Understanding Driving Behaviours: Classification and Regression Approaches

by

Qixuan (Josh) Sun

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2023

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The emergence of self-driving technology is poised to revolutionize transportation by empowering vehicles to operate autonomously. These advancements are classified into distinct levels of automation, delineating the progression from rudimentary driver assistance to complete autonomy. However, these advancements are accompanied by a spectrum of challenges. For example, the intricate interplay between autonomous and human-driven vehicles necessitates intuitive human interaction capabilities. This thesis focuses on driver behaviour learning, which is a way towards a safe and personal self-driving system. The eventual goal of driver behaviour learning is to equip self-driving vehicles with a deep understanding of human driving patterns. Leveraging learned human behaviours, self-driving vehicles can mirror familiar and intuitive driving actions, thereby fostering harmonious and predictable engagements on the road.

This thesis focuses on investigating using different machine learning methods to learn an accurate and robust predictor of driver behaviours from multiple drivers and also personalize outputs for each individual driver. We define three driving behaviours.

1. Driver Identification: Given an observed driving execution history, identify the driver in the vehicle.

2. Lane Positioning: Given an observed driving execution history, predict the subject vehicle in the lane in the foreseeable future.

3. Lane Keeping: Given an observed driving execution history, predict the speed and steering angle of the subject vehicle in the foreseeable future.

In the experiment, we mainly test with LSTM and Transformer, which are commonly used in time-series data. Specifically, we conduct experiments with LSTM and Transformer encoders to examine their capacity to model driving data in three behaviours. Furthermore, we employ LSTM and Transformer decoders to forecast lane positions, speeds, and angles. Variations in hyperparameter settings, such as width and depth, are investigated for all three behaviors in search of potential optimal configurations.

Acknowledgements

I extend heartfelt gratitude to my supervisor, Dr. Mark Crowley, for his invaluable guidance and unwavering support throughout this research journey. His expertise, constructive feedback, and encouragement have been instrumental in shaping the direction of this thesis. Working under Dr. Mark Crowley has been an honor, and I am privileged to have benefited from his wisdom and dedication.

I extend my sincere appreciation to the dedicated members of our research lab. Their collaborative spirit, insightful discussions, and willingness to share their expertise have significantly enriched this study. Special thanks to Laura McCrackin, Zehra Camlica, Shayan Shirahmad Gale Bagi, and Takin Tadayon for their valuable insights and support on the driving behaviour project. The vibrant intellectual environment fostered by my lab mates has been a constant source of inspiration. I am grateful for the camaraderie, knowledge sharing, and encouragement that have collectively contributed to the success of this research endeavor.

I also want to thank my friends, Jiahao Wu (University of Columbia), Xiaoying Xing (University of Chicago), and Yuan You (University of Alberta), whose unwavering encouragement and steadfast support during the challenging phases of thesis writing have been a guiding light. Your uplifting words, empathy, and unwavering presence provided the strength needed to overcome obstacles.

Dedication

I dedicate this thesis to my loving family, whose unwavering support and encouragement have been my source of strength throughout this journey. Your sacrifices and belief in me have made all the difference.

To my best friend, Dake Zhang, your friendship has been a constant source of inspiration and joy, grounding me during the challenging times. Thank you for always being there.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The emergence of self-driving technology is poised to revolutionize transportation by empowering vehicles to operate autonomously. These advancements are classified into distinct levels of automation, delineating the progression from rudimentary driver assistance to complete autonomy. The Society of Automotive Engineers (SAE) determines the intelligence level and automotive capabilities of vehicles on six levels from 0 to 5 [15]. At Level 0, human control remains paramount, while Level 1 introduces limited vehicular assistance, either in steering or acceleration. Level 2 marks partial automation, where vehicles manage both steering and acceleration under specific conditions. Progressing to Level 3, conditional automation allows vehicles to undertake most driving tasks within defined scenarios, permitting drivers to disengage temporarily. Elevated to Level 4, high automation empowers vehicles to navigate without human intervention within certain environments. Finally, Level 5 signifies full automation, where vehicles operate independently across all conditions.

However, these advancements are accompanied by a spectrum of challenges. Safety constitutes a paramount concern, demanding impeccable real-time decision-making abilities to navigate intricate traffic dynamics. Sensor technology assumes significance, as robust perception mechanisms are imperative, capable of operating seamlessly in diverse and challenging conditions. Effective decision-making algorithms must be developed, capable of accounting for ethical dilemmas, adhering to traffic laws, and ensuring swift, rational responses. Regulatory frameworks and infrastructure must be adapted to accommodate autonomous vehicles, and data collection must be balanced with privacy concerns. The intricate interplay between autonomous and human-driven vehicles necessitates intuitive human interaction capabilities. Mapping and precise localization systems are essential for

accurate spatial perception. Rigorous testing and validation protocols must ensure system reliability and safety.

This thesis focuses on driver behaviour learning, which is a way towards a safe and personal self-driving system. The eventual goal of driver behaviour learning is to equip self-driving vehicles with a deep understanding of human driving patterns. This newfound insight would facilitate the anticipation and seamless response to human driver actions, thereby reducing the potential for accidents and enhancing overall road safety. Moreover, the predictive prowess endowed by this fusion would allow self-driving systems to foretell the actions of other road users – a pivotal component for informed decision-making in complex traffic scenarios.

The significance of natural interactions between autonomous vehicles and their human-driven counterparts cannot be overstated. Leveraging learned human behaviours, self-driving vehicles can mirror familiar and intuitive driving actions, thereby fostering harmonious and predictable engagements on the road. This in turn contributes to the cultivation of public trust and acceptance, a key factor in accelerating the widespread adoption of self-driving technology.

1.1 Motivation and Goals

This thesis focuses on investigating using different machine learning methods to learn an accurate and robust predictor of driver behaviors from multiple drivers and also personalize outputs for each individual driver. The specific challenge is given a snippet of driving history, can we effectively predict the future status of this vehicle, or how each particular driver will behave in the next following time steps?

For example, with collected data from the drivers, one predictive or behavioral question would be "do drivers still perform lane-changing behavior even if there is a vehicle in their nearby lane? Or will different drivers stay in the same position of the lane when some vehicles are approaching them in the nearby lane?" There are plenty of such questions that need to be answered and we want to train an end-to-end model that can make such predictions for all of the situations. When such a model is trained offline, it can be equipped and used in a vehicle to boost the Advanced Driving Assistance System (ADAS) system or provide additional information for a fully self-driving system. A special behavior that is important for driver behaviors is identifying a particular driver from a certain snippet of driving history, called driver identification. In this behavior, we predict the particular driver using the snippet rather than the future status of the vehicle. In other words, we're

answering the question, "Who is the driver if we're given a snippet of driving history". Briefly, we iteratively predict a bunch of future statuses or just one state.

Another essential goal is to improve how well the model predictions could mimic actual human-like behaviors and each individual driver. When the data amount increases, the model that is learned from the data can be more "average" and thus lose the characteristics of every single driver's behaviors. We want to build a personalized model that can identify and specify individual drivers' styles. From a high level, there are two ways of doing this. One is through a two-stage paradigm, pre-training, and fine-tuning. A base model is pre-trained by many common driving behaviors, or at least "average" behaviors, from large initial training sets of real driver data. When a new owner of a vehicle appears with new data specific to their style, the model can "fine-tune" its prediction to better fit the particular driver. Since we don't have much data to train and we need to divide some for validation and testing, we are not able to use this pre-train and fine-tune paradigm. Thus, we have the second way, which is to learn an encoding for every single driver that contains their driving style information and then train a prediction model based on historical data and the driving style vector.

To achieve these goals, we define the following driving behaviours.

1.2 The Three Driving Behaviours

This thesis focuses on three particular driving behaviours of human drivers:

1. Driver Identification: Given an observed driving execution history, identify the driver in the vehicle.
2. Lane Positioning: Given an observed driving execution history, predict the subject vehicle in the lane in the foreseeable future.
3. Lane Keeping: Given an observed driving execution history, predict the speed and steering angle of the subject vehicle in the foreseeable future.

1.3 Assumptions

Before we get into our research questions, some assumptions were made for the collected data. Drivers were driving under normal driving conditions. Drivers were not driving under

the influence of alcohol. Drivers didn't lie about their information or driving experience. Drivers are representative samples of the population of interest. And for the features that are used for this thesis, these features are sufficient to capture the targetted behaviours.

1.4 Research Questions

We analyze the performance of Transformers and LSTMs concerning these three behaviours. Through multiple experiments encompassing varied configurations, we aim to address the subsequent research questions:

Do Transformers outperform LSTMs in modeling time-series driving data?

