downstream task.

The following is a list of opportunities after a review of existing research:

- Recent research has evaluated the performance of LLMs (either encoder-decoder or decoder-only) in predicting vehicle trajectories. However, research is lacking on how well a transformer-based model (specifically the encoder) performs at classifying driving scenario categories (multi-label classification).
- No standard evaluation metrics are defined to assess how well the model identifies transitions between scenario categories.
- Absence of an open-source BERT foundation model that is pre-trained on a vehicle trajectory dataset.

To address the above gaps, this study explores the core questions: *Can BERT be applied to a vehicle trajectory dataset for multi-label driving scenario category classification? How well does it perform compared to the conventional sequential model ((Bi)LSTM)?* On this basis, the following **contributions** are made:

- *Defining enhanced metrics to evaluate the model's ability to identify the transition between scenario categories.*
- *Designing, implementing, and evaluating - LSTM, BiLSTM, and BERT, trained on highD data for multi-label classification of scenario categories and comparing the performance of BERT with BiLSTM.*
- *Performing ablation studies to assess the impact of hyperparameters on BERT's pre-training/fine-tuning.*

The thesis will elaborate on industry-accepted ontologies and definitions for driving scenarios in Chapter 2, including domain knowledge of scenario-based testing and rudimentary ADS. Chapter 3 provides an overview of existing work on the labelling/identification/classification of driving scenario categories, as well as on forecasting future vehicle trajectories. The chapter covers both traditional and state-of-the-art methods. Chapter 4 includes dataset selection and its rationale, as well as descriptive analysis to gain insights into the data structure, distribution of scenario categories, feature trends, and data noise. Chapter 5 presents the research methodology, experimental results, and findings, including the definition of evaluation metrics, the experimental setup, and the implementation of the baseline models (BiLSTM/LSTM) and BERT. It also presents experimental results and findings, including hyperparameter tuning via an ablation methodology, observations on model performance across varying hyperparameter values, Comparisons of LSTM with BiLSTM, and comparisons of BERT with the best baseline configuration. Chapter 6 concludes the paper and outlines directions for future research.

Chapter 2

Background and Definitions

In this chapter, ontologies related to driving scenarios are defined. To avoid ambiguities, a widely accepted standard paper published by Erwin de Gelder is referenced [11], maintaining coherence with the industry-accepted definition. The paper aims to develop a framework that captures complex real-world driving situations in an abstract yet comprehensible manner. It formally defines the core units, concepts, and categories that serve as the building blocks for scenario modelling and assessment. For convenience, the chapter is categorised into (i) definition and ontology of driving scenarios, (ii) scenario definition for assessment of ADS, (iii) scenario-based testing, and (iv) automated Driving System.

2.1 Definition and ontology of driving scenarios

Driving involves continuous interaction with surrounding **physical elements**, such as road infrastructure, vehicles, traffic lights, and pedestrians. The set of these elements forms a driving **environment** where the ego vehicle operates. The **ego vehicle** is the main subject of a scenario through which the world is perceived (environment). Dynamic physical elements in its environment are termed **actors**. A **state variable** quantifies an actor at a given time. These (as exemplified in Table 2.1) are dynamic over time. They can be used to determine future responses based on the equation describing the dynamics [11].

Table 2.1: Major actors of a driving scenario along their possible state variables [11].

Actor	**Possible State Variables**
Ego Vehicle	Kinematics (Velocity, acceleration, position)
Pedestrian	Walking speed, position
Cyclist	Cycling speed, position
Traffic Light	Light colour, time remaining
Road	Surface temperature, condition (smooth, rough, potholes)

A system is said to be in a **mode** if there is no abrupt change in its state variables for a period. Parameter remains stable within a mode, reflecting its current operational state. Change in state variables over time defines an **activity**. Figure 9.1 graphically illustrates lateral (side by side) and longitudinal (straight) activities of a vehicle. Lateral activity is relative to the lane in which the ego vehicle is located. The start and end of an activity are marked by an **event**. It is a defining point in time that marks either a mode transition, a system reaching a given threshold, or both [11].

2.2 Scenario definition for assessment of ADS

A **scenario** is a sequence of activities and events within the time interval of the first and last relevant events [11]. It consists of a quantitative description of relevant characteristics highlighted in the Figure 9.2. Similar scenarios that share common characteristics and nature could be abstracted into qualitative **categories** such

as 'cut-in' and 'LC'. Within these, **critical scenarios** could be identified that pose a high risk or danger to the operation and safety of ADS, such as collision, traffic violations, or other hazardous situations. The criticality of a scenario can be assessed using a surrogate measure (see Table 9.1) that indicates how close the scenario is to a hazardous outcome [12].

A vehicle's **trajectory** consists of a consecutive group of scenarios. Moreover, **transition** is a point where a scenario ends, and a new one begins. Apart from this, the **scenario space** defines Operational Design Domain (ODD) under which an ego vehicle is designed to operate safely. In addition, the Target Operational Domain (TOD) refers to the area where ADS will be deployed. The ideal ADS's ODD should cover TOD [11, 12].

2.3 Scenario-based testing

The test cases are derived from real-world driving scenarios extracted from driving data [11]. The question here might arise as to *"why scenario-based testing for ADS instead of traditional mileage testing?"*[1] The answer is that driving is a complex task involving many test cases, which is infeasible to assess using mileage or engagement testing. Some limitations of mileage testing [3]:

- It is not possible to cover all test cases. For this, AV might need to drive for billions of km, and still, some edge cases might never occur.
- Infeasibility in terms of time and money. Driving an AV over such long distances in all kinds of scenarios takes exponential time and a large sum of money.
- Threat to the traffic, the AV under the test poses a continuous threat to other physical entities sharing the road (other vehicles, pedestrians, cyclists, etc.).

Strengths of scenario-based testing [13, 14, 15]:

- Test ADS on scenarios consisting of various combinations of activities, events, actors, and environmental factors.
- Allows focusing on critical and rare scenarios that might never occur in a standard mileage test.
- Vast exploration of scenario space using scenario generation and simulation.
- Requires less time and money.

The other question that could arise is: *"Why not activity-based testing?"*. The answer is that activity-based testing focuses on individual driving actions (such as accelerating, braking, and lane changing) rather than on the interplay and dependencies between activities that often lead to safety-critical situations. Moreover, scenarios best describe the ODD of ADS [15, 16].

2.4 Automated Driving System

The Society of Automotive Engineers (SAE) documented the taxonomy and definitions for terms related to ADS - SAE J3016. This became the international standard for future work. Table 2.2 lists the standardised levels of driving automation.

[1] Mileage testing involves driving AV for long distances either on public or test roads to evaluate ADS's safety under real-world conditions.

Table 2.2: The six standardised levels of driving automation [1]

Level	Name	Human Role	Example
0	No Automation	Full Control	Automatic emergency braking, blind spot and/or lane departure warning
1	Driver Assistance	Monitor Environment	Lane centering or ACC
2	Partial Automation	Supervise	Lane centering or ACC
3	Conditional Automation	take over when required	Traffic jam chauffeur[2]
4	High Automation	No intervention in defined ODD	Local driver-less taxi
5	Full Automation	No intervention anywhere	Same as 4, but can drive anywhere in all conditions.

As per SAE J3016, the ADS is a combination of software and hardware that jointly navigate the dynamic driving tasks of its ODD. Figure 2.1 graphically represents the basic functioning of ADS listed below [1]:

- The perception module uses raw sensor data from lidar, camera, radar, GPS, or IMU sensors on the vehicle to create a representation of the environment perceived by the ego vehicle. This representation may include road layout, obstacles, and details of surrounding vehicles (velocity, position, etc.).
- The prediction module uses this representation of the perceived world to predict trajectories of surrounding objects (vehicle, cyclist, pedestrian).
- The planning module uses these predictions to plan a safe future trajectory of the ego vehicle that satisfies goals (driving safely), constraints (road, traffic laws, etc.) and predicted behaviour of other vehicles.
- The control module converts these plans to low-level actions such as steering, braking, and accelerating.

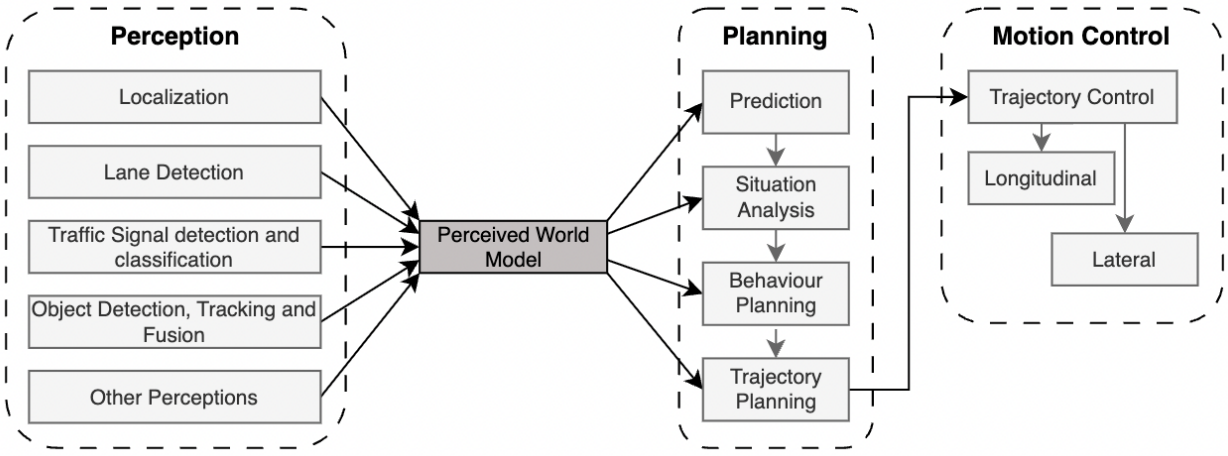


Figure 2.1: Rudimentary functioning of Automated Driving System, referred from [12].

Chapter 3

Related Work

This chapter discusses the initial and recent developments in driving scenario category classification and transition identification using a vehicle trajectory dataset. For convenience, the chapter is categorised into (i) rule-based and unsupervised machine learning (ML), (ii) conventional deep neural networks (DNN), and (iii) transformer-based models. There was little to no literature specifically focused on identifying scenario transitions; therefore, most of the literature concerns scenario category classification. Scenario transition is an implicit part of the scenario category classification process; without identifying the transition point, scenarios cannot be segmented and classified. Given the limited research on this specific problem statement, the review also examines the literature on trajectory prediction. Although the objectives differ, the shared domain makes them relevant enough to include. The fitness of papers published in journals was evaluated based on their impact factor and quartile. Papers of high impact factor (IF) and first quartile (Q1) were preferred. Additionally, papers published in conferences were evaluated on their presence and ranking in the CORE Conference. The following tools were used to retrieve relevant papers: arXiv, ChatGPT-5 (deep research feature enabled), DeepSeek, Perplexity.ai, Gemini, Google Scholar, Grok, IEEE Xplore, and CORE.

3.1 Rule-based and Unsupervised ML model

For testing of ADS, the inefficiency of mileage-based testing has been established, and scenario-based testing is widely accepted for its practicality and efficiency. A rule-based method was proposed to extract and classify scenarios from vehicle trajectory data. Dependence on a rule-based approach increased manual effort and the domain expertise required for rule creation [17]. Subsequent research sought to reduce this by limiting its use to either data segmentation or the creation of weak labels for a classifier [4, 5, 6]. Identifying and slicing at transition points to create scenarios of varying lengths. Clustering (K-Medoids, KPAM) was employed to group similar scenario segments into categories [4], and the elbow method was used to determine the optimal number of categories (clusters) [4]. The model was not evaluated quantitatively [4].

3.2 Conventional DNN models

A study by Elspas et al [5] proposed a fully connected 1D time-series convolutional network (1D-FCN) for semantic segmentation of driving data to detect two scenarios: 'LC' and 'cut-in'. Labelling each time step (frame) with one or more scenarios. Identifying consecutive time steps with a common scenario label defines a complete scenario. The model was trained on weak labels generated by a rule-based scenario detection method rather than manual labelling, as this is less time-consuming and more scalable. 1D-FCN, being a fully connected network, uses both past and future context, improving its predictive performance and generalisation. To compare the two scenario detection methods (1D-FCN and rule-based), manual labelling (actual ground truth) is performed on a subset of data, and the scenarios detected by both methods are compared. 1D-FCN achieved a higher F1-score and recall, but lower precision [5].

In another paper by Aboah et al [6], published around the same time, the authors demonstrate how '$driving\ data \rightarrow segmentation \rightarrow classification \rightarrow scenario\ detection$' is better than applying only classifi-

cation, i.e., '*driving data* → *classification* → *scenario detection*'. Used the Energy Maximisation Algorithm (EMA) for segmenting the trajectory data to identify the start and end of scenarios. Segmented scenario durations fairly match the actual, 59.3% accuracy in 'left LC' and 85.6% in 'lane keeping'. Among 1D-CNN, LSTM, RF, and SVM, 1D-CNN achieved the highest accuracy of 98.99% when classifying segments into pertinent scenario categories [6]. Recently, Zhou et al [7] incorporated bottom-up segmentation with Dynamic Time Wrapping (DTW) to merge adjacent similar points iteratively until a similarity threshold is reached. DTW helped normalise and align different scenario behaviours. A GRU + LSTM hybrid deep learning model was used to label these generated segmentations into discrete regimes such as acceleration, deceleration, cruise, and steady-state vehicle. Achieved an improvement of 43.38% and 26.1% reduction in displacement error (MSE) over baseline LSTM [7].

3.3 Transformer-based models

Lu et al [8] implemented transfer learning by fine-tuning a pre-trained transformer-based model to predict LC (left or right) intent from vehicle trajectory. It was fine-tuned and tested on the highD and NGSIM datasets, and outperformed eight state-of-the-art baselines by at least 10.5%. Average Displacement Error (ADE)[1] and Final Displacement Error (FDE)[2] were used as evaluation metrics [8].

Li et al [9] pre-trained a transformer model (encoder + decoder) on pNEUMA [18] (vehicle trajectory dataset) named TrajPT. It used an autoregressive pre-training framework in which the model learned to predict the next frame's trajectory (spatio-temporal position/interaction) from previous frames. With TrajPT, the paper introduced a graph-based joint spatial–temporal attention module that enables the model to learn complex vehicle dynamics, capturing spatial interactions with nearby vehicles within frames and temporal patterns across frames[3]. The authors performed two key downstream tasks: lane change and trajectory prediction. The evaluation metrics for lane change prediction are accuracy, and for trajectory prediction, ADE and FDE are used. Experiments showed that it performed better than the baseline models (LSTM and a classic feedforward NN) and generalised well across different vehicle trajectory datasets (highD for lane change prediction comparisons and NGSIM for trajectory prediction) [9].

There has been little research specifically on multi-label scenario category classification and transition identification using the BERT or transformer architecture. The nearest topic is 'vehicle trajectory prediction,' for which some recent research has used transformers. Some notable mentions are: [19] and [20].

3.4 Summarisation and conclusive remark

To summarise, the initial research used a rule-based approach to classify scenario categories [17]. Subsequent studies reduced reliance on rule-based methods because creating rules required manual effort and did not guarantee successful classification. Rule-based approaches were used only to detect transition points in scenarios, segment trajectories into prospective scenario segments, and cluster similar scenario segments [4]. However, rule-based methods were not entirely abandoned; deep learning networks such as 1D-FCN and 1D-CNN were trained on weak labels generated by rule-based systems or manually annotated data for scenario category classification [6, 5]. Over time, time-series models such as GRU and LSTM were explored, followed by the introduction of transformers [7, 8]. Recent work involves applying transfer learning to a pre-trained transformer-based model; an additional undertaking involves pre-training from scratch, such as TrajPT [9]. However, work on transformer-based architectures has been limited to trajectory prediction rather than classification or transition identification. Additionally, these models are based on either encoder-decoder (transformer) or decoder-only architectures. Regarding the dataset, most papers used publicly available vehicle trajectory datasets, except for a few that used their proprietary datasets. The public datasets used were: highD [10], NGSIM [21], Nebraska NDS [22], and pNEUMA [18]. The evaluation metrics used for scenario category classification were accuracy, precision, recall, and F1-score. This thesis bridges the gap in applying transformer-based architecture for the driving scenario category classification task. Moreover,

[1] ADE: Euclidean distance between actual and predicted values averaged across the trajectory.
[2] FDE: Euclidean distance between actual and predicted value for the final position at the predicted horizon
[3] Frame refers to an instance where the vehicle trajectory was recorded in tabular format.

defines comprehensive evaluation metrics that incorporate transition identification along with classification performance.

Chapter 4

Dataset

This chapter defines the dataset used in this study, its features, and explains the rationale for its selection. Moreover, exploratory and descriptive analyses were performed to gain insights into the feature trends.

HighD (Highway Drone Dataset) is an initiative of the Institute for Automotive Engineering (ika) of RWTH Aachen University. The highD dataset was collected primarily for the safety validation of highly automated vehicles. Trajectories for more than 110,500 vehicles were captured using a drone from six different locations on German highways, covering different traffic states. The vehicle trajectory, including type, size, and manoeuvre, is automatically extracted with a position error of less than 10 cm. Pre-extracted information includes surrounding vehicles, metrics such as time headway (THW) or time-to-collision (TTC), and manoeuvres such as lane changes (LCs) [10].

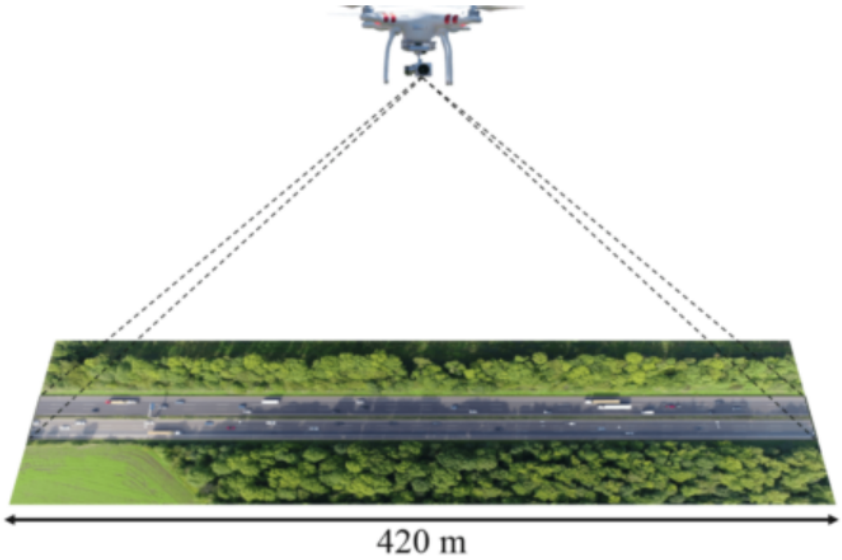


Figure 4.1: Visual depiction of data collecting [10]

4.1 Reason for choosing highD

Apart from highD, there are other vehicle trajectory datasets, such as SHRP2 [23], KITTI [24], Cityscapes [25], and NGSIM [21]. Each of them has limitations. SHRP2 (USA) was collected using radar, CAN bus, GPS position, and cameras, but is not in the public domain. KITTI and Cityscapes focus on computer vision tasks. NGSIM is the most favourable of these, but it also has limitations. Cameras mounted on tall

buildings captured the vehicle's trajectory on the highway. It was collected in 2005-2006; therefore, the camera resolution was poor, leading to inaccuracies during vehicle segmentation (overlapping vehicle boundaries resulting in false collisions) and speed estimation. It was unable to capture the complete description of the dynamic scenario as outlined in Figure 9.2.
highD was collected using 4K DJI Phantom 4 Pro Plus drones, capturing the vehicle's longitude and latitude with high accuracy. It is more diverse due to more sites and different times of the day. Moreover, has a larger number of critical manoeuvres. Therefore, it outperforms NGSIM in data quality, diversity, and accuracy [10].