Do Transformers outperform LSTMs in predicting future time steps using time-series driving data?

What is the impact of the width and depth of a model for time-series driving data?

How much data do we need to identify a particular driver with high accuracy?

1.5 Thesis Outline

This thesis is organized as follows. We briefly go through the background of driver behaviour learning and deep learning structures, e.g., Transformers, in Chapter 2. Following this, Chapter 3 presents a detailed depiction of the dataset. In the subsequent chapters, namely Chapter 4, Chapter 5, and Chapter 6, we show the experiments and results with the three behaviors: driver identification, lane positioning, and lane keeping.

In summary, we conduct experiments with LSTM and Transformer encoders to examine their capacity in modeling driving data in three behaviours (Chapter 4, Chapter 5, and Chapter 6). Furthermore, we employ LSTM and Transformer decoders to forecast lane positions (Chapter 5) and speed and angles (Chapter 6). Variations in hyperparameter settings, such as width and depth, are investigated for all three behaviors in search of potential optimal configurations. Additionally, we delve into the single-driver problem, defined in Chapter 3, within the context of driver identification (Chapter 4).

Chapter 2

Background

2.1 Autonomous Driving Background

Autonomous driving has attracted attention quite a lot in the past three decades due to its various benefits. First of all, it can help reduce the number of crashes on the road and build greater safety. 94 percent of crashes have been identified as a factor of driver behavior or error [19], and autonomous technology can help reduce driver errors. Higher levels of autonomy have the potential to reduce risky and dangerous driver behaviors. Secondly, high-level automation offers more personal freedom. People with disabilities are capable of being self-sufficient, and self-driving can help them make their dream come true. Thirdly, highly automated vehicles can help save money. They can reduce the cost of medical payments, vehicle repairs, and lost work time caused by accidents, and consequently, reduce the car insurance fee. Fourthly, with autonomous driving, people can recapture time to pursue time for more productive and entertaining activities, like responding to emails or browsing websites. Lastly, severe traffic congestion can be alleviated using self-driving vehicles as most of the traffic issues are caused by car crashes. Self-driving cars maintain a consistent and safe distance between vehicles, which helps to reduce the number of stop-and-go waves that generate road congestion.

Autonomous vehicles have reached various levels of automation, ranging from Level 0 (no automation) to Level 5 (full automation). Many vehicles on the road today offer advanced driver assistance systems (ADAS) that provide features like adaptive cruise control, lane-keeping assistance, and automated parking. However, true Level 5 autonomy, where vehicles can operate without human intervention in all conditions, is not yet widespread.

The field of autonomous driving has made significant advancements but still faces several challenges. Challenges include navigating complex urban environments, handling adverse weather conditions, and ensuring safety in mixed traffic scenarios. Developing robust perception systems that can accurately identify and react to a wide range of objects and situations remains a challenge.

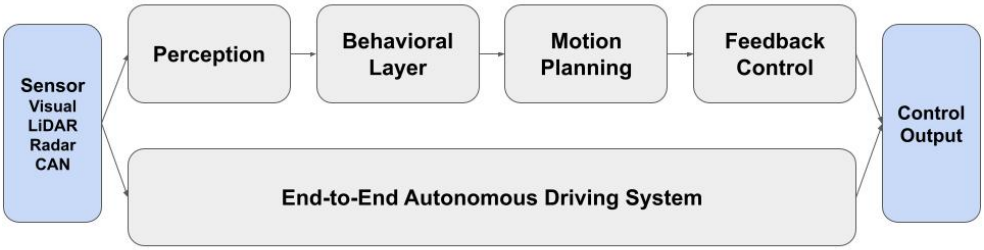


Figure 2.1: Two types of methods for autonomous driving systems.

Considering the numerous impacts that self-driving has, it's not surprising that self-driving technologies have attracted a lot of researchers and engineers. The two most commonly used methods that some recent surveys have claimed for autonomous driving planning systems are [16, 18]: **pipeline** and **end-to-end** methods. A typical pipeline method basically consists of four components (Figure 2.1). In the very beginning, a stream of observations from sensors is collected and passed through a perception module, processing and extracting useful information for further decision, which is followed by a behavior layer, deciding on a local driving task, e.g., towards a destination and abiding by rules of the road. A motion planning sub-system then selects a continuous path through the surroundings to achieve a local navigation task. Lastly, A control system corrects errors in the execution in reality which maintains the safety of the whole driving system. Most recently, some researchers switched to end-to-end methods that combined the above four modules together and showed promising results. This kind of method aims to predict the final motion directly through raw sensor data all in one big, impenetrable system. However, with the ever-increasing popularity of deep learning techniques, such as Transformers,

issues with verification and validation of autonomous end-to-end driving systems still need to be addressed and gain increasing urgency.

For either approach, from a machine learning view, the inputs to these decision-making systems can be seen as environmental clues (e.g. perception outputs, raw sensor images) and labels describing behaviors [12]. For pipeline systems, the output of the system could be either high-level behaviors low-level control commands, or even both. For end-to-end systems, there is essentially a single decision-making system while pipelines are composed of multiple sequential systems. This work focuses on driver behavior learning, which would be part of pipeline systems, aiming to predict high-level behaviors, e.g., lane-changing and lane-positioning, and low-level control commands, e.g., steering and angle.

Driver behavior learning is inevitably playing a crucial role in advancing self-driving technology. Firstly, understanding and predicting human driver behavior is vital for creating safe and predictable interactions between autonomous vehicles and human-driven vehicles. Learning driver behavior can help autonomous systems anticipate and respond to human actions, reducing the risk of accidents. Besides, autonomous vehicles need to predict the behavior of other road users, including human drivers, pedestrians, and cyclists. By learning from historical data, self-driving systems can make more accurate predictions about the likely actions of other road users, allowing them to plan and execute safer maneuvers. Safe and natural interactions are another advantage of driver behaviour learning. Autonomous vehicles need to communicate their intentions to human drivers and pedestrians. By learning from observed driver behaviors, self-driving systems can mimic natural and intuitive driving behaviors, enhancing the predictability and acceptability of autonomous vehicles on the road.

To predict high-level driving behaviours, one difficult task is lane-changing behaviors as the driver's intentions are hard to understand, and changing behavior might be caused by multiple reasons. [20] proposed a data-driven model to capture the lane change intention of human drivers. They collected data from typical lane-changing situations which were well-designed and trained Support Vector Machine (SVM) classifiers to perform personalized lane-changing behaviors. They further integrated the decision logic into a control part to build a self-driving system. [13] mentioned the complexity and randomness of autonomous vehicle lane change decisions, and thus defined three useful features based on analyzing factors of three different aspects: benefit, safety, and tolerance. They proposed an SVM model with Bayesian parameters optimization to solve this issue which verified the effectiveness compared to previous rule-based and other SVM decision models. There are some other works focusing on solving the lane-changing problem with SVMs. Some researchers [27] focused on specific truck scenarios. They presented an SVM model based on the space headways and relative longitudinal speed between the going truck and the

surrounding vehicles which can be in the same lane or in the target lane. They further analyzed curved roads which helped with the capability of generalization of their model. Beyond SVM models, there are other models used for lane-changing behavior prediction. [9] empirically compared two models: MOBIL, a well-established rule-based model, and a naive Bayes algorithm. Results showed that overtaking behaviors are quite different, which requires analyzing them separately. More recently, deep learning-based methods have been applied to this task [22, 25, 6, 23], which will be described in the next section.

However, such lane-changing conditions are generally selected and manipulated [20, 22, 25], which made it hard to collect and implement. There are some other papers focused on other discrete behavior planning. Xu et al.[23] proposed an FCN-LSTM architecture for discrete behavior prediction, straight, stop, left turn, and right turn, based on front-sided camera video data. So far, we have provided a brief overview of high-level driving behaviours and then we'll explain what researchers conducted with low-level behaviours, mostly using end-to-end methods.

End-to-end methods are popular for both machine learning and self-driving researchers in recent years, which are usually used to predict low-level commands or trajectories directly. Convolution neural networks(CNN) [3, 5] were utilized for steering angle prediction using raw images from cameras as input. Parkash et al.[17] proposed a Vision Transformer-based approach for multi-modal fusion and predicted the trajectory of the next four time slots directly.

However, end-to-end approaches are black boxes and thus make it hard to implement in reality due to safety reasons. To solve these problems, a series of visualization methods have been proposed to partly explain neural networks. This work [3] tried to explain a neural-network-based end-to-end system, known as PilotNet proposed by NVIDIA, and their method could learn obvious features such as road edges, lane markings, and other cars, which shows that PilotNet indeed leans to recognize objects on the road. Kim et al.[10] proposed a two-staged approach for an interpretable network for end-to-end self-driving cars. First, a visual attention model is trained to predict steering angle, in which the attention model highlights image regions that affect outputs more. Then, a causal filter step is applied to determine which region has the true influence. Later on, Mori et al.[14] designed an attention branch to visualize their convolution result. Through such visualizations, the region of interest is highlighted when operating steering and throttle commands. Zeng et al.[28] visualized predictive trajectory with probability on the HD map.

Another way of learning driver behaviors is through driver identification. A mature driver assistance system that responds to a short snippet of sensor data gives engineers the

potential to build applications that adapt to specific users. At the personal vehicle level, the system can quickly identify the driver behind the wheel and then adjust vehicle settings accordingly. This adjustment will not be limited to air conditioning or seat position, but will also help with vehicle maneuvering according to drivers' habits. Considerable efforts have been made with the aim of characterizing and identifying the drivers based on driving behavior analysis.

2.2 Machine Learning Background

Deep learning-based methods [22, 25, 6, 23] have attracted much attention for this task. A Deep Neural Network (DNN)[25] was proposed to make human-like overtaking maneuver decisions based on traffic-related and intention-related factors which were extracted from traffic scene interpretation processes. Their results showed that extracted decision factors contributed a lot to the performance of the model. Long-short-term memory(LSTM) network [7], designed for time series prediction tasks and can reduce gradient descent from typical Recurrent Neural Networks(RNN), are used for such driver behavior learning tasks. Wang et al.[22] designed a network that combined LSTM and Conditional Random Field (CRF) for on-road self-driving. They treated this task as a sequence labeling problem when its input is a bunch of time series and outputs a one-hot vector indicating the suitable maneuver. To mimic human-like lane-changing decision-making, Xu et al.[23] proposed a novel FCN-LSTM architecture that incorporated previous moving paths and video history together to learn a future ego-motion prediction model. And they used semantic segmentation as a side task to further improve the model.

Beyond LSTMs, Transformers which were developed for Natural Language Processing tasks have recently been applied for time series prediction tasks. Transformer is proposed by Vaswani et al. [21] that introduces the Transformer architecture, a groundbreaking neural network model designed for sequence-to-sequence tasks, such as machine translation. The paper proposes a novel attention mechanism that allows the model to efficiently capture global dependencies between input and output sequences, revolutionizing the field of natural language processing and finding applications beyond NLP.

As shown in Figure 2.2, the main components of a Transformer can be summarized as follows:

1. **Self-Attention Mechanism**: The central innovation of the Transformer is the self-attention mechanism, which replaces recurrent and convolutional layers in traditional

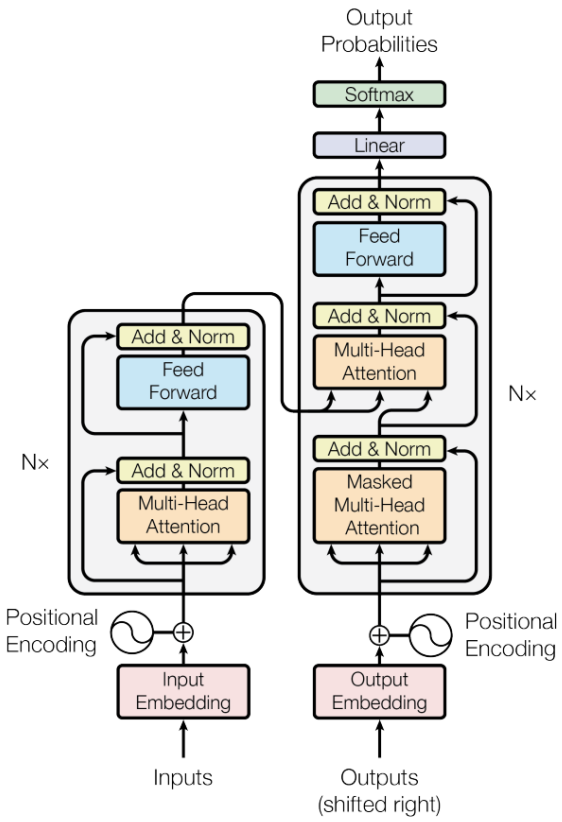


Figure 2.2: The Transformer

sequence models. This mechanism enables the model to weigh the importance of different positions in the input sequence when making predictions for a given position. The attention score is calculated by a compatibility function (dot product in the original paper) followed by a softmax normalization.

2. **Scaled Dot-Product Attention**: As shown in Figure 2.3, the scaled dot-product attention computes attention scores between a query vector Q and key vectors K to obtain weighted values V. The scaling factor $\sqrt{d_k}$, where d_k is the dimension of the key vectors, helps stabilize gradients during training:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

3. **Multi-Head Attention**: The Transformer employs multi-head attention, which

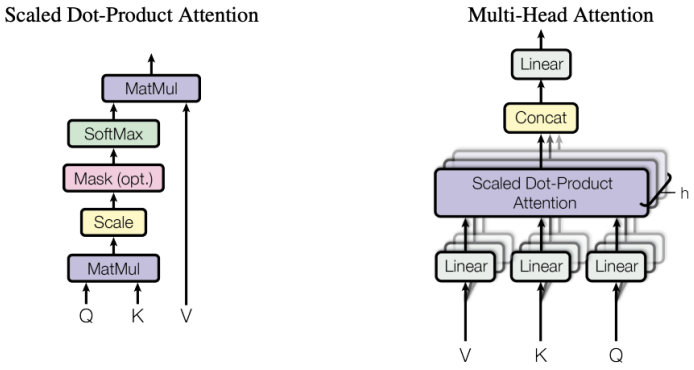


Figure 2.3: (left) Scaled Dot-Product Attention. (right) Multi-head attention consists of several attention layers running in parallel.

performs self-attention with multiple sets of linear projections (heads). The outputs from different heads are concatenated and linearly transformed to produce the final attention output. The right one in Figure 2.3 shows how multi-head attention is produced by several scaled dot-product attention.

$$\begin{aligned} \text{MultiHead}(Q,K,V) &= \text{ Concat } (\text{head}_1,\ldots,\text{ head }_\text{h})\,W^O \\ \text{where head } &= \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{2.2}$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

4. **Positional Encodings**: Since the Transformer does not inherently encode the order of sequence elements, positional encodings are added to the input embeddings to provide information about their positions in the sequence. These positional encodings have a specific mathematical form involving trigonometric functions. They use sine and cosine functions of different frequencies:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{\text{model}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_{\text{model}}}\right) \end{aligned} \tag{2.3}$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$.

5. **Encoder-Decoder Architecture**: The Transformer architecture comprises an encoder and a decoder. The encoder processes the input sequence, and the decoder generates the output sequence. Both encoder and decoder consist of stacks of layers, each containing self-attention and feed-forward neural network sub-layers.

6. **Position-wise Feed-Forward Networks**: The Transformer introduces position-wise feed-forward networks to each position in each layer. These networks consist of fully connected layers and serve as a non-linear transformation of the representation. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max\left(0, xW_1 + b_1\right) W_2 + b_2 \tag{2.4}$$

The Transformer architecture's success in NLP tasks has inspired its adaptation for time series tasks, where sequences of data points are processed. Time series data have temporal dependencies, and Transformers are well-suited to capture long-range relationships between data points. Here's how Transformers are applied to time series tasks:

1. **Encoding Time Series**: In the context of time series, each data point becomes an element in the sequence. The encoder's self-attention mechanism captures temporal dependencies, enabling the model to learn patterns and relationships across different time steps. Liu et al. [11] proposed a Gated Transformer, a two-tower Transformer with each tower working on time-step-wise attention and channel-wise attention. To merge the features of the two towers, a learnable weighted concatenation (also known as 'gating') is used.

2. **Forecasting and Prediction**: The decoder can be utilized to generate future values in a time series or predict missing values. It takes as input the encoded representation of the past and generates future values step by step. For forecasting tasks, the model can be autoregressive, where the predicted values become input for predicting the next time step. Zhou et al. [29] designed an efficient transformer-based model named Informer tailored for long sequence time-series forecasting (LSTF), like predicting extended electricity consumption patterns. Overcoming challenges of quadratic complexity and memory usage, Informer features a ProbSparse self-attention mechanism with O(Llog L) complexity (L represents the length of the input sequence), a self-attention distilling technique for handling long sequences, and a generative style decoder for faster predictions. Evaluated on large-scale datasets, Informer outperforms existing methods in LSTF, providing an innovative solution for precise and efficient forecasting of extended time-series data.