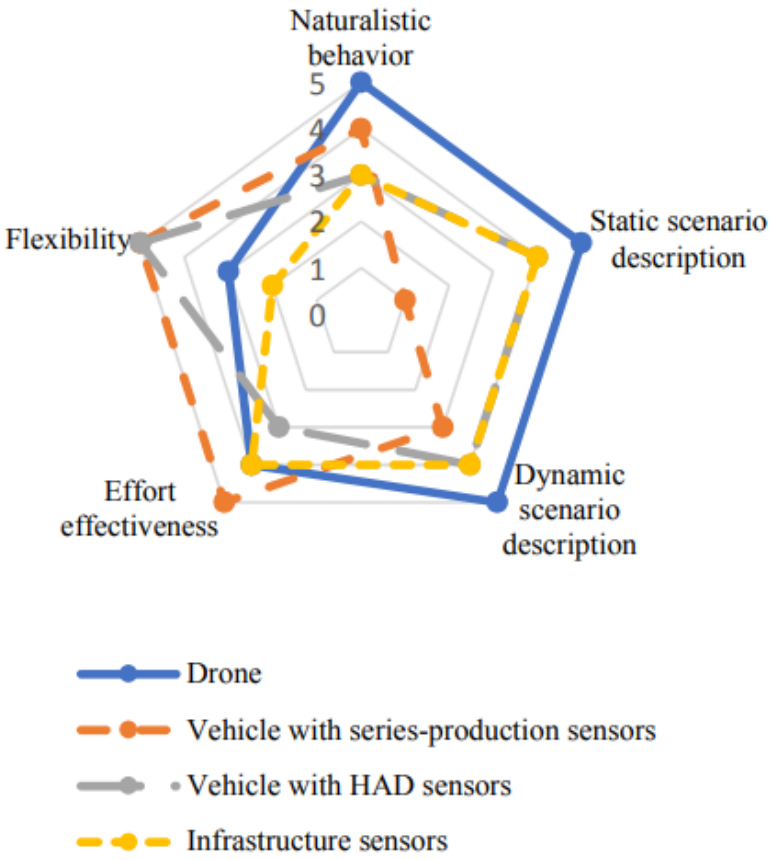


Figure 4.2: Comparison of vehicle trajectory capture methods [10]

4.2 Descriptive Analysis

Table 4.1: highD description [10]

Attribute	Value
drone vision/range [m]	420
Driven distance [km]	45 000
Driven time [hrs]	447
Sampling rate	25 frames per second (25 Hz; meaning 0.04 s gap between frames)
Lanes (per direction)	2-3
Vehicles	110 000
Cars	90 000
Trucks	20 000

Table 4.2: Comparison of Parameters between Car and Truck (values are approximate)

Parameter	**Car**		**Truck**	
	Mean	**Std**	**Mean**	**Std**
Speed [m/s]	32.8	4.9	23.5	1.85
Minimum DHW [m]	66.9	68.5	69.2	67.3
Minimum THW [s]	2.1	2.1	2.9	2.1
Minimum TTC [s]	163.17	2282.26	165.68	2430.74
LC per vehicle	0.15	0.37	0.01	0.11

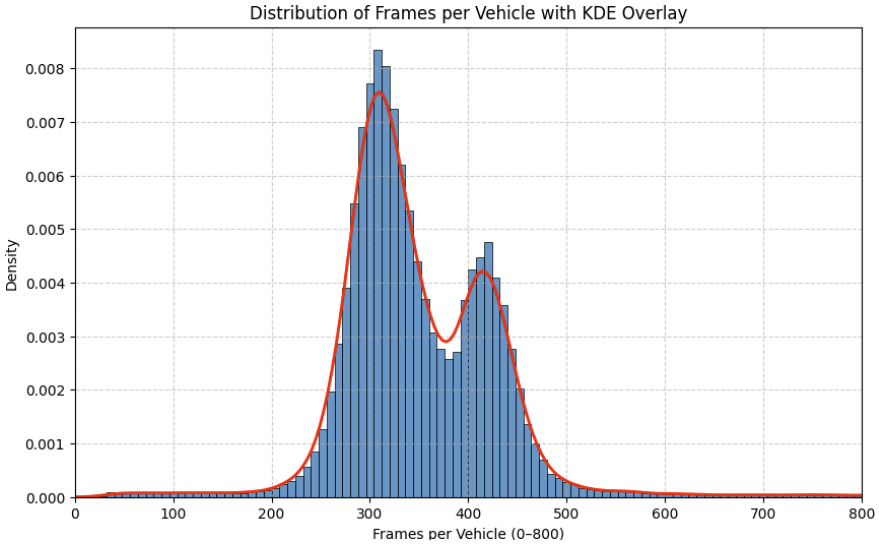


Figure 4.3: Probabilistic distribution of frames per vehicle. For most vehicles, frames range from 250 to 450 numbers.

4.2.1 Cut-in, LC, and traffic behaviour

From Figure 9.3a, it could be inferred that the most cut-ins occur with a THW between 0.75 and 2 s, peaking around 1 s and signifying that in most cut-ins, the following vehicle in the target lane is close. It could also be noticed that there are the following vehicles in the target lane with $THW < 1$ that are considered unsafe on a highway [26]. The distribution is skewed to the left, forcing the following vehicle in the target lane to

react quickly.
From Figure 9.3b, it can be seen that with an increase in the speed of the ego vehicle, there is also a slight, gradual increase in THW ($\approx 1 \rightarrow \approx 2$). Moreover, after 130 km/h, the variation in THW gradually reduces. These observations align with the general understanding and behaviour of driving, where the cut-in vehicle generally tends to maintain a larger gap if the following vehicle in the target lane is at a higher speed. However, irrespective of speed, drivers still perform unsafe cut-ins ($THW < 1$) [10].

From Table 9.2, it can be observed that the majority of LCs occur between $2 \leftrightarrow 3$ ($leftward\ direction$) and $4 \leftrightarrow 5$ ($rightward\ direction$). These lanes are the fast lanes for their respective directions. This observation makes sense, as slower, larger vehicles, such as trucks, typically occupy the outermost lanes and perform fewer lane changes. Another notable finding is that the portion of LCs from the inner lanes to the outer lanes exceeds that in the opposite direction. For example, $4 \rightarrow 5 > 5 \rightarrow 4,\ 5 \rightarrow 6 > 6 \rightarrow 5$

It was found that vehicles at higher speeds take less time to LC. Moreover, heavy vehicles (trucks) take more time in LC than light vehicles (cars) [27].

Furthermore, the areas near the entrance/exit point of the highway experience more lane changes than the rest [28]. It was also found that there was a negative correlation between LC and vehicle speed with traffic density [number of vehicles/km]. Similarly, THW showed a negative correlation with vehicle flow rate [29].

4.2.2 About extracted scenario categories from highD

Using a rule-based method (TNO's StreetWise tool), the highD data was classified into 12 unique driving scenario categories (see Table 9.3). In these scenario categories, the ego vehicle is either keeping or changing lanes while interacting with surrounding actors that may affect its trajectory. Some scenario categories can co-occur, while others are mutually exclusive (see Table 9.4). Table 9.5 illustrates the data imbalance, with 'Ego merging into an occupied lane' having the least presence (0.7%), and 'Lead vehicle cruising' having the most (52.8%). Moreover, it shows that the scenario categories in which the ego vehicle maintains the lane, whether or not it is following another vehicle, last the longest. Table 9.6 highlights smooth transition combinations. Generally, identifying a transition point between such smooth consecutive scenario categories is challenging due to the ambiguity. Still, some transitions are easier to identify because of a drastic change in parameters. Scenario categories like 'overtaking an ego vehicle' are independent of preceding scenarios and can co-occur with others. The plot in Figure 9.5 shows that there are no significant overlaps among consecutive scenario categories with smooth transitions.

The highD data set is large, comprising 60 CSV files totalling around 40 MB each. Therefore, the initial few files were processed to create sequences for supervised learning of the baseline ML models. Sample training datasets of 221,700 and 484,507 sequences were created using a sliding window of 25 and a stride of 1. Tables 9.3 and 9.5 were based on sample data of 484,507 sequences.

Chapter 5

Proposed method

During the literature review (Chapter 3), it was clear that the research was moving towards testing transformer-based architectures for multi-label classification of driving scenario categories and transition identification. In this paper, a BERT-based model is proposed and evaluated for the task. This chapter introduces BERT and explains the rationale for its selection.

5.1 BERT

BERT (see Figure 9.11), as an encoder in the transformer, introduced bidirectional multi-head self-attention [30]. This enables the model to process input tokens in parallel and to create contextual embeddings that capture global context and inter-dependencies between features (e.g., velocity, TTC, THW) or activities (e.g., acceleration, Lane change, deceleration). It is faster, more scalable, and can handle larger datasets. This thesis builds it from scratch, involving pre-training and fine-tuning [30]:

- Pre-training is a self-supervised task. It is done to create generalised weights that can serve as a foundation for downstream tasks such as classification. It involves training BERT for a fixed number of steps, generally a very large one ($> 100,000$). Moreover, it is preferred to do this on a large, unlabelled training dataset. The larger the training step and the data, the better the model will be able to generalise and form rich contextual embeddings.
- Fine-tuning is a supervised task. Pre-trained weights are further fine-tuned using labelled highD data for the respective downstream task.

BERT is also known as a masked language model (MLM), as during pre-training it masks random tokens in sequences and tries to predict them, thereby learning the context [31, 32].

5.1.1 Why transformer-based encoder?

The sequences of vehicle activities define the vehicle trajectory, and it has been observed that these activities sometimes exhibit interdependence. The current activities could influence the future activities. For example, an AV navigating through a city encounters an obstacle and performs a manoeuvre; in this case, the manoeuvre depends on obstacle detection. Moreover, a vehicle's action or activity is also dependent on the trajectory of surrounding vehicles. For example, a preceding vehicle in the left lane decides to LC to the ego vehicle's lane, which could prompt the ego vehicle to decelerate if there is a chance of collision.
Having established this understanding, it is clear how important it is to take the context of other actions/activities into account when identifying scenario category transitions. Naive RNN methods could be used, but they all come with limitations:

- RNN (Many-to-One): In hidden layers, uses activation functions like tanh and ReLU, which cause vanishing or exploding gradient issues. As a result, it is unable to retain information for longer sequences.

- LSTM: To some extent, it overcame traditional RNN's limitations by introducing control gates to withhold relevant context for longer down the line. However, it processes the input in a sequential order, which makes it slower and suboptimal for large datasets. Moreover, sequential input processing limited its ability to capture the global context.

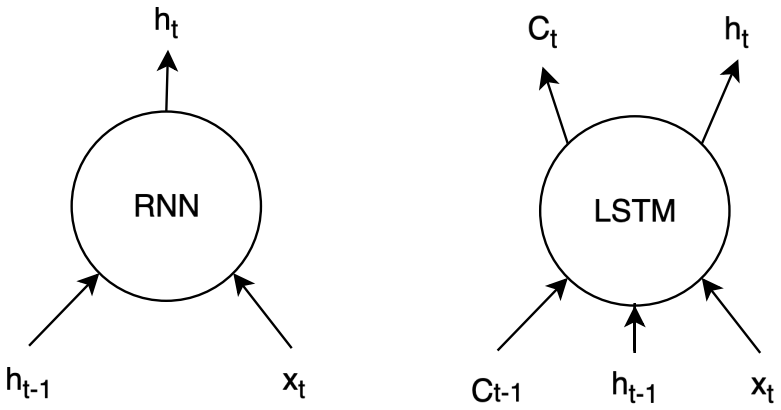


Figure 5.1: Simple schematic representation of RNN and LSTM, where x_t refers to the input at current timestamp t. $h_t - 1$ and h_t refer to the output of the previous and current hidden state, respectively. $C_t - 1$ and C_t refer to previous and updated context vectors, respectively, that carry relevant information.

Transformer introduced the concept of self-attention, which enables the model to process input tokens in parallel and to create contextual embeddings that capture global context and interdependencies between features (e.g., velocity, TTC, THW) or activities (e.g., acceleration-LC, ASV-deceleration, LVD-braking). It is faster, more scalable, and can handle larger datasets.

Chapter 6

Experiments

This thesis aims to answer the following core questions with its experiments:

- Can BERT be applied to a vehicle trajectory dataset for multi-label driving scenario category classification?
- What is the effect of various hyperparameters on BERT's pre-training and fine-tuning processes?
- How well does it perform compared to the conventional sequential model?

This chapter defines a baseline model and task-specific evaluation metrics to compare with BERT. Ablation studies were conducted to identify optimal hyperparameter configurations and understand their impact. Each configuration was run three times with different seed values to assess the model's robustness. For convenience, the chapter is categorised into (i) exploring methods for the baseline, (ii) scope and setup, (iii) (Bi)LSTM, and (iv) BERT.

6.1 Exploring methods for the baseline

Given the temporal data and the nature of the task LSTM and BiLSTM appeared to be the most suitable baseline. It can model time and handle high-dimensional data while capturing smooth transitions. It processes data sequentially and addresses the limitations of 1D-CNNs and RNNs when handling longer input sequences and capturing global context. The usage of three control gates (input, output, and forget gates) and additive gradient updates enabled it to mitigate the vanishing gradient problem and retain relevant information for longer [33]. Moreover, many recent studies used it as a baseline for a similar task [6, 7, 9]. Apart from this, GRU is a faster alternative but is less accurate.

6.2 Scope and setup

The research is limited to the kinematic (velocity, acceleration, etc.) and surrogate safety (TTC, THW, etc.) features presented in the highD dataset. However, driving is a complex task influenced by exogenous factors such as weather, lighting, and the driver's vision and physical health. These factors, which were not part of the dataset, were not considered. Moreover, highD with scenario category labelling done by a rule-based method (TNO's StreetWise tool[1]) was assumed to be the ground truth.

Apart from this, all implementations and experimentation were performed on a Kaggle environment. This itself comes up with memory and compute constraints. Therefore, the code was optimised to run on the following hardware configurations:

[1]For more information, click here

Table 6.1: Hardware configuration on Kaggle workspace with respect to the method used.

Method	Accelerator	GPU RAM	CPU RAM	Disk space
BERT	NVIDIA P100	16 GB	30 GB	57 GB
(Bi)LSTM	NVIDIA T4 ×2	56 GB	15 GB	57 GB

6.3 Model evaluation

Models were evaluated from two different perspectives- scenario category classification and scenario category transition identification. As shown in Table 9.5, the dataset is imbalanced, making traditional metrics such as accuracy misleading. Therefore, for multi-label scenario category classification, the selected evaluation metrics are macro F1-score and hamming loss. Macro F1-score treats all classes equally, and hamming loss captures the fraction of incorrectly predicted labels and is not as strict as subset accuracy (exact match). Reason for choosing $F\beta \rightarrow F_1$ was that it weights precision and recall equally when computing their harmonic mean. During their calculation, masked frames where mutually exclusive scenario categories co-occur were not considered (noise; not ground truth).

$$F_1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

$$F_1^{\text{macro}} = \frac{1}{C} \sum_{i=1}^{C} F_{1,i}$$

$$\text{Hamming loss} = \frac{\text{Number of incorrect label predictions}}{\text{Total number of labels across all samples}} = \frac{1}{N \cdot C} \sum_{i=1}^{N} \sum_{j=1}^{C} \mathbf{1}\{y_{i,j} \neq \hat{y}_{i,j}\}$$

Where:

- TP = True positive,
- FP = False positive,
- FN = False negative,
- N = number of frames,
- C = number of unique target classes (12 in the case of this study),
- $y_{i,j}$ = true label for frame i, class j,
- $\hat{y}_{i,j}$ = predicted label for frame i, class j,
- $1\{\cdot\} = 1$ if the condition is true, otherwise 0.

On the other hand, to evaluate the identification of the transition between scenario categories, Normalised Mean Absolute Boundary Error (NMABE) was used. It computes the difference between the predicted transition point for a scenario category and the actual transition point (the point refers to the frame). This difference is then averaged over all sequences for that scenario category. This yields 12 NMABE values for 12 distinct scenario categories. To get a single number, the average is taken.

$$\text{NMABE}_{average} = \frac{1}{C} \sum_{i=1}^{C} \frac{1}{L_v} \sum_{v} \frac{1}{T_{i,v}} \sum_{j=1}^{T_{i,v}} |b_{(i,v)_j} - \hat{b}^{\text{nearest}}_{(i,v)_j}|, \qquad 0 \leq \text{NMABE}_{average} \leq 1.$$

Where:

- $T_{i,v}$ = number of transitions ($0 \rightarrow 1$) for a particular class and vehicle in actual test data,
- L_v = trajectory length of the vehicle v,

- $b_{i,v}$ = true transition point for class i in the trajectory of vehicle v,
- $\hat{b}_{i,v}^{\text{nearest}}$ = predicted transition point for class i in v, that is nearest to the $b_{(i,v)}$.

As vehicle trajectories vary, longer trajectories are more prone to higher MABE. Therefore, it is necessary to have a normalised MABE value that provides an unbiased comparison regardless of varying trajectory lengths.

6.3.0.1 Supplementary evaluation

In addition to the three aforementioned metrics, two additional metrics were defined. These could be considered an extension of $\text{NMABE}_{average}$.

- Mean miss rate (MMR): Calculates the ratio of vehicles where the model missed (misclassified) the actual scenario category throughout its entire trajectory. $\text{MMR}_{average}$ takes the macro average across all scenario categories. See Algorithm 3 for the definition.
- Mean miss rate over smooth transitions (MMR_{ST}): Calculates the ratio of instances where the model missed (misclassified) the actual smooth transition (see Table 9.6 for smooth transition pairs) throughout the entire trajectory of the vehicles. $\text{MMR}_{ST_{average}}$ takes the macro average across all such pairs. See Algorithm 4 for the definition.

See Appendix 9.5.3 for the test set preparation.

6.4 (Bi)LSTM

Stateless (Bi)LSTMs are employed considering shorter average sequence length and sequence shuffling to mitigate overfitting and improve generalisation. It treats sequences independently and trains faster [34]. These layers were set to $return\_sequences = True$, which allowed the layers to pass output for individual frames rather than for the final frame of the sequence, which would have been the case otherwise. This allows the model to learn and backpropagate the cumulative loss across all the frames in the sequence. In the output layer, 'sigmoid' was used as an activation function (multi-label classification). For model definition summary, see Table 9.7 and for data preprocessing steps see Appendix 9.5.1.

(Bi)LSTM, being a supervised learning technique, requires labelled data. A rule-based method was used for the frame-level labelling of highD. Input with 38 features, among which 12 are one-hot encoded target features (scenario categories, Table 9.3).

6.4.1 Ablation study setup

An ablation study on key parameters (see Table 6.2) was conducted, varying each parameter individually while keeping the others at their baseline values. See Appendix 9.5.1 for data pre-processing steps.