Another important technique used in this thesis is teacher forcing. Teacher forcing [2] is a training technique commonly used in sequence generation tasks, such as machine translation or text generation, when training sequence models. It involves using the actual or "true" previous output token (word or character) from the training data as input for the next time step, rather than using the predicted token generated by the model itself.

Chapter 3

Data Collection and Pre-processing

Data is one of the most essential parts of machine learning tasks, serving as the foundation upon which models are trained, evaluated, and refined. The success and effectiveness of machine learning algorithms heavily depend on the quality, quantity, and diversity of the data they are trained on. In this chapter, we will discuss the data collection process, including the participant study to obtain paid volunteer drivers and the collection, and processing of the resulting data, and a detailed description of the sensors used in the vehicle.

3.1 Sensor Description

The experiment employed a 2009 Cadillac SRX as the test vehicle, which was modified and provided by the industry partner Magna International. It was enhanced with external sensors to gather data on the vehicle's surroundings and its condition. The types of external sensors and internal components utilized in this study are provided in Table 3.1. Additionally, the vehicle was equipped with a computer console featuring a wireless keyboard and monitor. This setup enabled a member of the UWaterloo study team to record observations while collecting data. This approach was employed and tested during a few initial drives.

Table 3.1: List of External Sensors and Inner Components

Number	Sensors/Components
2	Delphi SRR2
1	Delphi ESR
1	Velodyne Lidar
1	GlobalSat GPS Antenna
1	Garmin GPS Antenna
1	VLT Camera
1	VLT Logger
1	Data Logger
1	GPS antenna
1	Processing Unit

3.2 Data Collection

The study consisted of collection data from approximately 50 participants driving a predetermined route twice for approximately 1.5-2 hours. In this thesis, we use 15 drivers after feature alignment and selection. Each driver was asked to drive the route twice. Due to the COVID-19 pandemic, one particular driver drove the route 7 times. This situation is discussed in detail in the following data-splitting sub-section.

3.2.1 Route Planning

The route, depicted in Figure 3.1, consisted of three sections: **urban streets**, **freeways**, and **rural roads**. The Waterloo region proved to be a suitable area for gathering data due to its bustling city center caused by the existence of two universities, upgraded highways in response to the expanding population, and the availability of country roads beyond the city's core.

The urban environment, encompassing city roads, presented various challenges and scenarios for both the ego vehicle and the participants. These included encounters with pedestrians, individuals crossing the road illegally, designated pedestrian crossings, traffic lights, stop signs, cyclists, roads impacted by ongoing construction, parked cars, turns, narrow roads, and more.

The highway sections consisted of curved roads, multiple lanes, on and off-ramps, in-

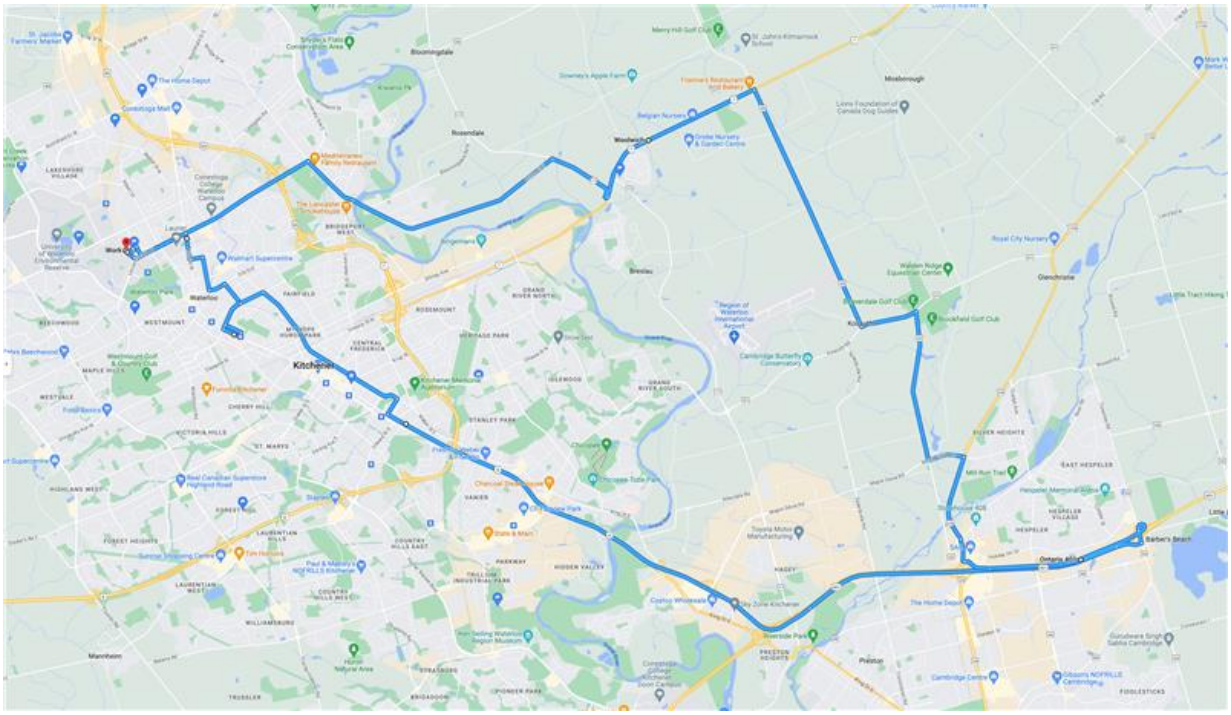


Figure 3.1: Final route used for participant drives around the Waterloo region.

terchanges facilitated by on-ramps, different speed limits, and divisions separating the opposing lanes. The country roads featured stop signs, yield signs, roundabouts, traffic lights, single-lane two-way roads, turns, varying speed limits, traffic, and other factors.

Highway Selection

A dense urban area requires drivers to navigate complexities such as turns and intersections and to be mindful of the behavior of public transit vehicles, cyclists, and pedestrians around them. A drive along a stretch of highway, meanwhile, requires following a comparatively simple protocol and involves much less situational complexity. For this reason, highway driving is widely considered to be an easier task than city driving for autonomous vehicle operation, however, it's more difficult to identify a particular driver.

In light of this, we also chose to limit the portions of our dataset used for our first machine-learning experiments to sections of highway driving. Specifically, we used GPS coordinates to filter out stretches of highway along our driving routes, excluding the initial merges onto the highway and areas where drivers could be expected to change lanes to exit

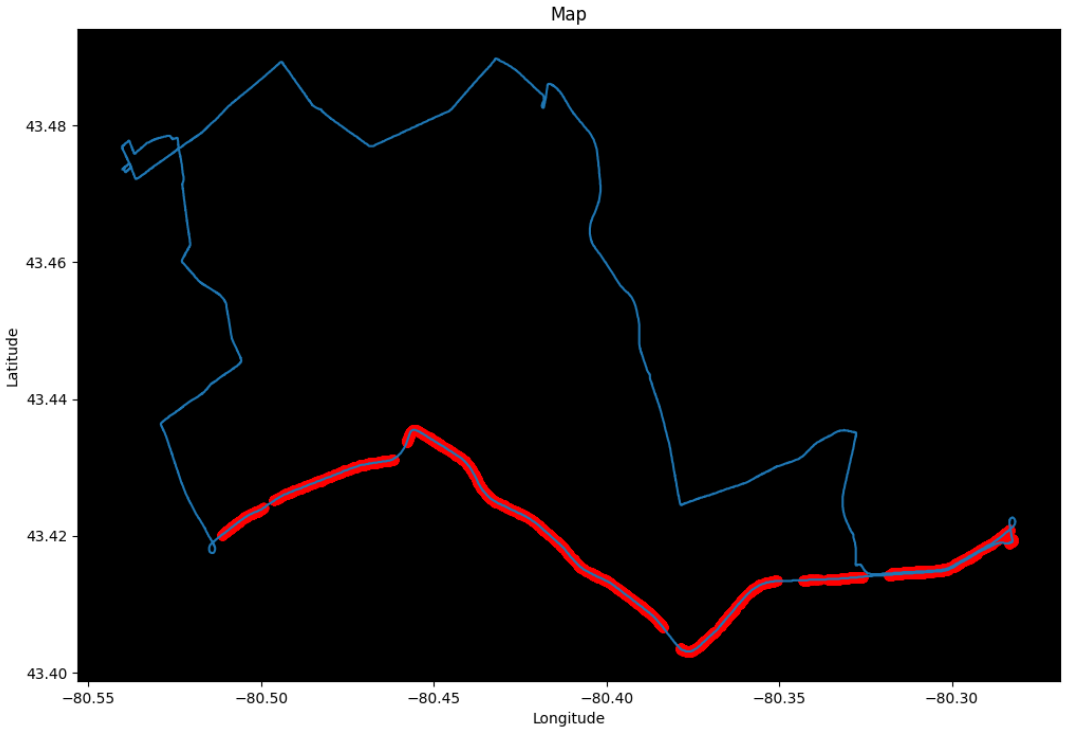


Figure 3.2: Subset of the route showing the selected highway sections only.

the highway. All continuous segments within these GPS ranges were extracted from each participant drive. An example of this subset of the route is shown in Figure 3.2.