Table 6.2: Hyperparameter Setting for Ablation Study on BiLSTM/LSTM — the bold values indicate the best results and were kept static for further ablation on [*training size*, *layer*]

Hyperparameter	**Ablation values**	**Baseline value**
Batch size	[**32**, 64]	32
Optimizer	[Adam, **AdamW**]	AdamW
Classifier threshold	[static (0.5), **custom** to each class]	custom
(Bi)LSTM layer (size: 64)	[2, 4]	-
Training size (sequence numbers)	[221700, 484507]	-

On performing the ablation study, the best hyperparameter values of each (highlighted in bold in Table 6.2) were taken and further ablation was performed on [*training size*, *layer*].

6.5 BERT

6.5.1 Pre-training: Ablation study setup

An ablation study on key parameters (see Table 6.3) was conducted, varying each parameter individually while keeping the others at their baseline values. See Appendix 9.5.2 for data pre-processing and pre-training setup.

Table 6.3: Hyperparameter Setting for Ablation Study on BERT pre-training - the bold values indicate the best results

Hyperparameter	**Ablation values**	**Baseline value**
Sequence length	[**64**, 128, 192]	64
Batch size	[256, **128**, 64]	256
Masking ratio	[0.15, 0.3, **0.9**]	0.15
Masking method	[default: **feature**/frame-level, dynamic, span]	default frame-level
Training loss function	[MSE, **Huber**]	Huber
Activation f (FFN in encoder)	[GeLU, **SwiGLU**]	GeLU
Dropout ratio	[0.0, 0.05, 0.1, 0.15]	0.0
Overlapping sequence ratio	[0.0, **0.2**, 0.5, 0.8]	0.0
Learning rate scheduling	[(**Warmup**+**CosineLRschedule**), $ReduceLROnPlateau$]	$(Warmup + CosineLRschedule)$

For completing the ablation study in a reasonable time, 10% of the total processed data of sequence length 64 was considered. Moreover, the model capacity was kept low, as fewer parameters→ lower computation/memory usage. See Table 6.4 for BERT's configuration when performing an ablation study and full pre-training on optimal hyperparameter configuration. A common seed was set for numpy, os, random, and tensorflow for consistent weight assignment and floating-point operations during experimentation.

Table 6.4: Configuration of BERT in ablation and full pre-training mode.

Mode	**Model dimension**	**Number of heads in an attention layer**	**Number of encoder blocks**
Ablation	64	4	2
Full pre-training	256	8	8

Note: Before analysing the ablation plots below, it is important to understand their nature.

- Up to a point where the val $(--)$ or train $(-)$ curves horizontally straighten up, meaning they become parallel to the x-axis, is the point till the model actually runs ($Epochs\_model\_ran$). For example, in the MAE plot of Figure 7.1a, the model with sequence length 64 (blue curve) runs for ≈ 90 epochs.
- Used $EarlyStopping(patience = 10)$. Therefore, when evaluating loss/MAE curves, will look till $Epochs\_model\_ran - 10$ $epochs$.

6.5.2 Computational and memory estimation for the pre-training

From the time of release of the quintessential paper on transformers [30], there has been a general practice of pre-training transformer-based models for a fixed number of steps. No additional stopping criteria, such as early stopping, are used. The transformer models are intrinsically data and training-hungry. The more data and training steps, the better the model understands and creates richer contextual embeddings, equipping it to generalise and perform better on downstream tasks [35, 36]. This makes it important to estimate the training time per epoch and the memory required for forward/backward propagation before actually pre-training. This mitigates the risk of encountering an out-of-memory (OOM) error or prolonged training time. Calculations are found in Appendix 9.5.4.

6.5.3 Fine-tuning: Ablation study setup

After completing the ablation study on the pre-training setup, clarity emerged on the optimal hyperparameter values. The model's performance during the ablation study was evaluated on the validation set under MLM rather than on a test set specific to the downstream multi-label classification task. Therefore, this underscores the importance of examining whether the performance of some of these hyperparameters translates into classification evaluation metrics on the test set or not. Fine-tuning is necessary to tune the pre-trained model in a supervised manner for the downstream task and labelled highD (done by StreetWise tool) data was used for this. Low-Rank Adaptation (LoRA) was adopted for fine-tuning, as recent research finds it matches or slightly exceeds full fine-tuning performance while being more parameter and memory-efficient [37]. 'All-linear-only' LoRA was chosen for its memory and compute efficiency over 'all-in fine-tuning' and better performance than 'attention-only' LoRA. LoRA adapters are injected into every linear transformation of the encoder, including both attention and feed-forward layers, while the base model weights were kept frozen (see equation below). Refer to Algorithm 8 to get an overview of fine-tuning using LoRA.

$$y = xW + \alpha \cdot xAB$$

Where:

- x is an input sequence of trajectory data,
- y is a multi-label binary scenario category indicator per frame,
- W is frozen pre-trained weights,
- A and B are LoRA adapters
- α controls the adaptation strength.

LoRA made it feasible to fine-tune around $\approx 108,850$ weights rather than $6,894,556$ pre-trained weights, which is a reduction of $63\times$. To keep the final evaluation unbiased, the pre-trained BERT was fine-tuned on the same labelled data (subset of highD) as the BiLSTM (baseline) was trained on. The best model configuration from Table 9.12 was used to pre-train BERT for 101,568 steps, and the resultant $BERT_{101{,}568\ steps}$ was used for the ablation study in Tables 9.13 and 9.14. Since the dataset was imbalanced, it was decided to exclude scenario categories occurring in fewer than 100 vehicles when computing the macro F1-score. A threshold of 100 was set heuristically and resulted in the removal of the following two categories: '*Ego vehicle performing lane change with vehicle behind*' and '*Ego merging into an occupied lane*'. This ensures that the model is not punished for performing poorly on rare categories, and that it did not encounter enough during training to generalise well.

Chapter 7

Results and observations

This chapter presents the results of the ablation studies conducted on LSTM, BiLSTM, and BERT. It evaluates how various hyperparameters affect the pre-training and fine-tuning processes of BERT. Additionally, it compares the performance of BERT against the baseline, focusing on their respective optimal hyperparameter configurations.

7.1 Ablation study on (Bi)LSTM

The purpose of conducting an ablation study on (Bi)LSTM is to identify optimal hyperparameter configurations. Utilising the determined hyperparameter values to establish an effective baseline against BERT. From the ablation study, it was observed that the BiLSTM achieved the best performance of macro f1- 0.38, hamming loss- 0.073, and $\text{MMR}_{ST_{average}}$- 0.77, evident from Table 7.1. However, LSTM achieved the lowest $\text{NMABE}_{average}$ of 0.146 and $\text{MMR}_{average}$ of 0.32. The coefficient of variation across all configurations and five evaluation metrics was less than 0.05, suggesting model reliability.

Table 7.1: Performance of LSTM and BiLSTM on test set with best hyperparameter values (refer to Table 6.2). For experiments, the model was trained on either 221,700 or 484,507 overlapping sequences of size 25 with either 2 or 4 (Bi)LSTM layers. Performance was evaluated on an unseen set of 41,689 non-overlapping sequences of size 25 (created from trajectories of 1,908 unique vehicles). The model with each configuration is run three times with different seed values to report *mean*$\pm$*standard deviation*. The coefficient of variation across all configurations and five evaluation metrics is less than 0.05, suggesting model reliability.

Metric	**LSTM**				**BiLSTM**			
	221,700		484,507		221,700		484,507	
	2 layers	4 layers	2 layers	4 layers	2 layers	4 layers	2 layers	4 layers
Macro F1-score ↑	0.35 ± 0.007	0.34 ± 0.007	0.37 ± 0.007	0.36 ± 0.0	0.37 ± 0.0	0.36 ± 0.007	0.38 ± 0.0	0.38 ± 0.007
Hamming loss ↓	0.079 ± 0.0	0.08 ± 0.0	0.077 ± 0.001	0.079 ± 0.001	0.074 ± 0.001	0.076 ± 0.0	0.073 ± 0.0	0.073 ± 0.0
$\text{NMABE}_{average}$ ↓	0.160 ± 0.009	0.170 ± 0.0	0.146 ± 0.02	0.176 ± 0.003	0.184 ± 0.016	0.167 ± 0.006	0.173 ± 0.005	0.178 ± 0.008
$\text{MMR}_{average}$ ↓	0.35 ± 0.004	0.37 ± 0.01	0.32 ± 0.007	0.32 ± 0.007	0.35 ± 0.002	0.38 ± 0.02	0.33 ± 0.007	0.37 ± 0.016
$\text{MMR}_{ST_{average}}$ ↓	0.95 ± 0.05	0.88 ± 0.057	0.87 ± 0.044	0.83 ± 0.04	0.89 ± 0.02	0.87 ± 0.014	0.94 ± 0.049	0.77 ± 0.014
Training time (mins)	≈ 20	≈ 28	≈ 57	≈ 70	≈ 16	≈ 29	≈ 72	≈ 85

Metrics (↑) indicate that higher values are better and vice versa for (↓). $mean \pm standard\ deviation$ suggests the best value for the respective evaluation metric.

The following is a list of observations from the experiments performed with varying data size and (Bi)LSTM layer:

- Increasing the capacity of the model, that is, adding more layers, generally decreased the performance on all three evaluation metrics. This seems contradictory at first; however, larger models require more data to generalise, and this is most likely the case here (see the following point). Moreover, in some cases, additional layers can cause gradients to vanish during backpropagation, leading the initial parameters to stop learning and the model to perform poorly (this is unlikely to be the case here).

- Increasing the size of the training set (sequences in Table 7.1) improved performance across all three evaluation metrics, indicating that more data adds complexity and, with an adequate number of model parameters, enhances generalisation. However, the $\text{NMABE}_{average}$ increased for the BiLSTM/LSTM model with 4 layers. This could be explained by the fact that training the model using negative log-likelihood enhances its understanding of the class distributions. Still, it does not necessarily improve temporal alignment for larger models (4 layers). It may require additional data to achieve better temporal alignment.

- Each BiLSTM configuration ($sequence - layer$ combination) performed better on macro F1-score and hamming loss compared to LSTM. However, the same cannot be said for $\text{NMABE}_{average}$. Indicating that the succeeding frames enhanced understanding of class distribution, but were unable to improve temporal alignment.

- For smaller data, both LSTM and BiLSTM converged after a similar time. However, on a larger set, a considerable increase in the convergence time was observed ($21 - 26\% \uparrow$).

- Increasing either the training set or model capacity improved its temporal alignment and lowered the $\text{MMR}_{average}$ and $\text{MMR}_{ST_{average}}$. Apart from this, at the model level, LSTM performed better on $\text{MMR}_{average}$, indicating better identification of transitions between scenario categories. However, BiLSTM performed better on $\text{MMR}_{ST_{average}}$, indicating better smooth-scenario category transition identification.

- Out of the total 16 possible smooth transition combinations (see Table 9.6), only 7 were found in the defined test set of 41,689 sequences ($1\ sequence = 25\ frames$). This suggests that the remaining smooth transitions are rare. Identifying the transition point for '*Ego vehicle driving in lane without lead vehicle* $\rightarrow$ *Lead vehicle cruising*' was the least challenging compared to other smooth transitions (see Table 9.11). On the contrary, '*Cut-out in front of ego vehicle* $\rightarrow$ *Lead vehicle cruising*' transition point seems most challenging to identify.

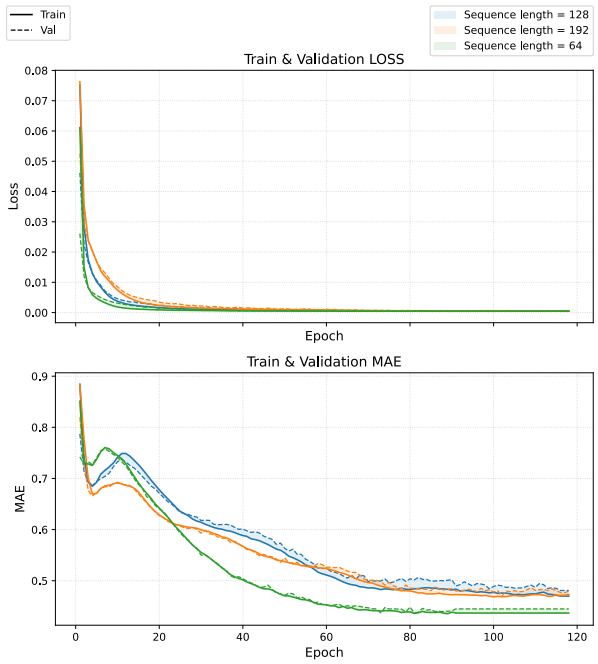
- As training instances for scenario category ('Vehicles' column) increase, the F1 score increases as well (see Table 7.3). This monotonic relationship was confirmed by fitting regression (OLS), and the results ($R^2 = 0.534$ & $p = 0.006$) suggest a statistically significant linear trend. The quadratic term did not significantly improve the R^2 *or* p, indicating that the relationship was primarily linear. Interestingly, *'Ego vehicle performing lane change'* achieved a decent F1-score of 0.65 despite lower support. Showing that it was easier for the model to learn its pattern.

Based on the observations, the BiLSTM model with hyperparameter $[sequences : 484,507; layers : 4]$ achieved the highest performance across three evaluation metrics. It excelled in classification-based metrics, specifically the macro F1-score and Hamming loss. However, it performed worse than the LSTM model on transition-based metrics, such as $\text{NMABE}_{average}$ and $\text{MMR}_{average}$, except $\text{MMR}_{ST_{average}}$. This indicates that while capturing temporal alignment is generally challenging, the BiLSTM model was more successful in recognising smooth transitions. Performing best on 3/5 evaluation metrics makes it the ideal choice as the final baseline against the proposed BERT-based model.

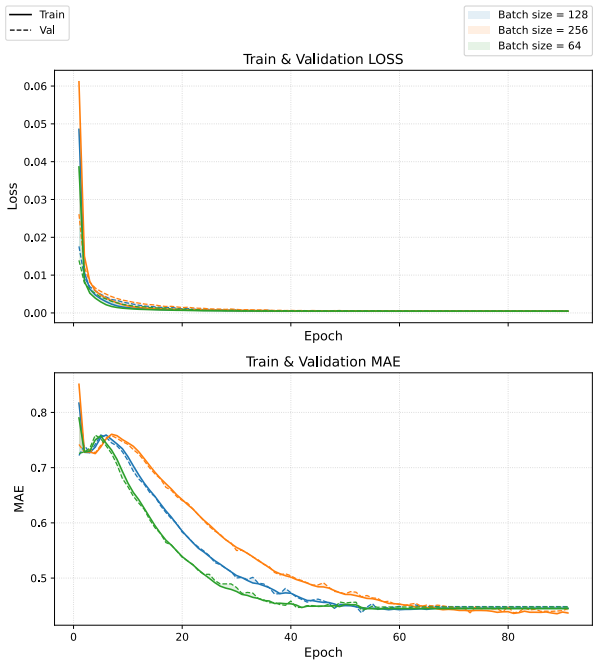
7.2 Ablation study on BERT

The purpose of this section is to understand how hyperparameters affect the pre-training and fine-tuning processes of BERT. Additionally, we aim to identify the optimal hyperparameter configuration for a complete BERT implementation.

7.2.1 Pre-training



(a) Training and validation loss/MAE on different input sequence lengths

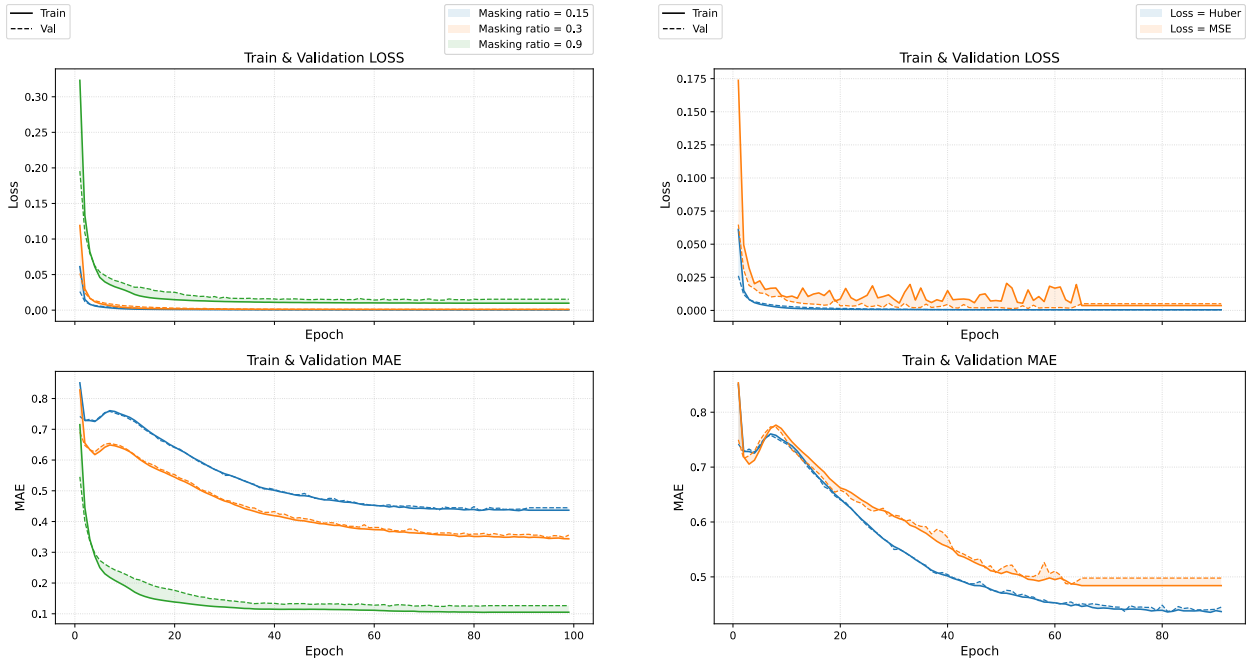


(b) Training and validation loss/MAE on different batch sizes

Figure 7.1: Comparing ablation results for sequence and batch size.

Observing Figure 7.1a, all three sequence lengths converged to near zero loss without much difference in $train - \& - val$. The sequence length of '64' showed the steepest/fastest convergence, reaching a lower loss earlier. Similarly, on the MAE graph, '64' converged to the lowest value. Larger sequences '128' and '192' showed slightly erratic behaviour, converging to a suboptimal higher MAE. Moreover, the $train - \& - val\ MAE$ difference seems to increase with larger sequence lengths. Additionally, the sequence length of '64' trains for ≈ 90 *epochs*, suggesting that a model trained on a shorter sequence length requires fewer iterations before it starts overfitting.

In Figure 7.1b, overlapping curves show that all three batch sizes showed similar convergence to a low loss. In the MAE plot, all three converged to a similar MAE. Batch size of '64' showed early convergence ($\approx 58^{th} epoch$) followed by '128' ($\approx 60^{th} epoch$), highlighting that increasing batch size reduces how fast the model learns, which is intuitive as the backward propagation of loss is less frequent. A batch size of '128', being the middle value, balances the convergence speed and training stability.



(a) Training and validation loss/MAE on different masking ratios

(b) Training and validation loss/MAE with different loss functions.

Figure 7.2: Comparing ablation results for masking ratio and loss function.