3.3 Description of the Collected Data

The data gathered throughout the participant study encompasses various types, including the VLT object stream, CAN-BUS data, RADAR data, Lidar data, and GPS data.

Each drive yields approximately 310 GB of VLT data. Additionally, each route generates around 300 GB of raw VLT data files, which then require processing. The processed data results in approximately 10 GB of processed CSV files. These CSV files undergo further processing based on information provided by Magna to UW. Subsequently, these files are combined, compressed, and consolidated into a single parquet file. It's important to note that multiple parquet files are generated since the processed CSV files produce several files for each category.

Moreover, each drive produces approximately 5 GB of CAN data. As mentioned earlier, the majority of the CAN data is included in the parquet files. The CAN bus data is presented in JSON format. Since the RADAR data is not included in the VLT dataset, it can be extracted from the CAN data.

The Lidar data is provided in .pcap files, with each complete drive typically ranging from 8 to 10 GB. These files are later converted into bag files using ROS (Robot Operating System). OpenPCDet, a publicly available repository, is used to further process the data. OpenPCDet is a PyTorch-based library that enables 3D object detection. It offers various state-of-the-art methods, and for this project, the PointPillars method was chosen. This method allows for efficient encoding of point cloud data into a usable format. The objective is to generate a series of object identifiers that include the center of a detected object and its cuboid bounding box vertices.

To ensure privacy and confidentiality, the dataset has been labeled in a way that removes any personal identifying information regarding the participants. The dataset labels specify the routes taken, the number of participants, and the date and time of data collection.

3.3.1 Feature Description

VLT and Lidar data are used in this work and RADAR data has not been used due to time alignment problems. In this section, we'll dive into detail about the features of extracted VLT and Lidar features. The processed data from VLT generates more than 740 attributes. However, not all these attributes contain useful information for behaviour learning purposes. Hence, a pre-processing and feature engineering step was needed before commencing the learning process. PCA is applied for feature selection, and a total of 174 input VLT features are selected after.

To ensure compatibility between the Lidar and VLT sensors' data when training models using both sources, we downsample the VLT sensor's data since its sampling rate is lower than that of the Lidar. For the VLT sensor, we utilize the data contained in the parquet files, while for the Lidar, we rely on the detections provided in the pickle files, Python's built-in object encapsulation method.

The pickle files store information about the detected objects, including their 3D location, dimensions of their bounding boxes, and their heading angle. These objects belong to one of the following classes: cyclist, truck, and car. In each frame, we focus on the bounding box features of the closest cyclist, car, and truck relative to the ego vehicle. This results in a total of 21 Lidar features available in each frame.

In both the driver identification and lane positioning tasks, our input comprises a total of 174 VLT features and 21 LiDAR features. The process of selecting the most informative VLT features involves a three-phase filtering approach. The initial phase involves a manual examination of feature exploration results to exclude features devoid of meaningful information. An instance of such features is when they maintain a constant value irrespective of other feature values, necessitating an understanding of the feature and its potential values for this manual selection. In the second phase, domain knowledge is employed to eliminate irrelevant features, with the specifics tailored to each behavior. The third phase incorporates correlation analysis to identify features with highly correlated values to others. These redundant features are excluded, as their information can be derived from other features without enhancing model uniqueness. A visualization of the correlation analysis is presented in Figure 3.3.

In the context of the lane-keeping task, our objective is to jointly predict both speed and angle. Recognizing that certain VLT features might inadvertently provide clues about the targets, we opt to eliminate a subset of these features. As a result, we finalized our feature selection process with 142 VLT features and 21 LiDAR features, which will be employed for training purposes.

3.3.2 Training-testing Partitioning

As mentioned in previous sections, each driver was asked to drive two loops of the route. So, commonly, we split one route for training and one for testing. This route-based partitioning is also to avoid data leakage problems. Sampling from input sequences requires overlapping which will be described in Section 4.1.1, and thus data leakage happens when we split through samples rather than routes. Thus, choosing some of these sequences at random to be training samples and some to be test samples would be inappropriate, as the model would have seen large parts of test sequences previously during training. To avoid this problem entirely, we, therefore, choose to split one route each for training and testing. This method of partitioning data between training and testing was used in all behaviors. The reason why we do NOT perform training-testing-validation partitioning is because we don't have enough data for every single driver. Ideally, we would want to collect three routes for a single driver in order to perform this route-based partitioning for training-testing validation. But we only have two routes just now.

A member of the study team (Driver no. 15 named in Chapter 4) drove the route 7 times in total due to COVID restrictions, and thus, four were selected for training and the rest three for testing for this specific participant. This provides a useful opportunity for

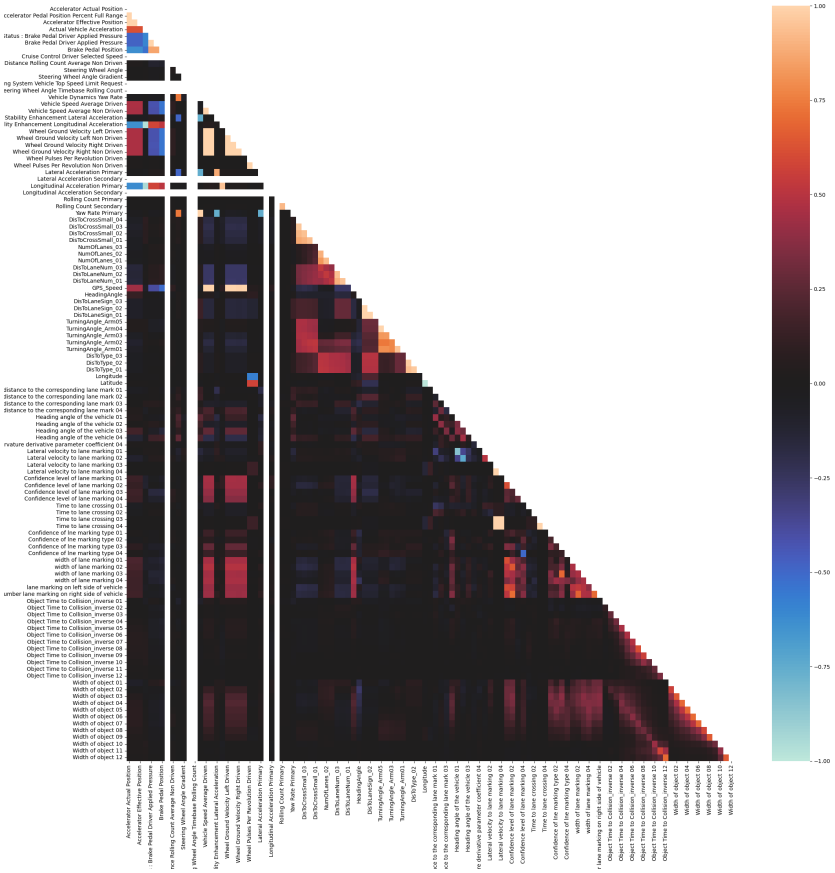


Figure 3.3: Visualization of correlation analysis results before applying PCA

ablation study. We are more interested in how the model will perform as the data amount increases. Therefore, we changed the amount of data driven by that particular driver who drove 7 routes in total. We pick one, two, three, and four routes for training, and we always keep three routes for testing to see how the model performs.

Chapter 4

B1 - Driver Identification

Identifying the operator of a vehicle can be achieved through biometric devices, but there are situations where their usage is not preferred. Instead, a driver identification system can rely on the growing array of sensors found in modern vehicles. These sensors are already utilized in applications like automatic braking, lane departure warning, and blind spot detection. By analyzing data from these sensors, these applications can be enhanced through personalized customization based on the inferred driver's identity.

A well-developed driver assistance system that can interpret a brief segment of sensor data presents engineers with the opportunity to create applications that adapt to individual users. At the personal vehicle level, the system can swiftly identify the driver and adjust vehicle settings accordingly. This adjustment goes beyond just air conditioning or seat position and can assist with vehicle maneuvering based on the driver's habits. A highly accurate driver identification system can also be designed to notify vehicle owners of unfamiliar driving patterns, acting as a deterrent against potential theft. One approach to constructing such a system involves mapping driving behavior to a user-specific representation, which can be utilized for various related tasks. However, a significant challenge is to ensure the system's adaptability to all scenarios since drivers exhibit different behaviors depending on factors such as road type (highways, urban or rural areas) and conditions (straight roads or turns). Developing a capable system relies on extensive training data encompassing various scenarios and different driving styles, which has proven to be challenging to obtain.

This behaviour was selected as the first behavior of interest while developing our initial algorithms and machine learning pipelines. This choice was made for a number of reasons. First, it's the simplest task for driver behavior learning, as it doesn't need future signals as

ground truth like other more complicated driver behavior learning tasks. It's easily defined by a given snippet of driving input and a simple driver-id prediction. Second, the driver's ID would be useful if it's predicted advanced for building a personalized driver assistance system.

4.1 Problem

In this section, we present the formal definition of the driver identification problem. The main objective is to predict the driver ID, which is treated as a classification task.

4.1.1 Definition of Input Sequence

The data for each participant is treated as a discrete time series, containing 174 VLT features and 21 LiDAR features collected by the vehicle's sensors at regular time intervals. At any given time t, we use a history of T_0 seconds as input features for our model and aim to predict the corresponding driver ID. To achieve this, we utilize input features from frames $t - f * T_0$ to $t - 1$, where f represents the frequency of the driving data. Each participant's drive can be divided into approximately 10,000 $f * T_0$-frame sequences. The interval T_0 varies from 30 seconds to 180 seconds, as we investigate how the length of history influences different driver behavior learning tasks as just one of our experiments.

To maximize the number of data samples from each drive, we treat each contiguous $f * T_0$-frame sequence as a single sample. We define these samples with a stride (or offset) of 1. Thus, if the sequence $t - f * T_0 : t - 1$ is considered a valid sample, we also consider $t - f * T_0 - 1 : t - 2$ and $t - f * T_0 + 1 : t + 1$ as valid samples as well.

4.2 Model Overview

The general model architecture of B1 is illustrated in Figure 4.1. It comprises two main components:

1. N-layered Temporal Encoder: This component takes both VLT (Vehicle Localization and Tracking) and LiDAR data as input. It processes the input data through N layers to create an output that captures relevant information from all the inputs.

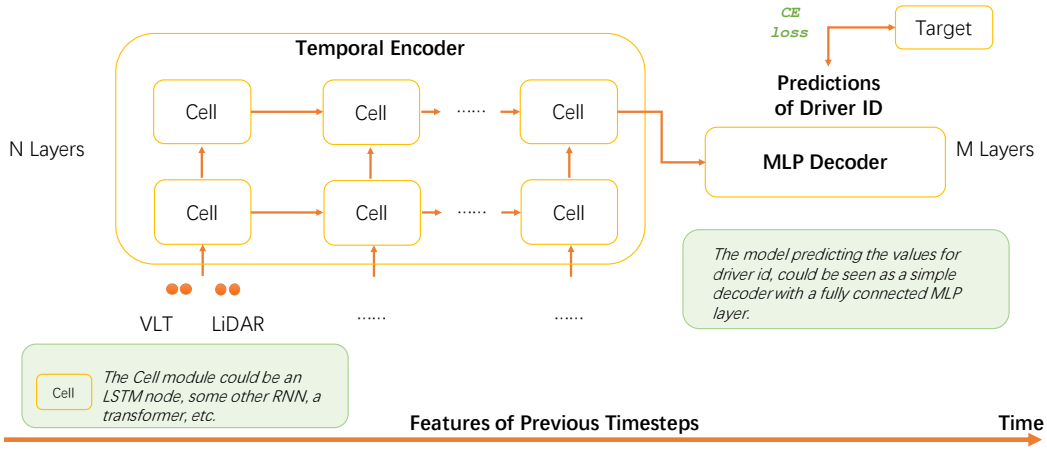


Figure 4.1: General model structure of B1.

2. M-layered MLP Decoder: The output generated by the temporal encoder is then passed to the MLP (Multi-Layer Perceptron) decoder. The decoder utilizes this information to predict the driver ID based on the learned representations from the encoder. Here we simplify the number of layers $M = 1$.

The cell depicted in Figure 4.1 can represent either an LSTM (Long Short-Term Memory) cell or a Transformer block. It should be noted that the direction of the arrows in the figure is merely a representation to illustrate how information flows from lower layers to upper layers. It may not indicate the exact direction of information flow between different cells or blocks. If Transformer blocks are used, the arrows should be depicted as "fully connected" instead of simply going from bottom to top, because Transformers are bidirectional.

4.2.1 Experiment Settings

Designing a deep-learning architecture for a given task is always a complex process with task- and data-specific considerations. A number of important questions, therefore, re-

mained even after these initial design decisions, and a number of architectural experiments were required to refine our model into one best suited for our task.

Sequence Length

History is of great importance in identifying a particular driver. Normally speaking [1], a longer history may help to differentiate a driver from others but it might not be the same in a neural network. In Section 4.1.1, we've mentioned the interval T_0 which varies from 30 seconds to 180 seconds. We pick 30, 90, and 180 seconds as interval, that is, 200, 600, and 1200 frames respectively.

Models

The architecture is of great importance as it affects the model capacity of encoding information. As shown in Figure 4.1, we tried Transformers and LSTMs as temporal encoders since they're the most commonly used ones in sequence data. LSTMs excel at capturing temporal dependencies by using recurrent connections and memory cells, making them suitable for sequential data analysis. They can effectively handle long-range dependencies and variations in time-series data. On the other hand, Transformers leverage self-attention mechanisms to capture global dependencies, enabling parallel processing and capturing contextual information across the entire sequence.

Depth of the Model

Model depth is a crucial aspect of deep learning. Increasing the depth of a neural network allows it to learn more complex and abstract representations from the data. Deeper models have a higher capacity to capture intricate patterns and relationships, enabling them to extract more meaningful features. However, deeper models also pose challenges such as vanishing or exploding gradients, increased computational complexity, and overfitting. Achieving the right balance between model depth and these challenges is essential for leveraging the full potential of deep learning and obtaining accurate and generalizable results. We decided to test Transformers with 2, 4, and 8 layers and LSTMs with 1, 2, 4, and 8 layers.

Dimension of the Model

The dimension of the model d_{model}, which is usually used in Transformers, describes the vector dimension of each frame, named as the number of units per layer in LSTMs. Increasing model width enhances its representational capacity, allowing it to capture more diverse features and learn complex patterns. Wider models can extract more information from the data and exhibit better generalization. However, wider models come with increased computational complexity and memory requirements, and the risk of overfitting. We chose to set $d_{model} = 128$, 256, and 512.

Batch Size

Passing in an entire dataset's training data at once during a training epoch is often impossible due to memory limitations, and therefore an epoch usually consists of groups of training data, called batches, each being passed in one after another until all data has been passed in. Moreover, there are other performance tradeoffs in selecting a batch size. Larger batches, where more samples are passed in at once, may allow an entire epoch of training data to be passed in faster, but they also result in network parameters being altered more slowly during the course of training, as these updates happen after each batch. Smaller batches mean an epoch takes longer to complete, but allows for more frequent, though potentially also more erratic, network parameter updates. A balance between these is preferred, so that the network can train efficiently and stably but also ensure strong performance. We set batch size 64 due to experimental results.

Learning Rate and Scheduler

The learning rate and learning rate scheduler play crucial roles in deep learning. The learning rate determines the step size at each iteration, affecting how quickly or slowly a model converges. Setting it too high can lead to overshooting while setting it too low can result in slow convergence or getting stuck in local optima. On the other hand, the learning rate scheduler adjusts the learning rate dynamically during training, enabling fine-tuning as the model progresses. It helps improve optimization by adapting the learning rate based on factors like loss improvement, epoch number, or training progress, leading to better model performance and faster convergence. We set our learning rate with 1e-3 and picked the StepLR scheduler, which reduces the learning rate by a predetermined factor at regular intervals during training.