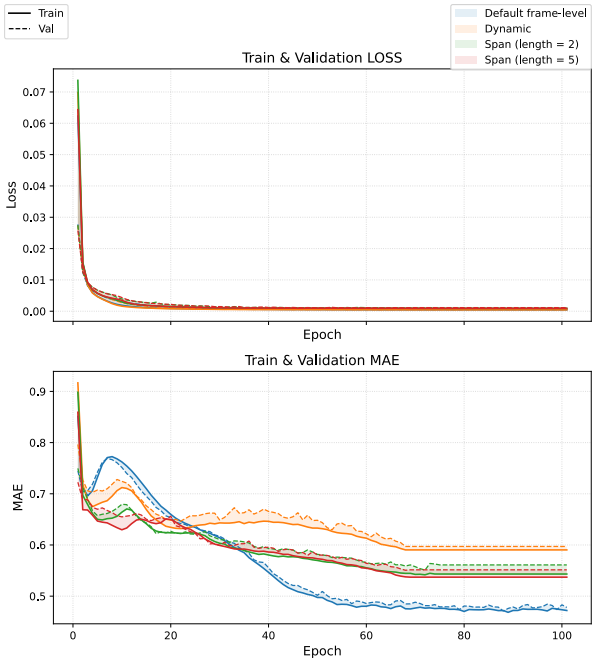
Observing Figure 7.2a, a trend could be noticed where increasing the masking ratio also increases the loss/$train - \& - val$ gap, and counterintuitively, the opposite was observed in the MAE plot. This strange behaviour could be narrowed down to the following reasons:

- A higher masking ratio significantly increases the number of masked tokens per batch. This also increases the likelihood that each batch will contain hard-to-predict trajectory events, such as *'Ego merging into an occupied lane'* and *'Cut-out in front of ego vehicle'* (see Table 9.9), at the masked position. These behave as outliers when the model attempts to predict them, leading to a higher MSE. The paper [38] proposes using the RMSE-to-MAE ratio for outlier detection. The effect of these outliers was apparent when the batch-level $RMSE/MAE$[1] ratio was plotted. Model with 90% masking showed ratios between $2.8 - 3.4$, compared to the $1.4 - 1.6$ range observed with 30% masking (see Figure 9.8). MSE penalises errors quadratically; therefore, occasional outliers (significant errors) dominate the training loss, even when the majority of predictions are accurate.

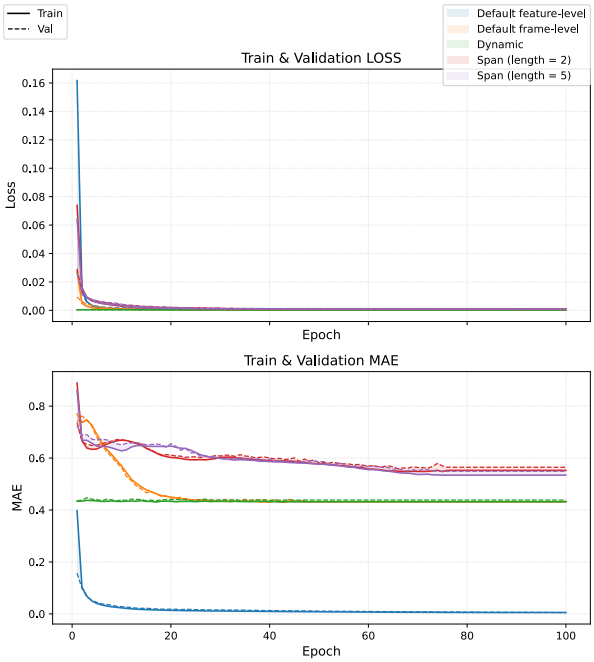
- MAE, being a linear error metric, is less sensitive to these outliers. 90% masking derived model with the steeper learning curve, causing it to have richer supervision, resulting in lower MAE on the masked positions.

In Figure 7.2b, model training with MSE as a loss function and MAE curve showed high instability compared to when using Huber loss. This instability is caused by high MSE from outliers, leading to drastic weight changes and an increased risk of overfitting.

[1]Higher values indicate that the model has a few significant errors, often due to outliers, which inflate the RMSE relative to the more evenly distributed errors measured by MAE.



(a) Training and validation loss/MAE on different masking methods



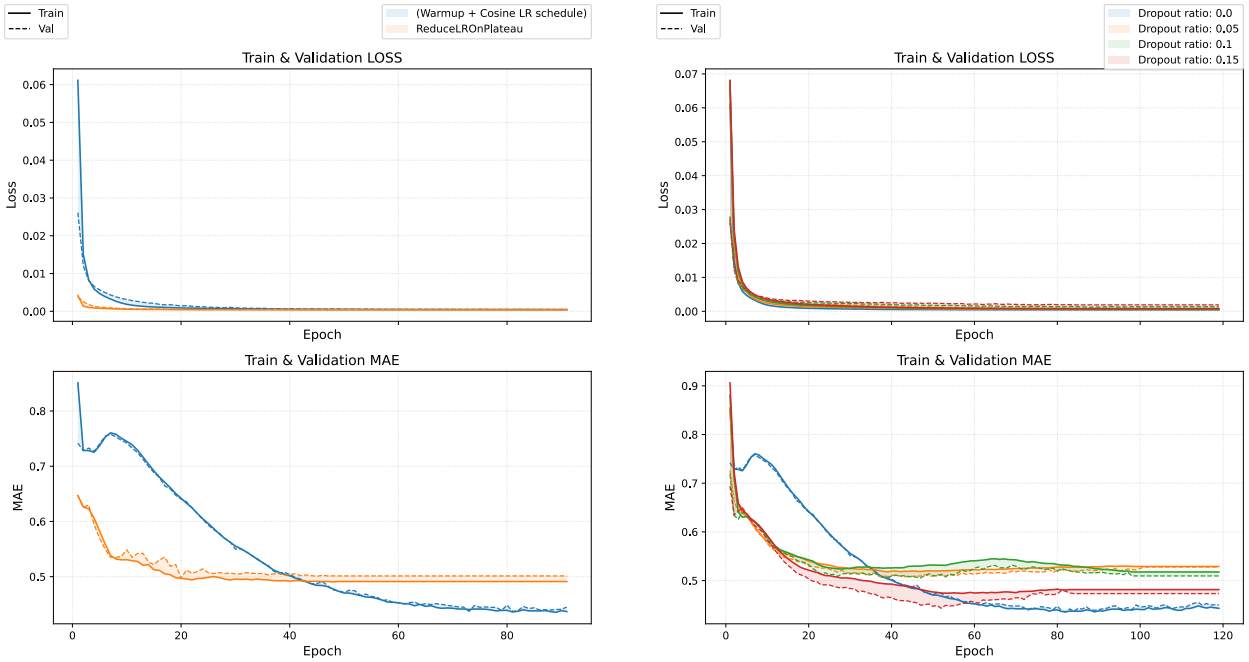
(b) Default feature-level masking included in Figure 7.3a

Figure 7.3: Comparing ablation results for masking method.

In Figure 7.3, an ablation was performed on the following masking methods:

- Default: randomly masked 15% of the input sequence (either frame or feature-level) while the masked tokens remain the same throughout training, meaning masking locations will remain static irrespective of epoch.
- Dynamic: randomly masked 15% of the input sequence while the pattern of masking is dynamic per epoch, meaning masking locations will change after each epoch.
- Span: it is similar to the default masking, except it masks the contiguous tokens as per the span length.

Figure 7.3a shows that all methods lead to similar losses. In Figure 7.3a, the default frame-level masking method converged to the lowest MAE and maintained it throughout training. The introduction of the default feature-level masking in Figure 7.3b, shows that it converged early and sustained the lowest MAE. It shows that fine-grained masking helps the model learn context more effectively, resulting in faster convergence and improved generalisation.

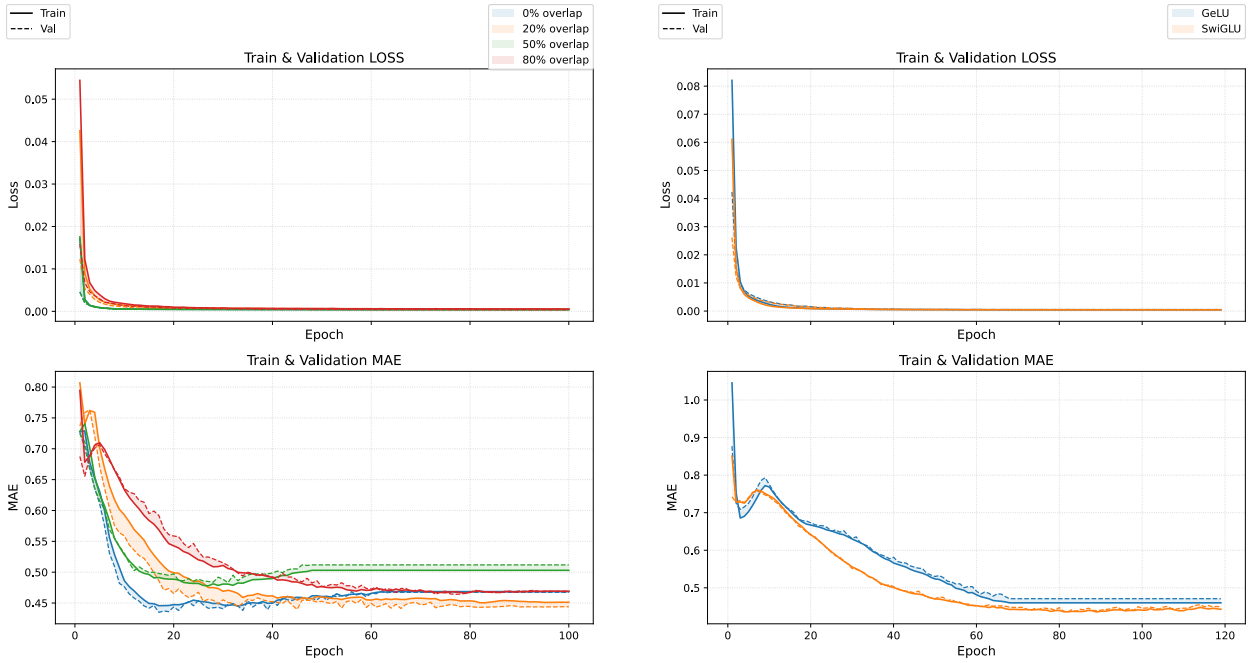


(a) Training and validation loss/MAE with different learning rate decay methods

(b) Training and validation loss/MAE on different dropout ratios

Figure 7.4: Comparing ablation results for learning rate decay method and dropout ratio.

From Figure 7.4a, it can be seen that employing $(Warmup + CosineLRschedule)$ helps the model to converge to a lower MAE. On the other hand, using *ReduceLROnPlateau* plateaus MAE early. Ablation on dropout ratio in Figure 7.4b shows that a higher ratio diminished the performance. This is mainly because a smaller model size is less prone to overfitting and does not require higher dropout layers. During pre-training of the larger BERT, keeping model generalisation in mind, a dropout of 0.1 will be used.



(a) Training and validation loss/MAE with different scenario overlapping rate.

(b) Training and validation loss/MAE with different activation functions.

Figure 7.5: Comparing ablation results for scenario overlapping rate and activation function.

From Figure 7.5a, it can be observed that the model trained on 20% overlapping sequences generalised better and achieved the lower MAE. Training MLM on overlapping sequences helps the model better understand the context of the initial and later frames of the sequence. The reason behind this is that if a $sequence_x$ ends with $frame_x$, then the consecutive $sequence_y$ will have $frame_x$ within it, enabling the model to understand the context of $frame_x$ with reference to preceding and succeeding frames.

Recent advancements have been made in the activation function for FFNs in transformer-based architectures. [39] shows how replacing standard ReLU/GeLU with GLU variants improves the transformer's pre-training and downstream performance. Though in the paper, the author tested performance on T5 (encoder-decoder) rather than on a vanilla encoder such as BERT. This made it crucial to determine whether the performance translates to BERT. Figure 7.5b shows SwiGLU (GLU's variant) performs better than GeLU for BERT pre-training. Apart from this, experimenting with sparse attention did not yield a significant improvement in performance or computational efficiency compared to self-attention.

Based on the observations, the BERT model with the following hyperparameter values performed the best in their respective ablation study: [$sequence\_length : 64$, $batch\_size : 128$, $masking\_ratio : 0.9$, $masking\_method : feature{-}level$, $dropout\_ratio : 0.0$, $loss\_function : Huber$, $overlapping\_sequence\_ratio : 0.2$, $LR\_scheduling : warmup + CosineLRschedule$, $activation\ f\ (FFN\ in\ encoder) : SwiGLU$].
Most of these values would be used for pre-training, except [$sequence\_length$, $batch\_size$, $dropout\_ratio$, $masking\_method$]. As already established, curriculum learning will be used for pre-training. This will change $sequence\_length$ and $batch\_size$ per learning stage. Instead of 0.0 $dropout\_ratio$, it was decided to keep it at 0.1, as a larger model size demands stronger regularisation than a smaller ablation model. Finally, the frame-level masking method aligns more with the downstream task of per-frame classification. Additionally, feature-level masking brings more implementation complexity.

7.2.2 Fine-tuning

Table 9.12 shows that the BERT pre-trained on 90% masked data, when fine-tuned for the downstream task, performed better on AUPRC and F1-score compared to BERT pre-trained on 30%. This shows that, with

a higher masking ratio, the model acquired representations that are much better at distinguishing whether a label is present. Moreover, adding dropout and replacing GeLU with SwiGLU improved the performance. Apart from this, to confirm the superiority of curriculum-style pre-training over pre-training with a fixed sequence length, a BERT was pre-trained with $sequence\_length : 64$, $and\ batch\_size : 128$ (best values from the pre-training ablation, see Table 6.3) and fine-tuned with the same configuration as curriculum-style pre-trained BERT for fair comparison. It was found that curriculum-based learning improved AUPRC and F1 scores by $\approx 3-6\%$ compared to the non-curriculum method.

Based on the observations, the BERT model pre-trained with [$curriculum\ manner$, $masking\_ratio : 0.9$, $masking\_method : frame-level$, $dropout\_ratio : 0.1$, $loss\_function : Huber$, $overlapping\_sequence\_ratio : 0.2$, $LR\_scheduling : warmup + CosineLRschedule$, $activation\ function\ (FFN\ in\ encoder) : SwiGLU$] performed the best with the following fine-tunning hyperparameter values: [$sequence\_length : 192$, $batch\_size : 4$, $LoRA\ rank : 4$, $alpha : 4$].

It was also observed that the BERT pretrained with half the training steps performed equivalently to the larger BERT under the same hyperparameter configuration (performance: $BERT_{50{,}784\ steps} \equiv BERT_{101{,}568\ steps}$). This signals that the norm of pre-training LLMs for the maximum number of training steps to produce better weights does not translate to MLMs such as BERT, which are pre-trained on trajectory data. This suggests that a stopping criterion could be applied during BERT's pre-training to save time, memory and compute. Further research could identify appropriate stopping criteria to limit the execution of additional redundant training steps.

7.3 BERT vs BiLSTM (baseline)

From Table 7.2, it becomes evident that BERT outperforms BiLSTM. BERT performed better on 4 of 5 metrics, improving $\approx$ 26.3% in macro F1-score, 0.6% in hamming loss, 9.4% in $\text{NMABE}_{average}$, 1% in $\text{MMR}_{average}$ with $\approx$ 94% less time and $\approx$ 68% less trainable parameters.

Table 7.2: Performance of BERT compared to baseline- BiLSTM, on a test set of 41,689 non-overlapping sequences of size 25 (created from trajectories of 1,908 unique vehicles). Both models are run three times with different seed values to report $mean \pm standard\ deviation$. The coefficient of variation across all configurations and two evaluation metrics is less than 0.05, suggesting model reliability. $mean \pm standard\ deviation$ suggests the best value for the respective evaluation metric.

Metric	**BiLSTM**	**BERT**
Macro F1-score ↑	0.38 ± 0.007	0.48 ± 0.009
Hamming loss ↓	0.073 ± 0.0	0.067 ± 0.001
$\text{NMABE}_{average}$ ↓	0.178 ± 0.008	0.084 ± 0.007
$\text{MMR}_{average}$ ↓	0.37 ± 0.016	0.36 ± 0.012
$\text{MMR}_{ST_{average}}$ ↓	0.77 ± 0.014	0.89 ± 0.017
Training time (mins)	≈ 85	≈ 5

BiLSTM had 341,516 trainable parameters, whereas BERT had 108,892. Making BERT yield better performance on 4/5 metrics converging in $\approx$ 94% less time and with $\approx$ 68% less trainable parameters.

Table 7.3: Average duration (seconds) of scenario categories from a sample of 41,689 seconds of driving data, along with their F1-score calculated using BiLSTM and BERT with best configurations. 'Vehicles' column signifies the number of vehicles in which the particular scenario category occurs.

Scenario categories	**Average duration(s)↑**	**Vehicles**	**F1-score$_{\text{BiLSTM}}$**	**F1-score$_{\text{BERT}}$**
Vehicle overtaking ego vehicle	3.06	1352	0.61	0.59
Ego vehicle overtaking vehicle in adjacent lane	3.32	1265	0.37	0.57
Cut-in in front of ego vehicle	3.3	130	0.03	0.04
Cut-out in front of ego vehicle	3.52	124	0.02	0.11
Ego vehicle performing lane change	3.88	279	0.65	0.69
Lead vehicle cruising	5.28	3040	0.69	0.71
Ego vehicle approaching slower lead vehicle	6.8	201	0.47	0.43
Ego vehicle driving in lane without lead vehicle	6.88	2655	0.84	0.85
Lead vehicle decelerating	6.91	588	0.23	0.34
Lead vehicle accelerating	7.56	696	0.37	0.47

(↑) indicates column sorted in ascending order.

From Table 7.3, it can be noted that BERT performs better on 8 of 10 scenario categories. Overall, the improvement is not much; however, considering how little time and parameters it took to fine-tune a pre-trained BERT and achieve slightly better performance on 4/5 metrics is notable. This shows that, if a vehicle trajectory pre-trained model is available, using it as a foundation model and fine-tuning it for a downstream task takes less time and resources than training a conventional ML model.

Chapter 8

Conclusion and Future Work

This thesis aims to evaluate the ability of sequential ML models, such as LSTM, BiLSTM, and BERT, to classify vehicle trajectories into multi-label driving scenario categories. A rule-based method was used to generate labelled data for (Bi)LSTM and BERT-fine-tuning training. The thesis proposes evaluating models from two different perspectives: scenario category classification and transition identification. A total of five evaluation metrics were used for model comparison: macro F1-score, hamming loss, $\text{NMABE}_{average}$, $\text{MMR}_{average}$, $\text{MMR}_{ST_{average}}$. An ablation study was performed for each of the three models to understand the impact of different hyperparameters and get the optimal configuration. LSTM and BiLSTM were initially implemented. After the ablation study, BiLSTM, which performed better, was selected as the baseline against the proposed BERT-based classifier.

An ablation study on BERT's pre-training setup showed that a masking probability of 90% outperformed the standard 15% and 30%. It produced smoother convergence of the training and validation curves, leading to lower MAE and better downstream classification after fine-tuning. A plausible explanation is that higher masking forces the model to learn from a steeper learning curve with stronger supervision per observed token, which, in turn, reduces MAE at masked positions. Additionally, it was found that the model, pre-trained or fine-tuned on overlapping trajectory sequences, better understands the context of the initial and later frames of a sequence. In the ablation study, a 20% overlap rate for a 64-length sequence was found optimal.

Regarding the activation function for the FFN in the encoder, SwiGLU performed better than GeLU. Furthermore, learning rate scheduling of $warmup{+}CosineLRschedule$ performed better than $ReduceLROnPlateau$. Moreover, the use of a dropout layer helped the model generalise and prevented overfitting. Lastly, it was found that the curriculum-style pre-training—conducted in three stages with a gradual increase in the model's exposure to longer sequences—performed $\approx 3-6\%$ better on the downstream classification task than pre-training with a fixed sequence length.

BERT was pre-trained using masked frame modelling for $101,568$ training steps in a curriculum-based, three-stage approach. The pre-trained model was then fine-tuned using LoRA. It performed slightly better than the BiLSTM (baseline) on 4 of 5 evaluation metrics and achieved a higher F1 score on 8 of 10 scenario categories. Demonstrating that it improved the classification and identification of transitions of driving scenario categories. However, BiLSTM outperforms on $\text{MMR}_{ST_{average}}$, suggesting that it is more effective at recognising smooth transitions. Additionally, BERT fine-tuning took $\approx 94\%$ less time and $\approx 68\%$ fewer trainable parameters. Showing the efficiency of transfer learning over the conventional training method. This shows that, if a vehicle trajectory pre-trained model is available, using it as a foundation model and fine-tuning it for a downstream task takes less time and resources than training a conventional ML model.

Further research could be done to identify appropriate stopping criteria that limit the execution of additional redundant training steps when pre-training MLMs (specifically, BERT based on vehicle trajectory). Ultimately saving time, memory and computing. Additionally, future work could involve generating or recording vehicle trajectory data alongside exogenous factors such as weather, lighting, and the driver's vision and physical health. Further, use it to pre-train BERT and see if it improves performance on the downstream task of multi-label classification of scenario categories. Moreover, highD fails to record the following vehicle's velocity/acceleration; further research could be conducted to understand its importance for the ego vehicle's

behaviour. Along the same lines, the distribution of feature importance across scenario categories could be studied, and further understanding of how preceding or succeeding scenario categories affect the current one could be gained.

Bibliography

[1] SAE International. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles, April 2021. Revised version issued April 2021.

[2] Oliver Wyman. Mobility investment radar 2025: Mobility trends, February 2025. Accessed: 2025-06-30.

[3] Erwin de Gelder and Jan-Pieter Paardekooper. Assessment of automated driving systems using real-life scenarios. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 589–594, 2017.

[4] Jacob Langner, Hannes Grolig, Stefan Otten, Marc Holzäpfel, and Eric Sax. Logical scenario derivation by clustering dynamic-length-segments extracted from real-world-driving-data. 05 2019.

[5] Philip Elspas, Yannick Klose, Simon T Isele, Johannes Bach, and Eric Sax. Time series segmentation for driving scenario detection with fully convolutional networks. In *VEHITS*, pages 56–64, 2021.

[6] Armstrong Aboah, Yaw Adu-Gyamfi, Senem Velipasalar Gursoy, Jennifer Merickel, Matt Rizzo, and Anuj Sharma. Driver maneuver detection and analysis using time series segmentation and classification, 2022.

[7] Shirui Zhou, Jiying Yan, Junfang Tian, Tao Wang, Yongfu Li, and Shiquan Zhong. A driving regime-embedded deep learning framework for modeling intra-driver heterogeneity in multi-scale car-following dynamics, 2025.

[8] Yuhuan Lu, Pengpeng Xu, Xinyu Jiang, Ali Bashir, Thippa Gadekallu, Wei Wang, and Xiping Hu. Lane change prediction for autonomous driving with transferred trajectory interaction. *IEEE Transactions on Intelligent Transportation Systems*, PP, 02 2025.

[9] Yongwei Li, Yongzhi Jiang, and Xinkai Wu. Trajpt: A trajectory data-based pre-trained transformer model for learning multi-vehicle interactions. *Transportation Research Part C: Emerging Technologies*, 171:105013, 2025.

[10] Robert Krajewski, Julian Bock, Laurent Kloeker, and Lutz Eckstein. The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2118–2125, 2018.

[11] Erwin de Gelder, Jan-Pieter Paardekooper, Arash Khabbaz Saberi, Hala Elrofai, Olaf Op den Camp, Steven Kraines, Jeroen Ploeg, and Bart De Schutter. Towards an ontology for scenario definition for the assessment of automated vehicles: An object-oriented framework. *IEEE Transactions on Intelligent Vehicles*, 7(2):300–314, June 2022.

[12] Xinhai Zhang, Jianbo Tao, Kaige Tan, Martin Törngren, José Manuel Gaspar Sánchez, Muhammad Rusyadi Ramli, Xin Tao, Magnus Gyllenhammar, Franz Wotawa, Naveen Mohan, Mihai Nica, and Hermann Felbinger. Finding critical scenarios for automated driving systems: A systematic mapping study. *IEEE Transactions on Software Engineering*, 49(3):991–1026, 2023.

[13] Zhiyuan Wei, Hanchu Zhou, and Rui Zhou. Risk and complexity assessment of autonomous vehicle testing scenarios. *Applied Sciences*, 14(21), 2024.

[14] Yu Zhu, Jian Wang, Fanqiang Meng, and Tongtao Liu. Review on functional testing scenario library generation for connected and automated vehicles. *Sensors*, 22(20), 2022.

[15] Ziyuan Zhong, Yun Tang, Yuan Zhou, Vania de Oliveira Neves, Yang Liu, and Baishakhi Ray. A survey on scenario-based testing for automated driving systems in high-fidelity simulation, 2021.

[16] J. Ploeg, E. de Gelder, M. Slavík, E. Querner, T. Webster, and N. de Boer. Scenario-based safety assessment framework for automated vehicles, 2021.

[17] Erwin de Gelder. *Safety assessment of automated vehicles using real-world driving scenarios*. PhD thesis, Delft University of Technology, 2022.

[18] Emmanouil Barmpounakis and Nikolas Geroliminis. On the new era of urban traffic monitoring with massive drone data: The pneuma large-scale field experiment. *Transportation Research Part C: Emerging Technologies*, 111:50–71, 2020.

[19] Tao Yang and Shengwu Xiong. Lane-aware transformers for multi-agent trajectory prediction. In *Proceedings of the 2023 5th International Conference on Internet of Things, Automation and Artificial Intelligence*, IoTAAI '23, page 659–664, New York, NY, USA, 2024. Association for Computing Machinery.

[20] Jiquan Ngiam, Benjamin Caine, Vijay Vasudevan, Zhengdong Zhang, Hao-Tien Lewis Chiang, Jeffrey Ling, Rebecca Roelofs, Alex Bewley, Chenxi Liu, Ashish Venugopal, David Weiss, Ben Sapp, Zhifeng Chen, and Jonathon Shlens. Scene transformer: A unified architecture for predicting multiple agent trajectories, 2022.

[21] U.S. Department of Transportation Federal Highway Administration. Next generation simulation (NGSIM) vehicle trajectories and supporting data. Provided by ITS DataHub through Data.transportation.gov, 2016. Accessed: February 25, 2026.

[22] Laurence R. Rilett, Fred Choobineh, Elizabeth Jones, et al. Nebraska naturalistic driving study. Technical report, Mid-America Transportation Center, University of Nebraska-Lincoln, 2010. Project Report.

[23] Federal Highway Administration. Strategic highway research program 2 (SHRP2) naturalistic driving study (NDS). Roadway Safety Data Program (RSDP), 2018. Accessed: February 25, 2026.

[24] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[25] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016.

[26] Bowen Cheng, Yunchao Wei, Rogerio Feris, Jinjun Xiong, Wen mei Hwu, Thomas Huang, and Humphrey Shi. Decoupled classification refinement: Hard false positive suppression for object detection, 2020.

[27] Gen Li, Zhen Yang, Yiyong Pan, and Jianxiao Ma. In-depth analysis of durations of discretionary lane changes on freeway under varying traffic conditions, 2022.

[28] Zhaohan Wang and Yang Gao. Lane change inconsistencies in the highd dataset. *Findings*, 03 2025.

[29] Friedrich Kruber, Jonas Wurst, Samarjit Chakraborty, and Michael Botsch. Highway traffic data: macroscopic, microscopic and criticality analysis for capturing relevant traffic scenarios and traffic modeling based on the highd data set, 2019.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[31] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[32] Benjamin Clavié, Nathan Cooper, and Benjamin Warner. It is all in the [mask]: Simple instruction-tuning enables bert-like masked language models as generative classifiers. *Natural Language Processing Journal*, 11:100150, 2025.

[33] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[34] Luca Rossi, Andrea Ajmar, Marina Paolanti, and Roberto Pierdicca. Vehicle trajectory prediction and generation using lstm models and gans. *PLOS ONE*, 16(7):1–28, 07 2021.

[35] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022.

[36] Yixing Fan, Xiaohui Xie, Yinqiong Cai, Jia Chen, Xinyu Ma, Xiangsheng Li, Ruqing Zhang, and Jiafeng Guo. Pre-training methods in information retrieval, 2022.

[37] Daniel Frees, Aditri Bhagirath, and Moritz Bolling. Exploring efficient learning of small bert networks with lora and dora, 2025.

[38] Dulakshi Santhusitha Kumari Karunasingha. Root mean square error or mean absolute error? use their ratio as well. *Information Sciences*, 585:609–629, 2022.

[39] Noam Shazeer. Glu variants improve transformer, 2020.

[40] Erwin de Gelder, Maren Buermann, and Olaf Op Den Camp. Coverage metrics for a scenario database for the scenario-based assessment of automated driving systems. In *2024 IEEE International Automated Vehicle Validation Conference (IAVVC)*, page 1–8. IEEE, October 2024.

[41] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[43] Jay Alammar. The illustrated transformer. `http://jalammar.github.io/illustrated-transformer/`, 2018. Accessed: 2025-11-24.

Chapter 9

Appendix

This appendix contains entities that would have taken up too much space to include in the main thesis. For convenience, the appendix is categorised into (i) algorithm, (ii) tables, (iii) figures, (iv) model/data pipeline, (v) ablation study- tables/line graphs.

9.1 Algorithms

Algorithm 1: Average of scenario categories excluding all-zero groups

Input: DataFrame D with columns: identifier *vehicle id* and binary encoded scenario categories $b_1, b_2, \ldots, b_m$

Output: Mean values per scenario category

1. Group D by identifier *vehicle id* to form groups $G_1, G_2, \ldots, G_n$;
2. For each group G_i:
   For each scenario category b:
   compute $s_{ij} = \sum_{j=1}^{m} b_j$ (the total number of 1s in column b_j for group G_i);
   If $s_{ij} = 0$, set $s_{ij} \leftarrow \text{NA}$ (ignore this entry);
3. For each scenario category b_j:
   compute column mean

$$\mu_j = \frac{\sum_{i=1}^{n} s_{ij}}{\#\{i \mid s_{ij} \neq \text{NA}\}}$$

Algorithm 2: Data preprocessing pipeline for (Bi)LSTM

Input: highD CSV data
Output: Scaled train, val, and test data with valid_mask
Step 1: Data cleaning and direction normalization
  Remove rows where all scenario category labels are zero
  if $xVelocity < 0$ **then**
    $xVelocity \leftarrow -xVelocity$
    Flip the signs of features to maintain consistency across the data.
  end
  Remap `laneId` to a consistent numerical representation
  Compute `valid_mask` (exclude overlapping mutually exclusive scenario categories)
  Compute `boundary` (1 if $y_t \neq y_{t-1}$, else 0)
end
Step 2: Feature engineering and indicators
  Binary encoding of features containing vehicle ID ($Id > 0 \rightarrow 1$)
end
Step 3: Sequence generation and splitting
  Group data by `(document_id, vehicle_id)`
  Generate sliding window sequences of length$\rightarrow$ 25
  Perform $train : 70\%,\ val : 15\%,\ test : 15\%$ random split on the generated sequences.
end
Step 4: Leakage-free scaling
  Fit `StandardScaler` S_{main} on X_{train} (non-binary features).
  Fit `StandardScaler` S_{prec} on X_{train} ('precedingXVelocity') **only** where it is non-null.
  Apply S_{main} and S_{prec} to Train, Val, and Test sets.
end

Algorithm 3: $\text{MMR}_{average}$

Input: Ground truth sequences Y_{true}, predicted sequences Y_{pred}, set of labels L, and vehicle identifiers V
Output: Average miss rate across all classes

1. Group Y_{true} and Y_{pred} by identifier *vehicle id* to form groups $V_1, V_2, \ldots, V_n$;
2. For each group V_i (vehicle):
   For each label $l \in L$:
   If class l occurs in $Y_{true}^{(i)}$, increment $support_l$ by 1;
   If $Y_{pred}^{(i)}(l)$ contains no positive frames, increment $missed_l$ by 1;
3. For each label $l \in L$:
   Compute miss rate per class:
$$\text{MMR}_l = \frac{missed_l}{support_l}$$
4. Compute overall average miss rate:
$$\text{MMR}_{average} = \frac{1}{|L|} \sum_{l \in L} \text{MMR}_l$$

Algorithm 4: $\text{MMR}_{ST_{average}}$

Input: Ground truth sequences Y_{true}, predicted sequences Y_{pred}, set of smooth transition pairs $\mathcal{P} = \{(p, c)\}$, and vehicle identifiers V

Output: Average missed transition identification rate across all classes

1. Group Y_{true} and Y_{pred} by identifier *vehicle id* to form groups $V_1, V_2, \ldots, V_n$;
2. For each group V_i (vehicle):
   For each transition pair $(p, c) \in \mathcal{P}$:
   Identify true transition instances where parent p ends and child c begins ($Y^{(i)}_{true}$), Increment $support_{(p,c)}$ by $|instances|$;
   Identify predicted transition instances ($Y^{(i)}_{pred}$);
   If fewer predicted transitions ($Y^{(i)}_{pred}$) of type (p, c) are found, increment $missed_{(p,c)}$ by the difference.;
3. For each label $(p, c) \in \mathcal{P}$:
   Compute miss rate per transition pair:

$$\text{MMR}_{ST_{(p,c)}} = \frac{missed_{(p,c)}}{support_{(p,c)}}$$

4. Compute overall average miss rate:

$$\text{MMR}_{ST_{average}} = \frac{1}{|\mathcal{P}|} \sum_{(p,c)\in\mathcal{P}} \text{MMR}_{ST_{(p,c)}}$$

Algorithm 5: TFRecord sharding and benchmarking for BERT pre-training

Input: Parquet directory P, TFRecord output directory T, sequence lengths $S = \{64, 128, 192\}$, batch sizes $B = \{256, 128, 64\}$, number of features F, target shard memory M ($unit : MB$).
Output: Sharded TFRecords, and benchmark results.

1. Load data from parquet files
  foreach $seq\_len \in S$ **do**
    Extract sequences with a sliding window, and a stride of length seq_len across all features F.
    Store these non-overlapping sequences in $all\_sequences[seq\_len]$.
  end
end
2. Compute sharding strategy and write TFRecords
  $target\_bytes \leftarrow M \times 1024 \times 1024$
  foreach $seq\_len \in S$ **do**
    $memory\_per\_sequence \leftarrow seq\_len \times F \times bytes\_per\_float$
    $seq\_per\_shard \leftarrow \lceil target\_bytes / memory\_per\_sequence \rceil$
    $num\_shards \leftarrow \lceil total\_sequences / seq\_per\_shard \rceil$
    Partition $all\_sequences[seq\_len]$ into num_shards
  end
  foreach i **do**
    write sequences to TFRecord file
  end
end
3. Build dataset pipeline
  Parse serialised TFRecords into fixed-length feature vector (usable tensors).
end
4. Benchmark loading performance
  foreach $(seq\_len, batch\_size) \in (S, B)$ **do**
    Load dataset
    Measure execution time and report throughput (batches/sec)
  end
end

Algorithm 6: End-to-end data pipeline: From raw CSV to multi-scale TFRecords

Input: highD CSVs: training, fine-tuning, and testing CSVs, $seq\_sizes \in \{64, 128, 192\}$
Output: Preprocessed parquet files, and serialised TFRecords
Phase 1: Global scaling (leakage prevention)
  Initialize `StandardScaler` for independent vehicle features
  foreach *Training file* **do**
    Normalize driving direction: If $v_x < 0$, flip signs of $-ve$ features
    `partial_fit` scalers on current file data `// partial_fit mitigates OOM error`
  end
end
Phase 2: Parquet creation & feature preprocessing
  foreach *dataset (Train, Fine-tune, Test)* **do**
    Apply direction normalization and `laneId` remapping
    Binary encoding of features containing vehicle ID ($ID > 0 \rightarrow 1$)
    Transform features using global scalers
    if *generating file for BERT pre-training* **then**
      Save 20-feature matrix to `.parquet` with integer headers
    end
    else if *generating file for fine-tuning the pre-trained BERT or testing the fine-tuned model* **then**
      Stack [IDs (3), Features (20), Labels (12)] into combined `parquet`
    end
  end
end
Phase 3: Multi-scale TFRecord sharding
  Partition data by unique `(document_id, vehicle_id)`
  70%/15%/15% split at the vehicle level to ensure no data leakage
  foreach *split* $\in$ *{Train, Val, Test}* **do**
    foreach *size* $\in$ seq_sizes **do**
      Extract windows of length *size* from each vehicle trajectory using sliding window logic
      Apply 0-padding to reach max_seq_len (192)
      Generate `valid_mask` (1 for active frames, 0 for padding)
      `Serialize` to `tf.train.Example` with float features
      Write to shards (Target: 20MB per shard for I/O efficiency)
    end
  end
end

Algorithm 7: BERT pre-training via masked frame modelling

Input: Multi-size TFRecords, masking ratio $mask\_prob$, number of epochs E
Output: Pre-trained BERT model weights
Build `BERT`: Transformer-based encoder with positional embeddings and FFN
Define `WarmUpCosine` scheduler with 10% linear warmup (fixed training steps)
Curriculum learning strategy:
  Stage 1: Sample from $\{64\}$ length records
  Stage 2: Sample from $\{64, 128\}$ with weights $[0.7, 0.3]$
  Stage 3: Sample from $\{64, 128, 192\}$ with weights $[0.5, 0.3, 0.2]$
end
foreach E **do**
  Select current stage based on training progress
  foreach *batch in curriculum stage's dataset* **do**
    $P \leftarrow$ Padding mask for current batch
    Generate random mask M where $M_{i,j} < mask\_prob \wedge P_{i,j} == 1$
    $X_{masked} \leftarrow X \odot (1 - M)$ `// Zero-out masked frames`
    $\hat{Y} \leftarrow \text{Model}(X_{masked}, P)$ `// Forward pass with padding mask`
    $Loss \leftarrow loss\_f(\hat{Y}, X, \text{weight} = M)$ `// Loss on masked frames only`
    Update weights via `AdamW` with Mixed Precision
  end
end

Algorithm 8: Fine-tuning and evaluating BERT for multi-label driving scenario category classification using LoRA

Input: Labelled TFRecords, pre-trained BERT Weights, LoRA rank r, Alpha α
Output: Multi-label fine-tuned model, Macro F1-score, and Hamming loss
Phase 1: LoRA parameter adaptation
  Load Pre-trained BERT Model
  foreach *Dense Layer in Encoder* **do**
    Freeze original weights $W \in R^{d \times k}$
    Initialize LoRA adapters: $A \in R^{d \times r}$ (Normal) and $B \in R^{r \times k}$ (Zeros)
    Define LoRA output: $h = xW + \frac{\alpha}{r}(xAB)$
  end
  Attach a Classification Head: $Dense$(units=12, activation='sigmoid')
end
Phase 2: Supervised training
  Initialize WarmUpCosine schedule with $base\_lr$, min_lr, and $warmup\_steps$
  Compile using masked binary cross-entropy to ignore padded frames
  `Loss` $= \frac{\sum(\text{BCE}(y,\hat{y}) \cdot \text{valid\_mask})}{\sum \text{valid\_mask}}$
  Train using `AdamW`; apply Early stopping on *validation loss*
end
Phase 3: Threshold optimization & evaluation
  Predict probabilities $\hat{y}$ on Test Set
  foreach *Scenario Category* $i \in [1, 12]$ **do**
    Iterate $t \in [0.1, 0.9]$ to find threshold maximizing $F1_i$
  end
  Apply best thresholds to calculate final **Macro F1** and **Hamming Loss**
end

9.2 Tables

Table 9.1: Common criticality measures for driving scenarios [12]

Type	**Example Metric**	**Indicating**
Spatial	Minimum Distance	Distance remaining to hazard
Temporal	TTC, PET	TTC: Time remaining to collision assuming physical entities remain in their constant state. PET: Time gap between one actor leaving the conflict region and another entering.
Severity	Impact Speed	Seriousness of the impact.
System Response	Evasive Action	Emergency brakes and/or manoeuvres.
Safety Margin	Distance/Speed, DRAC	Predefined safe limits.