4.2.2 Loss Function

Cross-entropy loss is commonly chosen for driver identification due to its effectiveness in handling classification tasks. By measuring the dissimilarity between predicted and actual driver labels, cross-entropy loss encourages the model to assign high probabilities to the correct driver and low probabilities to incorrect ones.

$$CELoss = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(p_{ij}) \tag{4.1}$$

where

- N is the number of samples in the dataset.
- C is the number of classes.
- y_{ij} is the ground truth label (1 if the sample i belongs to class j, 0 otherwise).
- p_{ij} is the predicted probability that the sample i belongs to class j.

Other Hyper-parameters

The head number of Transformers is set to 4. For training time, we trained Transformers for 10 epochs and LSTMs for 30 epochs respectively, to both make them fit properly and save training time. We report the average accuracy of every single driver.

4.3 Results and Discussion

In this section, we present the results and discuss their implications, providing valuable insights into the findings and their significance in the context of the research question.

4.3.1 Main Results

The goal of this task is to correctly identify a driver given a snippet of driving history. For every driver, we use accuracy as our metric $Acc = \frac{Correct\ cases}{All\ cases}$ and report the average accuracy among all the drivers.

Enc	avg(train)	avg(test)
Trm	47.72	24.46
LSTM	93.03	38.08
random guessing	$\frac{1}{15}$	$\frac{1}{15}$

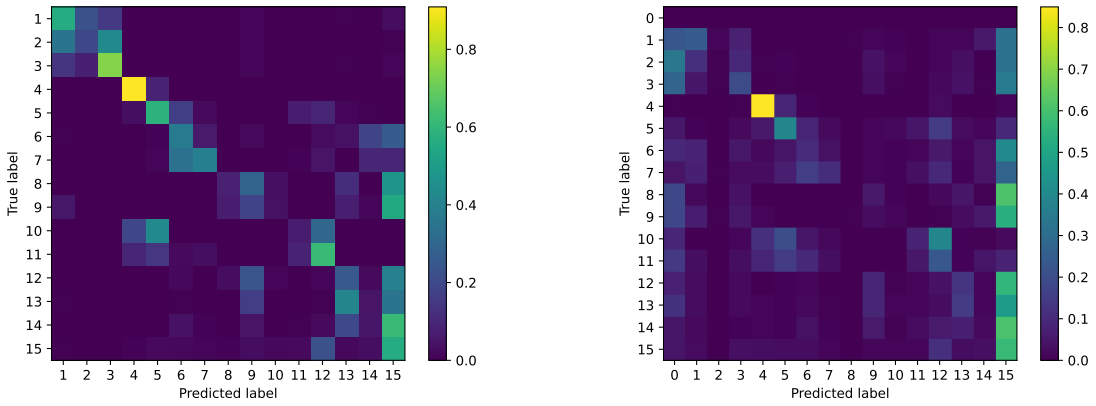
Table 4.1: Accuracy results for 15-class driver identification. We show the results of Transformers and LSTMs with a sequence length of 200, depth of 2, and width of 256. LSTMs outperform Transformers under a two-tailed student t-test at the level of 95% confidence. The accuracy of each driver is used to perform the student t-test.

For this driver identification task, we try different encoders and the decoder is always a multi-class MLP. The results presented in Table 4.1 indicate that Transformers achieve 25% accuracy in the driver identification task, while LSTMs generally perform better with over 35% accuracy. This suggests that LSTMs are more effective at capturing information relevant to modeling specific drivers. The training accuracy of Transformers being below 50% indicates that they are underfitting for this task compared to the approximately 90% accuracy achieved by LSTMs. To gain a better understanding of how these models perform for individual drivers, we visualize the confusion matrix for driver identification using both LSTM and Transformer in Figure 4.2. The figure clearly shows that both models perform reasonably well, but LSTM outperforms Transformer. There are some similarities and differences in the error patterns between LSTMs and Transformers, which will be discussed in Section 4.3.3.

Multi-way Top-k Driver Identification

In real-world scenarios, vehicles are typically used by a small group of drivers, and a deployed model should be able to identify the driver from this limited pool. To evaluate this, we conducted experiments to determine the average accuracy for various combinations of drivers, such as pairwise (2 drivers), 3 drivers, 5 drivers, and the full set of 15 drivers[24, 1]. Calculating the results becomes challenging due to the large number of possible combinations, denoted as $\begin{pmatrix} 15 \\ n \end{pmatrix} = \frac{15!}{n!(15-n)!}$, for a group of n drivers. To address this, we randomly selected subsets for evaluation, and the specific details can be found in Table A.1. The evaluation includes four metrics: top 1, 2, 3, and 5 accuracies[1]. Top-k accuracy measures how often the model correctly identifies the target driver among the k drivers with the highest probabilities. The results are presented in Table 4.2.

When using the full set of drivers, LSTM significantly improves its accuracy from



(a) LSTM (b) Transformer

Figure 4.2: Confusion matrix for driver identification, normalized over the true labels (e.g. rows).

	Transformer				LSTM			
	top-1	top-2	top-3	top-5	top-1	top-2	top-3	top-5
2-way	50.85	–	–	–	46.89	–	–	–
3-way	46.55	54.60	–	–	46.28	47.67	–	–
5-way	40.88	52.39	55.04	–	-	-	47.68	–
15-way	24.46	35.02	43.87	59.05	38.03	55.40	68.59	80.43

Table 4.2: n-way top-k accuracy on Transformers and LSTMs for driver identification.

38.03% with top-1 to 80.43% with top-5. On the other hand, Transformers achieve 59.05% accuracy with top-5, indicating that misidentified cases are still quite likely to appear among the top predictions for Transformers, while LSTMs have more such cases. In the case of Transformers, only 19% of misidentified cases rank second and third, whereas in LSTMs, this number is 30%. This suggests that misidentified cases tend to rank higher in LSTMs compared to Transformers.

Even though LSTMs reach over 80% accuracy in 15-way top-5 evaluation, it is essential to note that their performance is still similar to random guessing when considering 2-way top-1 accuracy, which is slightly lower than 50%. Upon calculating the average accuracy, it becomes evident that there are some hard drivers for whom the accuracy is almost 0.

Enc	Dep	avg	75%	med	25%
Trm	2	24.46	33.04	**14.56**	**5.29**
Trm	4	**24.70**	**34.56**	13.85	4.93
Trm	8	24.49	33.01	14.39	4.72
LSTM	1	34.27	55.08	30.91	6.45
LSTM	2	**35.15**	**64.18**	**32.57**	**7.96**
LSTM	4	16.48	0.81	0.00	0.00

Table 4.3: We show the average, median, 75 percentile, and 25 percentile accuracy of every single driver. Models are trained under a sequence length of 200 and a width of 256.

These challenging cases will be identified and discussed in detail in Section 4.3.3.

4.3.2 Analysis on Hyper-parameters

In order to answer several questions, we evaluate the performance of different multi-modal time-series architectures with different hyper-parameter settings for several driver identification scenarios on the full set of 15 drivers.

Do we need a deeper model? No, a deeper model might not be beneficial for the driver identification task. When increasing the depth of Transformers from 2 to 4, the average score only slightly improves from 24.46% to 24.7% but then decreases to 24.49% when the model becomes even deeper. As for LSTMs, training fails when the depth exceeds 2, and the reasons for this failure will be discussed in the following section. Interestingly, Transformers exhibit a different trend compared to LSTMs when considering the 75th percentile, median, and 25th percentile accuracy. With a depth of 4, Transformers achieve the highest accuracy at the 75th percentile but not at the median and 25th percentile. This indicates that they tend to perform better in cases where the accuracy is already high but struggle to improve accuracy for more challenging cases. On the other hand, LSTMs perform better in both high and low-accuracy cases.

Do we need a wider model? The necessity for a wider model determines the model's capacity. The highest performance for LSTM is achieved at 36.95% and 38.25% when the width is increased to 256 but drops to 35.69% and 36.40% with a width of 512 when we have an input sequence with a length of 1200 and 600. This indicates that an excessively wide model can be detrimental to performance in the case of LSTMs. Similar patterns

Enc	Seq	Width		
		128	256	512
Trm	600	24.47	**24.64**	24.63
Trm	1200	24.73	**25.05**	24.79
LSTM	600	38.07	**38.25**	36.40
LSTM	1200	34.67	**36.95**	35.69

Table 4.4: Effect of width on average accuracy with a depth of 2.

can be found in Transformers as well. Therefore, determining the appropriate width of the model should be based on the specific requirements and capacity of the model architecture.