Table 9.2: LC direction and corresponding LC portions from classified 10,127 LCs from location 1 [1] [28]. Refer to Figure 9.4 for visuals.

Lanes 4, 5, 6 (rightward)	4 to 5	5 to 6	6 to 5	5 to 4
LC portion	0.37	0.17	0.13	0.33
Lanes 1, 2, 3 (leftward)	1 to 2	2 to 3	3 to 2	2 to 1
LC portion	0.13	0.30	0.38	0.19

Table 9.3: Scenario categories and activities detected by the TNO's StreetWise tool. Multi-label scenario category classification was performed on the highD dataset across 12 unique categories [40]. The 'Count' represents the number of sequences for a particular scenario category from a sample of 484,507 sequences created using a sliding window of 25 and a stride of 1.

Symbol	**Name**	**Ego Vehicle Activity**	**Main Actor(s) Activity**	**Count**
C1	Cut-in in front of ego vehicle.	Keeping lane	Changing lane to become leading vehicle.	2156
C2	Cut-out in front of ego vehicle.	Keeping lane, leading the ego vehicle, and then changing lane.	2954	
C3	Merging into an occupied lane.	Changing lane	Both main actors stay in their lanes and become leading and following vehicles after the ego vehicle's lane change.	811
C4	Approaching slower vehicle.	Keeping lane	Keeping lane and driving slower than ego vehicle.	15,025
C5	Ego vehicle driving in lane without lead vehicle.	Keeping lane	None	184,261
C6	Ego vehicle overtaking vehicle.	Keeping lane	Keeping lane, overtaken by ego vehicle in adjacent lane.	63,400
C7	Ego vehicle performing lane change.	Changing lane	None	11,189
C8	Changing lane with vehicle behind.	Changing lane	Behind ego vehicle on adjacent lane.	3445
C9	Leading vehicle accelerating.	Keeping lane	Keeping lane and accelerating.	40,320
C10	Leading vehicle cruising.	Keeping lane	Keeping lane and cruising.	256,027
C11	Leading vehicle decelerating.	Keeping lane	Keeping lane and decelerating.	37,028
C12	Vehicle overtaking ego vehicle.	Keeping lane	Keeping lane and overtaking ego vehicle on adjacent lane.	67,035

Table 9.4: The scenario categories listed form a mutually exclusive set: No two (or more) categories from this set can co-occur.

Mutually Exclusive Scenario Categories
{C5, C10, C9, C11}
{C7, C8, C3}

Table 9.5: '$(\frac{\text{Av. frames}}{25})$ secs' and '$[\frac{\text{count}}{484507} * 100]\%$' computes average duration and class distribution of scenario categories. See Algorithm 1 for the average frame calculation.

Scenario category	**Average Frames**	$(\frac{\textbf{Av. frames}}{\textbf{25}})$ **secs**	$[\frac{\textbf{count}}{\textbf{484507}} * \textbf{100}]\%$
Cut-in in front of ego vehicle	66	2.64	0.44
Cut-out in front of ego vehicle	86	3.44	0.16
Ego merging into an occupied lane	111	4.44	0.16
Ego vehicle approaching slower lead vehicle	156	6.24	3.1
Ego vehicle driving in lane without lead vehicle	175	7	38
Ego vehicle overtaking vehicle in adjacent lane	70	2.8	13
Ego vehicle performing lane change	91	3.64	2.3
Ego vehicle performing lane change with vehicle behind	84	3.36	0.7
Lead vehicle accelerating	163	6.52	8.3
Lead vehicle cruising	166	6.64	52.8
Lead vehicle decelerating	151	6.04	7.6
Vehicle overtaking ego vehicle	96	3.84	13.8

Computation done using a sample of 484,507 sequences or 80460 seconds of driving data created using a sliding window of 25 and a stride of 1.

Table 9.6: Identified smooth transition combinations from the 12 unique scenario categories (Table 9.3). Smooth transitions are more likely to exhibit scenario overlap around the transition point due to ambiguity. 16 unique combination pairs are forming.

Scenario A	**Scenario B (consecutive scenario to A)**
Cut-in in front of the ego vehicle (C1)	• Lead vehicle accelerating (C9) • Lead vehicle cruising (C10) • Lead vehicle decelerating (C11)
Cut-out in front of the ego vehicle (C2)	• Ego vehicle driving in lane without lead vehicle (C5) • Lead vehicle cruising (C9) • Lead vehicle decelerating (C10) • Lead vehicle accelerating (C11)
Ego merging into an occupied lane (C3)	• Lead vehicle cruising (C9) • Lead vehicle decelerating (C10) • Lead vehicle accelerating (C11)
Ego vehicle performing lane change (C7)	• Ego vehicle driving in lane without lead vehicle (C5)
Ego vehicle performing lane change with vehicle behind (C8)	• Ego vehicle driving in lane without lead vehicle (C5)
Ego vehicle driving in lane without the lead vehicle (C5)	• Ego vehicle approaching slower lead vehicle (C4) • Lead vehicle accelerating (C9) • Lead vehicle cruising (C10) • Lead vehicle decelerating (C11)

Table 9.7: Baseline model definition summary - 4 layers

Layer (type)	**Output shape**	**Param #**
Input layer	(None, 25, 20)	0
$\text{LSTM}_{stateless}$	(None, 25, 64)	21,760
$\text{LSTM}_{stateless}$ (x3)	(None, 25, 64)	99,072
$\text{Dense}_{TimeDistributed}$	(None, 25, 12)	780
Total trainable params		121,612

(a) LSTM definition

Layer (type)	**Output shape**	**Param #**
Input layer	(None, 25, 20)	0
$\text{BiLSTM}_{stateless}$	(None, 25, 128)	43,520
$\text{BiLSTM}_{stateless}$ (x3)	(None, 25, 128)	296,448
$\text{Dense}_{TimeDistributed}$	(None, 25, 12)	1,548
Total trainable params		341,516

(b) BiLSTM definition

Table 9.8: Mapping of $'right \rightarrow left'$ *lane ID to equivalent* $'left \rightarrow right'$ *lane ID* in a three lane highway.

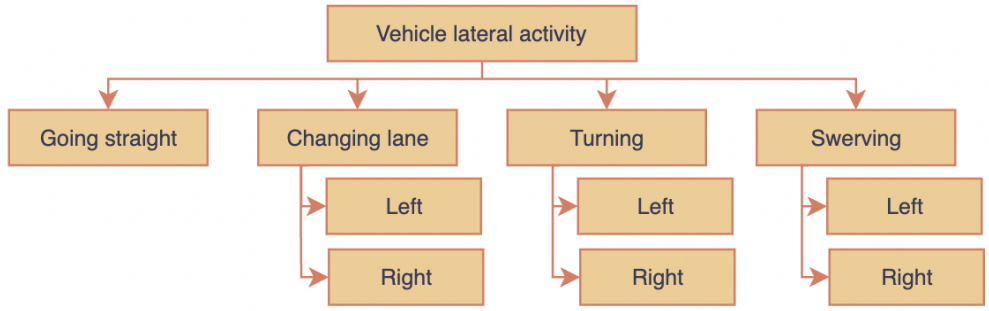
$\boldsymbol{right \rightarrow left}$ **lane ID** $\rightarrow$ **equivalent** $\boldsymbol{left \rightarrow right}$ **lane ID**
lane 1 $\rightarrow$ *lane* 6
lane 2 $\rightarrow$ *lane* 5
lane 3 $\rightarrow$ *lane* 4

Table 9.9: Average duration (seconds) of scenario categories from a sample of 41,689 seconds of driving data, along with their F1-score calculated using BiLSTM with best configurations. 'Vehicles' column signifies the number of vehicles in which the particular scenario category occurs.

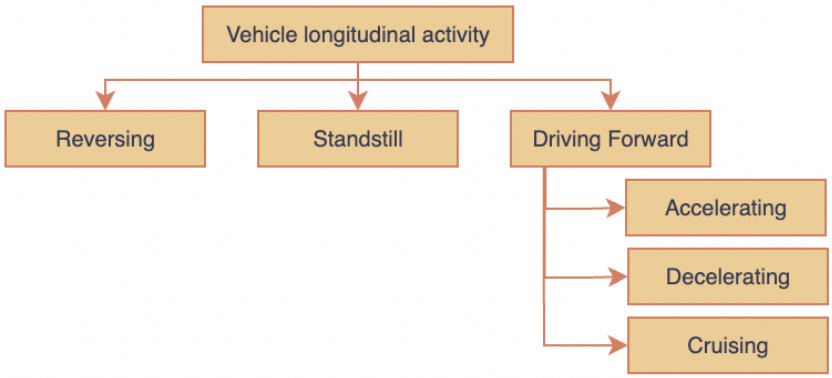
Scenario categories	**Average duration (s)** ↑	**Vehicles**	**F1-score**
Vehicle overtaking ego vehicle	3.06	1352	0.61
Ego vehicle performing lane change with vehicle behind	3.31	93	0.33
Ego vehicle overtaking vehicle in adjacent lane	3.32	1265	0.37
Cut-in in front of ego vehicle	3.3	130	0.03
Cut-out in front of ego vehicle	3.52	124	0.02
Ego vehicle performing lane change	3.88	279	0.65
Ego merging into an occupied lane	5.11	20	0.00
Lead vehicle cruising	5.28	3040	0.69
Ego vehicle approaching slower lead vehicle	6.8	201	0.47
Ego vehicle driving in lane without lead vehicle	6.88	2655	0.84
Lead vehicle decelerating	6.91	588	0.23
Lead vehicle accelerating	7.56	696	0.37

(↑) indicates column sorted in ascending order.

9.3 Figures



(a) Lateral activities of a vehicle.



(b) Longitudinal activities of a vehicle.

Figure 9.1: Lateral and longitudinal activities of a vehicle [11]

. .

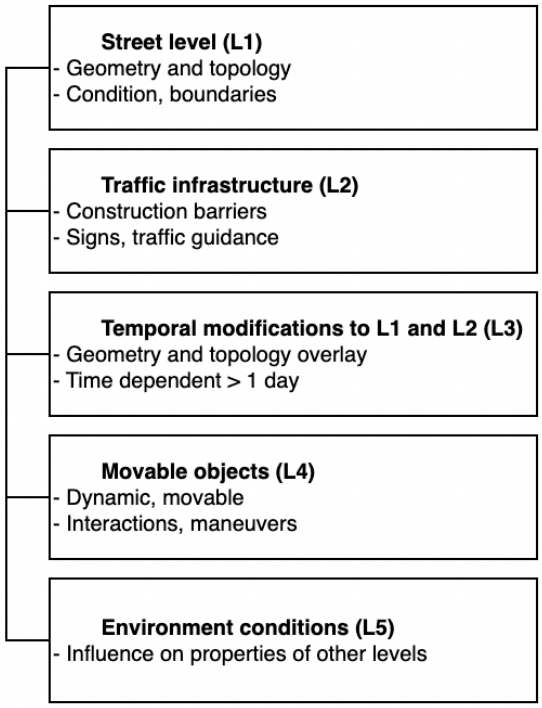
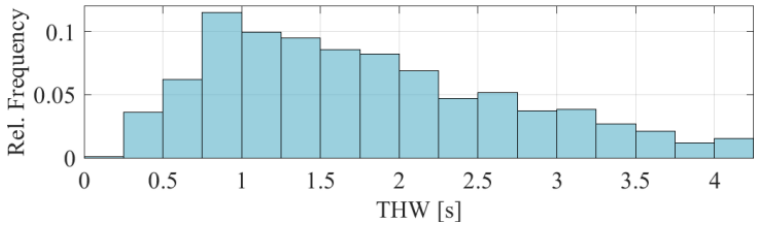
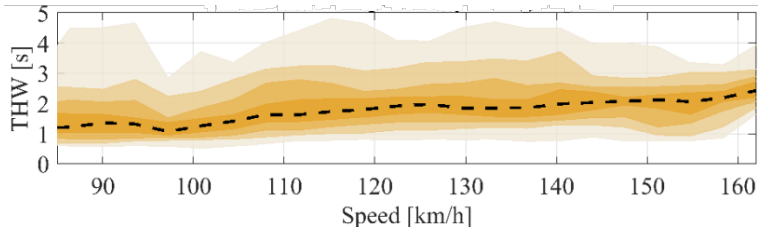


Figure 9.2: 5 Layer model defining scenario description [10]



(a) Cut-in THW distribution



(b) Cut-in THW dependency on ego vehicle's speed

Figure 9.3: (a) Distribution of the tailing vehicle's THW to a lane-changing vehicle at the time it enters the tailing vehicle's lane. (b) The dependency of this THW on the tailing vehicle's speed. The dashed line shows the median, while the shades indicate deciles [10].
Note: THW and DHW are calculated at the instant when the midpoint of the cut-in vehicle crosses the target lane marking.

.

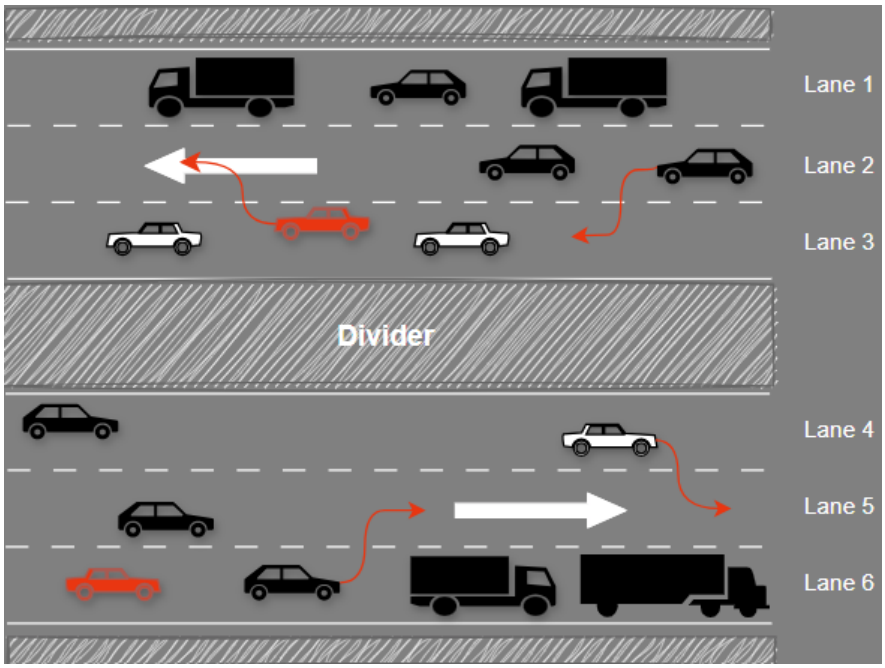


Figure 9.4: Diagrammatic representation of a 3-lane highway depicting a driving scene with LCs.

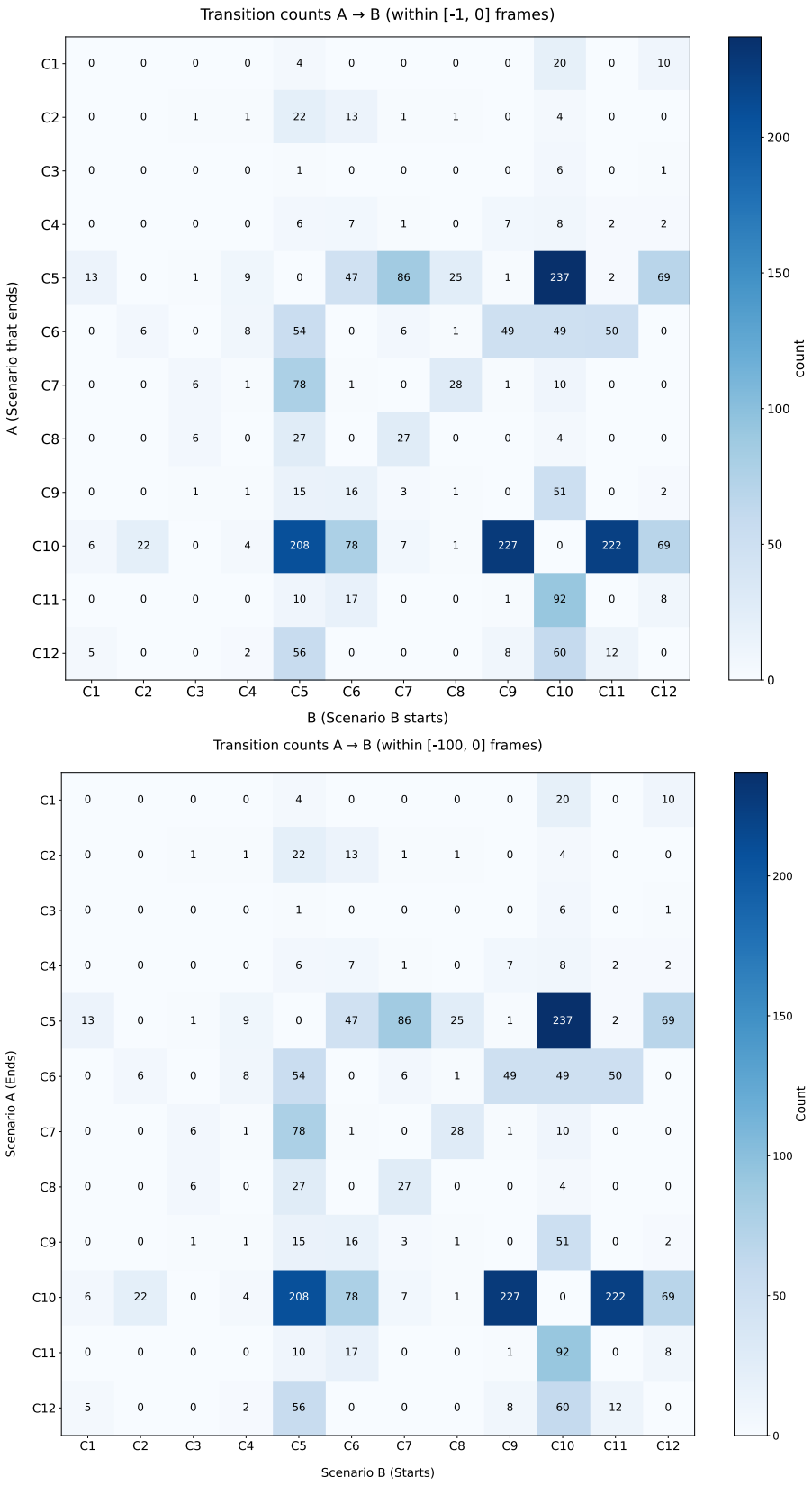


Figure 9.5: The above two heatmaps show that there is no significant scenario overlap (in the case of a smooth transition, refer to Table 9.6), as the count is almost the same.

.

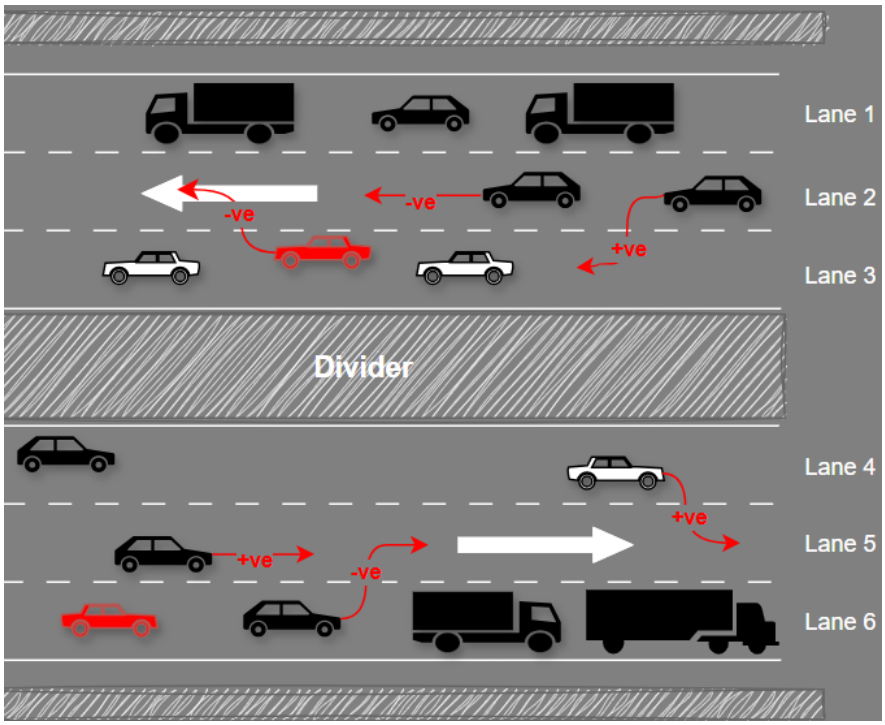


Figure 9.6: Longitudinal and lateral kinematics signs as per vehicle direction.

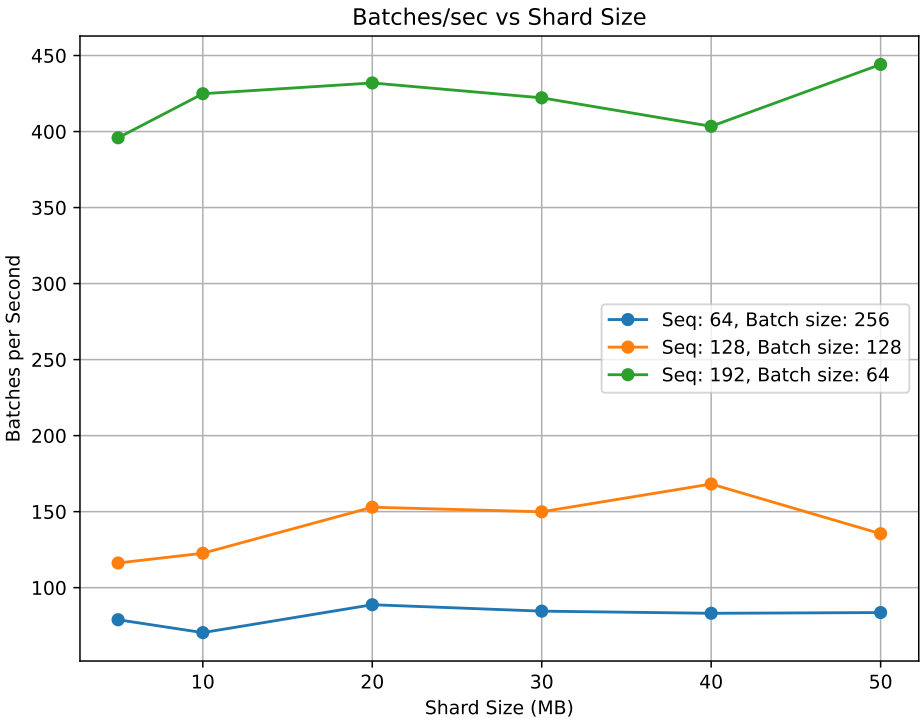


Figure 9.7: Batches processed per second against different shard sizes for the labelled sequence length and batch size configurations. For the algorithm, see 5.

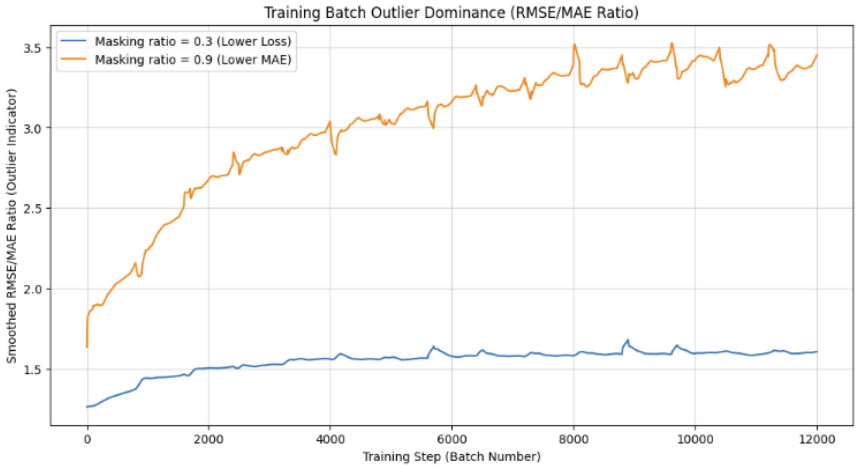


Figure 9.8: $RMSE/MAE$ ratio showing outlier dominance at masked positions with different masking ratios.

9.4 Model/data pipeline

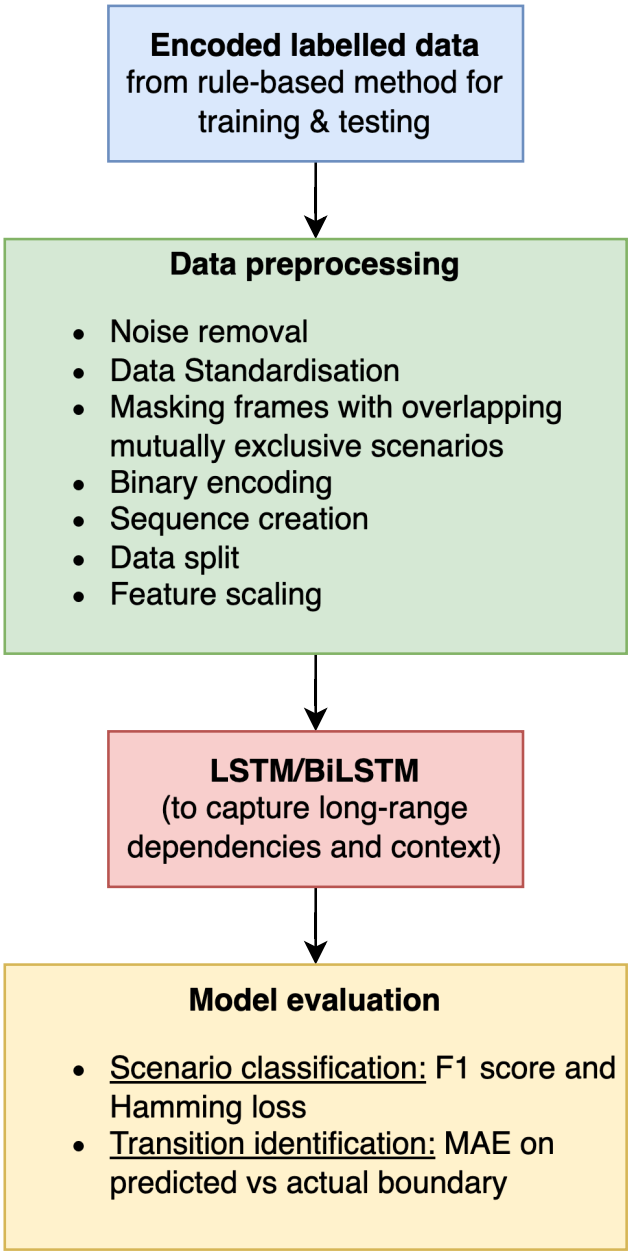


Figure 9.9: (Bi)LSTM pipeline

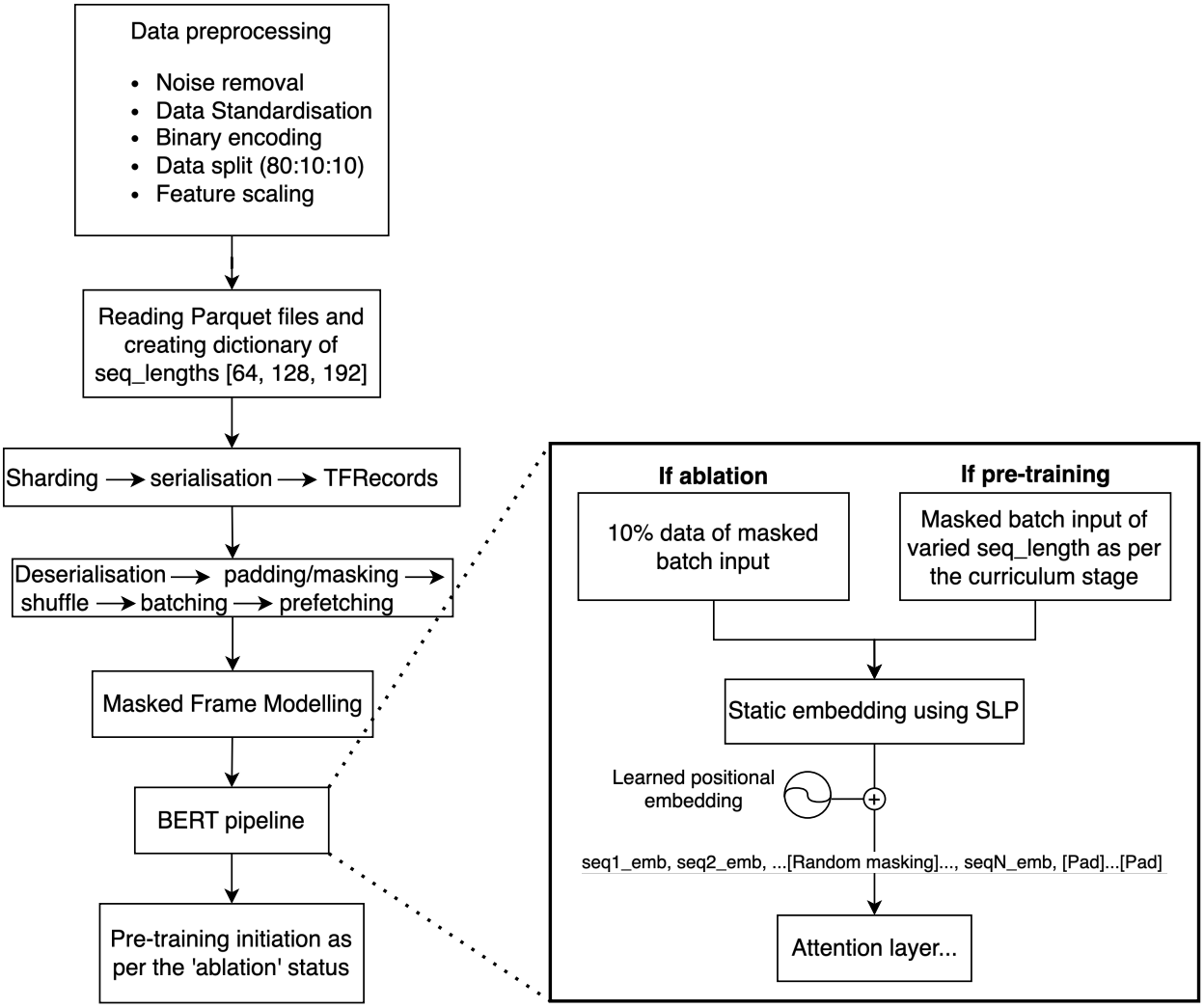


Figure 9.10: Overview of the BERT data pipeline

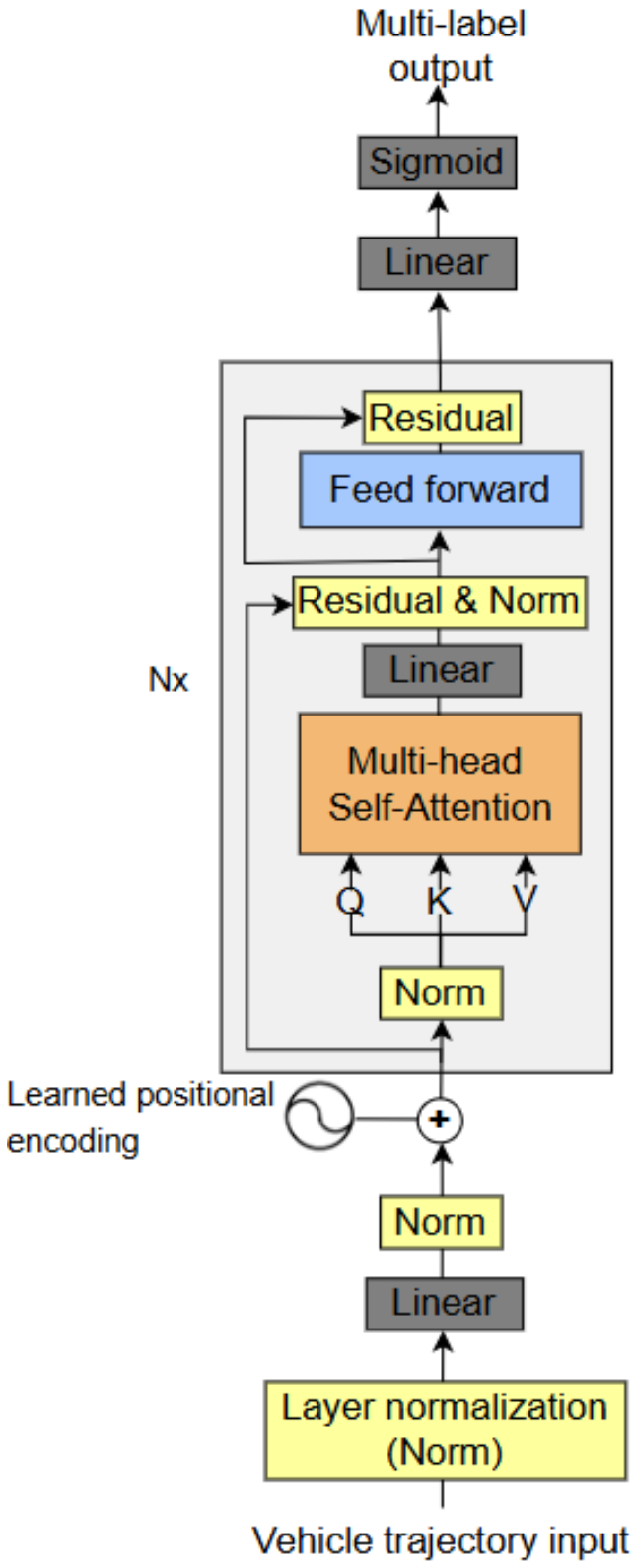


Figure 9.11: Overview of the BERT pipeline processing vehicle trajectory.

9.5 Experiments

This section contains data preprocessing steps, calculations, and ablation results that would have taken up too much space to include in the main thesis.

9.5.1 (Bi)LSTM: Data preprocessing

The majority of the time was spent planning and designing this module. Below are the steps followed in sequential order to get labelled data ready for model training and testing:

1. Noise removal: The rows where all the 12 targets are inactive (having value '0', meaning not present) were removed. This is because the last 100 m of the 420 m was not labelled for any vehicle. Otherwise, all other vehicles in front would suddenly disappear, resulting in inconclusive scenario boundaries. Such frames constituted $\approx 29\%$ of the sampled data.

2. Data standardisation: highD was collected for highways directing both ways ($left \rightarrow right$ & $right \rightarrow left$). Due to the image coordinate system, for opposite directions, the vehicle kinematic features, such as velocity and acceleration (both longitudinal and lateral), were assigned opposite signs. For example, a vehicle travelling from $left \rightarrow right$ will have a positive kinematics value compared to a vehicle travelling $right \rightarrow left$. Therefore, to maintain consistency across the data, vehicles travelling $right \rightarrow left$ were identified, and the sign of their kinematic features was reversed, along with assigning a new relevant lane ID, see Figure 9.6 and Table 9.8.

3. Masking frames with overlapping mutually exclusive scenario categories: Masking frames where mutually exclusive scenario categories, as listed in Table 9.4, are co-occurring. For masking, a new binary column was added, with '1' indicating co-occurrence. These frames were treated as noise and excluded from loss computation during backpropagation and model evaluation.

4. Binary encoding: Columns containing vehicle IDs were converted to binary, where '1' represents a vehicle's presence. This was done to reduce noise, as the vehicle ID itself does not provide any information about the scenario category, and keeping it might lead the model to learn illogical patterns and generalise poorly.

5. Sequence creation: The data is grouped by ego vehicle ID, and then a sequence of 25 frames (also called a sliding window) is created with a stride of 1 within each vehicle trajectory.

6. Data Split: $[train : val] \rightarrow [80 : 20]$ split was performed keeping shuffle enabled. This shuffled the sequences while preserving the within-sequence frame-level temporal consistency.

7. Feature scaling: After the split, the numerical features were scaled using $StandardScaler$. Scaling is essential for model performance and smooth gradient convergence, especially when dealing with large magnitudes.

- See Algorithm 2, defining the preprocessing pipeline and Figure 9.9 for overall (Bi)LSTM pipeline.

  Note: Initially, $[train : val : test] \rightarrow [75 : 15 : 15]$ split was used. However, during experimentation, it was found that the model was giving unusually high performance on the test set. However, when tested on a new, isolated test set (not split during training and validation set creation), the same model performed very poorly, highlighting data leakage in the earlier test set. The reason for the leakage was that, after shuffling the train and test sets, both sets shared sequences from a common set of vehicles. Though the sequences themselves were different, the fact that they came from the same vehicle caused the leakage. Apart from this, data splitting by vehicle and then shuffling was attempted to prevent data leakage. However, the model started overfitting after an epoch and performed poorly on the test set.
  Finally, it was decided to train the model as is, but to use an isolated test set not generated from the conventional train-test split.

9.5.2 BERT: Data preprocessing and pre-training setup

Steps 2, 4, 6, and 7 of Section-9.5.1 were followed as is. The standardised data was stored as Parquet files to save RAM and enable quick operations (e.g., reading and writing to the file), reducing storage requirements by 85% from 4.8 GB (CSV) to 0.697 GB (Parquet). Moreover, when creating TFRecords, it is preferred due to compatibility with TensorFlow. TFRecords holds a serialised object (binary) of the data, making it optimised for input-output tasks and memory [41]. See Algorithm 6, defining the pipeline: $CSV \rightarrow Parquet\ creation\ \&\ feature\ preprocessing \rightarrow TFRecords\ generation$.

It was decided to pre-train BERT on three different input sequence lengths rather than a single one. The objective is that each of these values will cover the spectrum of scenario category durations. A shorter sequence length will force the model to learn local patterns and scenario categories with shorter durations. On the other hand, a larger sequence length will push the model to capture more global patterns and scenario categories spanning across multiple frames. These three values were heuristically decided using Table 9.5. From Table 9.5, it could be observed that 5/12 scenario categories have a duration of $> 6s$, 5/12 have $3 - 5s$, and the remaining 2 have $2 - 3s$. Therefore, the resulting three sequence length values become [64 ($\approx 2s$), 128 ($\approx 5s$), 192 ($\approx 7s$)] [2]. All three are kept as multiples of 2 for maximum cache utilisation and to ensure the least is wasted during padding. Having different sequence lengths makes it important to have a custom batch size for each. The following are the main reasons:

- Quadratic memory complexity: Attention layer has memory complexity of $O(n^2)$ where n is the sequence length. This makes it challenging to maintain a consistent batch size that effectively utilises GPU resources across all sequence lengths without triggering OOM errors.
- GPU/TPU utilisation: Effective resource utilisation could be done by performing dynamic batching. Smaller sequence lengths can sustain larger batch sizes, whereas longer lengths must limit batch sizes. This makes sure that the resource capacity is utilised without crashing (OOM).

The following batch sizes were decided: [256, 128, 64], respectively. Both sequence lengths and batch sizes are kept even to align with the GPU architecture and achieve maximum parallelism, efficiency, and cache optimisation. Data read from Parquet files is split into shards and converted to TFRecord files. Shards (chunks of data) improve: data loading, data shuffling, prevent bottlenecks, and GPU utilisation during BERT pre-training.

From Figure 9.7, it is visually evident that 20MB is the optimal shard size across the three sequence lengths and batch size combinations. Additionally, it was observed that with larger sequence length and smaller batch size, the batches/sec increase. It may seem counterintuitive at first, but it all makes sense when it is realised that GPU parallelisation prefers fewer, larger operations over many smaller ones. With a 20MB shard size, the number of shards for the three pairs (sequence length, batch size) came out to be 152 (see 'num_shards' in Algorithm 5).

Before performing operations on the data to make it input-ready for BERT, TFRecords were deserialised and reconstructed as tensors. After which, tensors are passed through the following pipeline: $padding \rightarrow attention\ masking \rightarrow shuffling \rightarrow batching \rightarrow prefetch(tf.data.AUTOTUNE) \rightarrow MFM$. The discussion below entails the necessity of each part of this pipeline:

- Padding: It is done to increase shorter sequence lengths ([64, 128]) to make them as long as the maximum sequence (192) the model is going to see. This maintains a common input sequence length throughout the pre-training.
- Attention masking: In padded shorter sequences, the actual sequence length of importance is less. The rest ([pad]) should not be considered in the attention layer for context learning. Moreover, it has the added benefit of keeping the compute and memory load lower. Attention mask: [$1 \rightarrow vaid\ frame\ present,\ 0 \rightarrow [pad]$].
- Shuffling: It ensures the trajectory sequences are shuffled before batching. Shuffled set improves model generalisation.

[2] [Number of frames (corresponding duration in seconds)]

- Batching: The shuffled data is batched into three batches ([256, 128, 64]) corresponding to the three sequence lengths ([64, 128, 192]).
- Prefetch(tf.data.AUTOTUNE): It prepares the next batch of data while the model processes the current. This results in faster training by overlapping CPU (data loading, parsing, shuffling, etc.) and GPU (forward and backward passes) jobs. This way, the GPU hardly sits idle. AUTOTUNE here determines the number of batches to prefetch based on the system's CPU, GPU, and memory.
- Masked frame modelling (MFM): The $x\%$[3] batched data are then masked either at the frame-level or the feature-level. Masking is performed only on the valid tokens/frames and not $[pad]$.
- Curriculum learning: It was decided to pre-train BERT in a curriculum manner, starting with shorter sequence lengths and gradually showing larger ones. From Table 9.5, it was observed that the scenario categories have varying durations, and setting a common pre-training sequence length might not be optimal, potentially limiting the model's ability to fully understand the dynamics across all scenario categories. Moreover, for different sequence lengths, it is imperative that their batch sizes be set accordingly rather than a single global batch size. As sequence length increases, the batch size is reduced to maintain a similar computational cost. The curriculum learning is divided into the following three stages:
  - Stage 1: Start training on 100% of sequence length 64 with batch size 256.
  - Stage 2: Training continues with model weights from stage 1, on 70% of sequence length 64, having batch size 256, and the remaining 30% of sequence length 128, having batch size 128.
  - Stage 3: Training continues with model weights from stage 2, on 50% of sequence length 64 having batch size 256, 30% of sequence length 128 having batch size 128, and the remaining 20% of sequence length 192 having batch size 64.

  Note: In the later Section 7.2.2, BERT was pre-trained with and without curriculum stages, and it was found that pre-training with curriculum performed $\approx 3-6\%$ better on the downstream classification task.

- See Algorithm 7 defining the pre-training via masked frame modelling and Figure 9.10 for the entire data pipeline.

9.5.3 Test set preparation

An isolated test set was used that underwent the data preprocessing steps mentioned in Appendix 9.5.1, except for data splitting and shuffling. In addition, unlike the training set, which involved overlapping sequence creation, the test set sequences were non-overlapping to avoid duplicate frames. Previously fitted scalers were applied for the feature scaling. The trajectory is generally fed as a continuous temporal set for scenario category classification. This makes it essential that the test set maintains sequence-level temporal continuity for unbiased evaluation.

9.5.4 Computational and memory estimation for the pre-training

The calculations below estimate compute and memory requirements for a full pre-training setup (see Table 6.4).

[3] x here is a tunable parameter; baseline value is kept 0.15, i.e., 15% during ablation study.

Table 9.10: Trainable parameter breakdown for BERT's pre-training configuration (*model dimension* = 256, *encoder layers* = 8, *Input features* = 20)

Component	Layer Type	Formula	Parameters
A. Input Stack			
Input Projection (Dense)	Dense	$(20 \cdot 256) + 256$	5,376
Positional Embedding	Embedding	$192 \cdot 256$	49,152
Input Layer Norms	LN	$2 \cdot 20$	40
		Subtotal (A)	**54,568**
B. Single Transformer Encoder Layer (Repeated 8 times (nlayers))			
Multi-Head Attention (MHA)	Dense (Q, K, V, Out)	$4 \cdot 256 \cdot 256$	262,144
Feed-Forward Network (FFN)	Dense (Inner, Outer)	$(256 \cdot 256) + 256$	65,792
Gated FFN (SwiGLU)	Dense (Inner, Outer)	$2 \cdot [(256 \cdot 682) + 682] + (256 \cdot 682) + 256$	525,396
Layer Normalization (x2)	LN	$2 \cdot (2 \cdot 256)$	1,024
		Subtotal Per Layer	854,356
		Total Transformer Stack (8 layers)	**6,834,848**
C. Output Head			
Output Projection	Dense	$(256 \cdot 20) + 20$	5,140
		Subtotal (C)	**5,140**
		TOTAL TRAINABLE PARAMETERS	**6,894,556**

Total memory footprint (model size + temporary memory):

- Model size:

$$M_{\text{Weights}} = P_{\text{Total}} \times Size_{\text{FP32}}$$
$$M_{\text{Weights}} = 6,894,556 \times 4 = 27,578,224 \; bytes \approx 27.5MB$$

- Temporary memory:[4]

$$M_{\text{Optimizer}} = 2 \cdot M_{\text{Weights}} \approx 2 \times 27.5 \approx 55MB$$
$$M_{\text{Gradient}} \approx M_{\text{Weights}} \approx 27.5MB$$
$$M_{\text{Activation}_{\text{base}}} \approx batch\ size \times sequence\ length \times model\ dimension \times size_{\text{FP16}}$$
$$M_{\text{Activation}_{\text{base}}} \approx 256 \times 192 \times 256 \times 2 \approx 2,51,65,824 \; bytes \approx 25.1MB$$
$$M_{\text{Activation}} \approx M_{\text{Activation}_{\text{base}}} \times |Activation|$$
$$M_{\text{Activation}} \approx 25.1 \times [1 + 5 \times |layers|] \approx 25.1 \times [1 + 5 \times 8] \approx 1029MB$$

- Total:

$$M_{\text{Total}} = M_{\text{Weights}} + M_{\text{Optimizer}} + M_{\text{Gradient}} + M_{\text{Activation}}$$
$$M_{\text{Total}} = 1,139MB \approx 1.13GB \approx 1GB$$

FLOPS and training time calculation:
The following formula is derived from reference [30, 35, 42, 43]. Calculates floating-point operations in a single training step containing a forward and a backward pass.

$$\text{FLOPs/step} = 3 \times 2 \times L \times B \times [2 \times S^2 \times D + 13 \times S \times D^2]$$

Where:

- L is the number of encoder layers,

[4]Temporary memory generated during forward pass remains in cache for the backpropagation and is cleared once training completes. To be conservative, attention memory is computed using the maximum batch size and sequence length.

- B is the batch size,
- S is the sequence length,
- D is the model's dimension,
- $\times 3$ to get FLOPs for both forward and backward passes. A backward pass is considered to have $\times 2$ FLOPs of the forward pass,
- $\times 2$ MACs to FLOPs conversion.

Find below the FLOPs/step computed for each (S, B) configuration- $[(64, 256), (128, 128), (192, 64)]$.

$$\text{FLOPs/step}_{(64,\ 256)} = 6 \times 8 \times 256 \times [2 \times 64^2 \times 256 + 13 \times 64 \times 256^2] \approx 0.69\ TFLOPS$$
$$\text{FLOPs/step}_{(128,\ 128)} = 6 \times 8 \times 128 \times [2 \times 128^2 \times 256 + 13 \times 128 \times 256^2] \approx 0.72\ TFLOPS$$
$$\text{FLOPs/step}_{(192,\ 64)} = 6 \times 8 \times 64 \times [2 \times 192^2 \times 256 + 13 \times 192 \times 256^2] \approx 0.56\ TFLOPS$$

Theoretically, NVIDIA P100 is documented to have $\approx$ 9.5 TFLOPS; however, in practice, it varied between 1.4 and 1.9 TFLOPS. For the time-per-step calculation, the middle value of 1.7 TFLOPS was used.

$$\text{Time/step}_{(64,\ 256)} \approx 0.69e12/1.7e12 \approx 0.405\ s$$
$$\text{Time/step}_{(128,\ 128)} \approx 0.72e12/1.7e12 \approx 0.423\ s$$
$$\text{Time/step}_{(192,\ 64)} \approx 0.56e12/1.7e12 \approx 0.329\ s$$

To estimate the total training time, the number of training steps per pre-training curriculum stage is needed.

$$\text{Steps}_{(\text{Stage 1})} = 2760,\ \text{where } 100\% \rightarrow (64,\ 256)$$
$$\text{Steps}_{(\text{Stage 2})} = 2760,\ \text{where } 70\% \rightarrow (64,\ 256)\ \textit{and}\ 30\% \rightarrow (128,\ 128)$$
$$\text{Steps}_{(\text{Stage 3})} = 2944,\ \text{where } 50\% \rightarrow (64,\ 256),\ 30\% \rightarrow (128,\ 128)\ \textit{and}\ 20\% \rightarrow (192,\ 64)$$

$$\text{Epoch time}_{(\text{Stage 1})} = (2760 \times 0.405)/60 \approx 18.63\ mins$$
$$\text{Epoch time}_{(\text{Stage 2})} = [(2760 \times 0.7 \times 0.405) + (2760 \times 0.3 \times 0.423)]/60 \approx 18.87\ mins$$
$$\text{Epoch time}_{(\text{Stage 3})} = [(2760 \times 0.5 \times 0.405) + (2760 \times 0.3 \times 0.423) + (2944 \times 0.2 \times 0.329)]/60 \approx 18.38\ mins$$

Planned on running each stage for 12 epochs (101,568 training steps). The following would be the estimation of the total pre-training time:

$$\text{Total pre-training time}_{((101{,}568\ \text{steps})} = [(12 \times 18.63) + (12 \times 18.87) + (12 \times 18.38)]/60 \approx 11.2\ \text{hrs}$$

9.5.5 Ablation study tables

The results given here consist of all conditions, of which three models were given in the main thesis. As in the thesis, ablation studies were performed to identify optimal hyperparameter configurations and assess their impact.

Table 9.11: Smooth scenario category transitions against their NMABE_{avg} when using BiLSTM ([*sequences* : 484, 507; *layers* : 4]). 'Support' signifies the total number of instances summed up across all vehicle trajectories. 'Missed' signifies the total number of instances where the model missed that particular transition for that vehicle. '$scenario_i \rightarrow scenario_j$' signifies the pair which was difficult to identify and missed by almost all LSTM/BiLSTM configurations.

Smooth transitions present in the test set (*7 out of possible 16 pairs.*)	**NMABE_{avg}**	**Support**	**Missed**
Cut-in in front of ego vehicle → Lead vehicle cruising	0.72	9	7
Cut-out in front of ego vehicle → Ego vehicle driving in lane without lead vehicle	–	1	1
Cut-out in front of ego vehicle → Lead vehicle cruising	–	21	21
Ego performing lane change → Ego vehicle driving in lane without lead vehicle	0.15	2	0
Ego vehicle driving in lane without lead vehicle → Lead vehicle accelerating	–	1	1
Ego vehicle driving in lane without lead vehicle → Lead vehicle cruising	0.13	30	19
Ego vehicle driving in lane without lead vehicle → Lead vehicle decelerating	–	1	1

Table 9.12: Evaluating fine-tuned $BERT_{50{,}784\ steps}$ (signifies BERT pre-trained for 50,784 steps) with different configurations while doing iterative hyperparameter optimisation. Kept base configuration of $seq\_len = 192$ *and* $batch\_size = 32$. The average, is calculated using the formula: $\frac{\text{F1+AUPRC}}{2}$, giving equal weightage to both F1 and AUPRC.

Model configuration	**AUPRC ↑**	**F1 Score ↑**	**Average ↑**
Model (30% masking)	0.4646 ± 0.0091	0.1462 ± 0.0064	0.3054
Model (90% masking)	0.5121 ± 0.0569	0.1461 ± 0.0084	0.3291
Model (90% masking+dropout (0.1))	0.7914 ± 0.0121	0.3519 ± 0.0405	0.5716
Model (90% masking+dropout (0.1)+SwiGLU)	0.7896 ± 0.0116	0.3775 ± 0.0196	0.5835

Metrics (↑) indicate that higher values are better. $BERT_{50{,}784}$ used for quicker initial comparison. The best overall performance, as determined by the **average score**, was achieved by 90% *masking*, 0.1 *dropout ratio*, *and SwiGLU as activation function* (0.5835). The model with each configuration is run three times with different seed values to report *mean* ± *standard deviation*. The coefficient of variation across all configurations and two evaluation metrics is less than 0.05, suggesting model reliability.

Table 9.13: Ablation Study: Impact of sequence length and batch size on fine-tuning performance for multi-label classification. Study done on $BERT_{101,568\ steps}$ while keeping LoRA $rank = 4$ *and* $alpha = 4$. The composite score is calculated using the normalised formula: $\frac{\text{F1+AUPRC+(1−Hamming loss)}}{3}$.

Seq. length	Batch size	AUPRC ↑	F1 Score ↑	Hamming loss ↓	Composite Score ↑
64	4	0.8315 ± 0.0068	0.3969 ± 0.0088	0.0711 ± 0.0006	0.7191
	8	0.8096 ± 0.0090	0.3920 ± 0.0040	0.0746 ± 0.0010	0.7090
	16	0.7725 ± 0.0115	0.3504 ± 0.0252	0.0786 ± 0.0026	0.6814
	32	0.7320 ± 0.0027	0.3299 ± 0.0282	0.0826 ± 0.0026	0.6598
128	4	0.8492 ± 0.0080	0.4200 ± 0.0101	0.0717 ± 0.0018	0.7325
	8	0.8309 ± 0.0069	0.4025 ± 0.0130	0.0724 ± 0.0020	0.7203
	16	0.8131 ± 0.0057	0.3932 ± 0.0141	0.0758 ± 0.0039	0.7102
	32	0.7731 ± 0.0156	0.3666 ± 0.0066	0.0788 ± 0.0029	0.6869
192	**4**	0.8564 ± 0.0053	0.4356 ± 0.0091	0.0678 ± 0.0017	**0.7414**
	8	0.8501 ± 0.0029	0.4278 ± 0.0089	0.0700 ± 0.0007	0.7359
	16	0.8156 ± 0.0115	0.3920 ± 0.0182	0.0710 ± 0.0008	0.7122
	32	0.7868 ± 0.0030	0.3805 ± 0.0046	0.0759 ± 0.0030	0.6971

Metrics (↑) indicate that higher values are better and vice versa for (↓). The best overall performance, as determined by the **composite score**, was achieved by the configuration: $seq\_len = 192$ *and* $batch\_size = 4$ (0.7414), effectively balancing high values of AUPRC and F1 against the low value of hamming loss. Apart from this, performance with $batch\_size = 1$ (not included in the table) was also tested and found to perform poorly. The model with each configuration is run three times with different seed values to report $mean \pm standard\ deviation$. The coefficient of variation across all configurations and two evaluation metrics is less than 0.05, suggesting model reliability.

Table 9.14: Ablation Study: Impact of LoRA rank and alpha on the best base configuration ($seq\_len = 192$ *and* $batch\_size = 4$).

LoRA Rank	LoRA Alpha	AUPRC ↑	F1 Score ↑	Hamming loss ↓	Composite Score ↑
1	1	0.8512 ± 0.0043	0.4224 ± 0.0048	0.0713 ± 0.0021	0.7341
1	4	0.8427 ± 0.0127	0.4165 ± 0.0111	0.0713 ± 0.0034	0.7293
4	**4**	0.8564 ± 0.0053	0.4356 ± 0.0091	0.0678 ± 0.0017	**0.7414**
4	8	0.8505 ± 0.0068	0.4249 ± 0.0127	0.0709 ± 0.0011	0.7348
8	8	0.8459 ± 0.0075	0.4387 ± 0.0086	0.0724 ± 0.0003	0.7374
8	16	0.8440 ± 0.0031	0.4207 ± 0.0043	0.0703 ± 0.0015	0.7315
16	16	0.8520 ± 0.0053	0.4269 ± 0.0073	0.0689 ± 0.0014	0.7367
16	32	0.8420 ± 0.0076	0.4381 ± 0.0092	0.0687 ± 0.0018	0.7371
32	32	0.8469 ± 0.0009	0.4160 ± 0.0027	0.0713 ± 0.0027	0.7305
32	64	0.8389 ± 0.0073	0.4223 ± 0.0227	0.0700 ± 0.0011	0.7304
64	64	0.8398 ± 0.0068	0.4155 ± 0.0065	0.0682 ± 0.0004	0.7290
64	128	0.8349 ± 0.0106	0.4124 ± 0.0167	0.0696 ± 0.0002	0.7259

The maximum composite score was achieved by the $rank = 4$ *and* $alpha = 4$ configuration (0.7414), indicating the best trade-off between maximising AUPRC and F1 while minimising hamming loss. The model with each configuration is run three times with different seed values to report $mean \pm standard\ deviation$. The coefficient of variation across all configurations and two evaluation metrics is less than 0.05, suggesting model reliability.