Failure of LSTMs on the depth of 4

The reason behind LSTMs failing with a depth of 4 can be understood by examining the training loss. In Figure 4.3, the training loss of LSTMs with various widths and depths for input data is visualized. As the model depth increases, the model converges and reaches its capacity earlier during training. However, when the depth reaches 4, LSTMs are unable to learn anything from the data, even when different widths are tested. As illustrated in Figure 4.3, the LSTM architecture with a depth of 4 exhibits an inability to reduce losses over an extended training duration.

This failure is particular to LSTMs and does not occur with Transformers. The difference indicates that the losses in LSTMs cannot be effectively traced back to the individual layers of the model, which might hinder the learning process and cause difficulties in optimizing the deeper LSTMs. Further investigation is needed to understand the specific reasons behind this behavior and to potentially identify ways to overcome this limitation for LSTMs in the driver identification task.

4.3.3 Case Study

Highway Only

As previously mentioned, driving along a stretch of highway entails adhering to a relatively straightforward protocol and encountering less situational complexity. Consequently, it becomes more challenging to identify a specific driver since most drivers exhibit similar behaviors to one another.

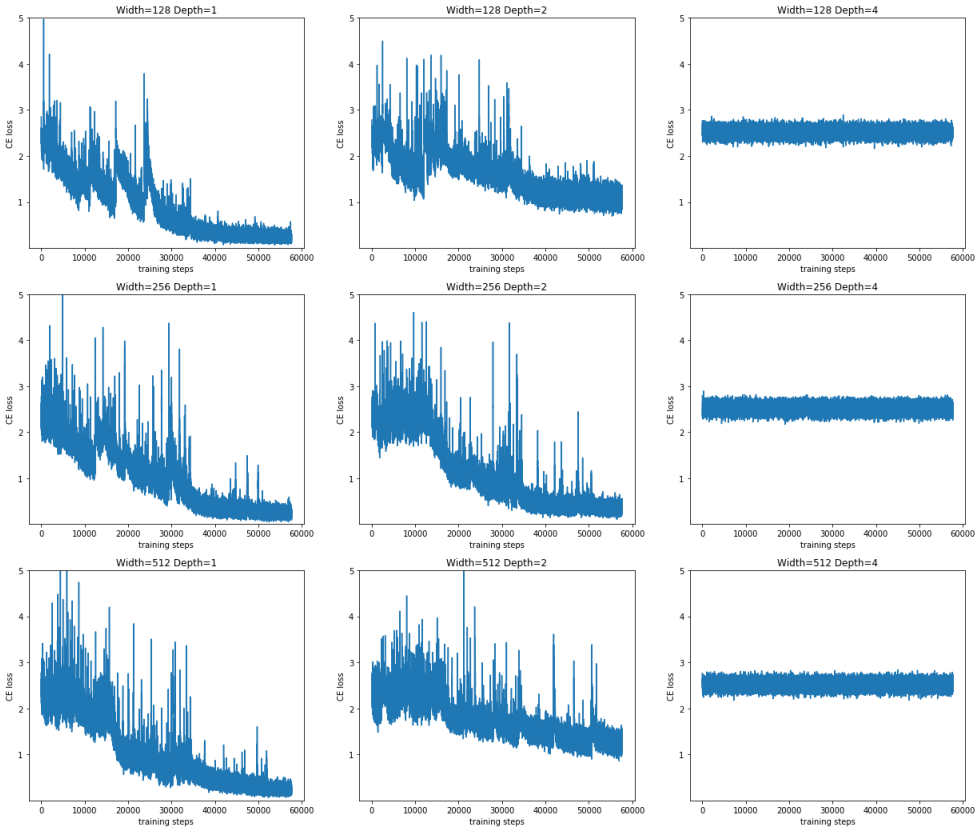
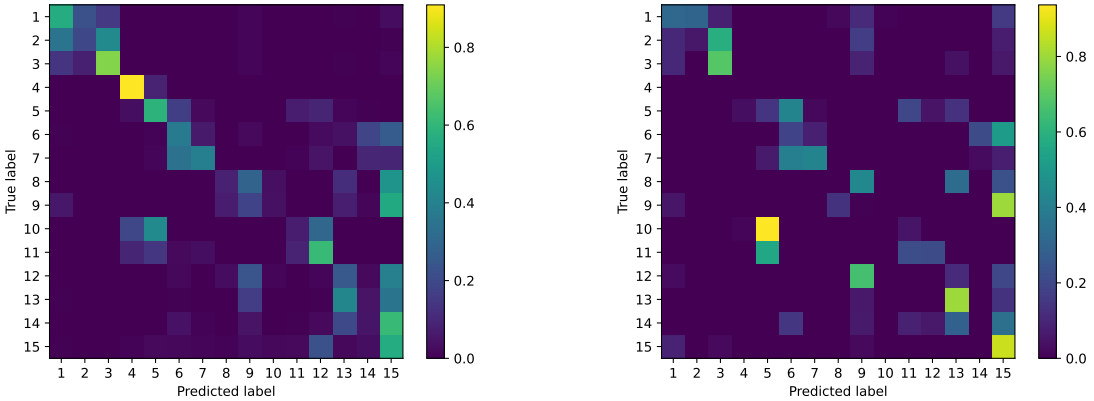


Figure 4.3: Training loss of LSTMs for driver identification with a sequence length of 1200.

Enc	Full Route	Highway
Trm	24.46	14.09
LSTM	38.08	26.47

Table 4.5: Accuracy of Transformers and LSTMs with highway only.

From Table 4.5, we observe a decrease in performance by 12% and 10% for LSTM and Transformer, respectively. This finding supports our previous speculation that drivers tend to exhibit similar behaviors on highways, making it more challenging to distinguish them accurately. To provide a clearer illustration of the distribution, we present the confusion matrix for both scenarios. Generally, these two scenarios exhibit similar distributions, often misidentifying drivers 9, 13, and 15. However, some differences are notable. For



(a) Complete Route (b) Highway Only

Figure 4.4: Confusion matrix of LSTM testing on highway only.

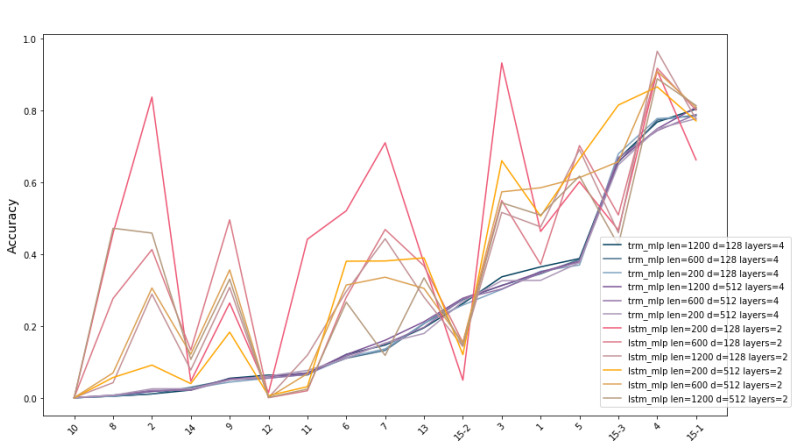
instance, in the case of driver 11, the model more frequently predicts label 5 on highway roads compared to the full routes, suggesting that this driver is more easily identifiable on city roads. On the other hand, for driver 13, the model performs better on highways than on city roads.

Analysis on each individual driver

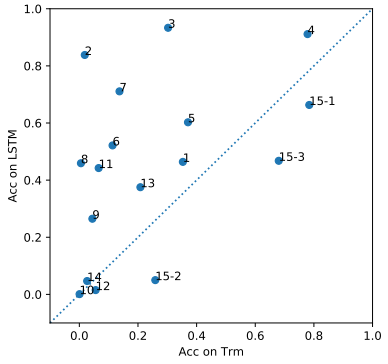
Our main focus here is on comparing the predictions of Transformers and LSTMs for each individual driver. To achieve this, we organize the results by reordering the driver IDs in ascending order by accuracy for Transformers. From Figure 4.5b, when considering easily identifiable drivers, those with an accuracy of over 40% using Transformers, we find that LSTMs tend to perform better in identifying them accurately. However, discrepancies arise when dealing with difficult-to-identify drivers. For drivers 10, 14, and 12, both LSTMs and Transformers struggle to make accurate predictions. On the other hand, for drivers 9, 2, and 3, Transformers perform poorly, while LSTMs demonstrate better identification capabilities.

4.3.4 Single Driver Problem

As previously discussed, we have a participant (Driver no, 15) who drove the route 7 times due to COVID restrictions. For our experimentation, we selected four routes for training and the remaining three for testing. Our primary interest lies in understanding the model's



(a) Results for each individual driver.



(b) Results for each individual driver on LSTM and Transformer.

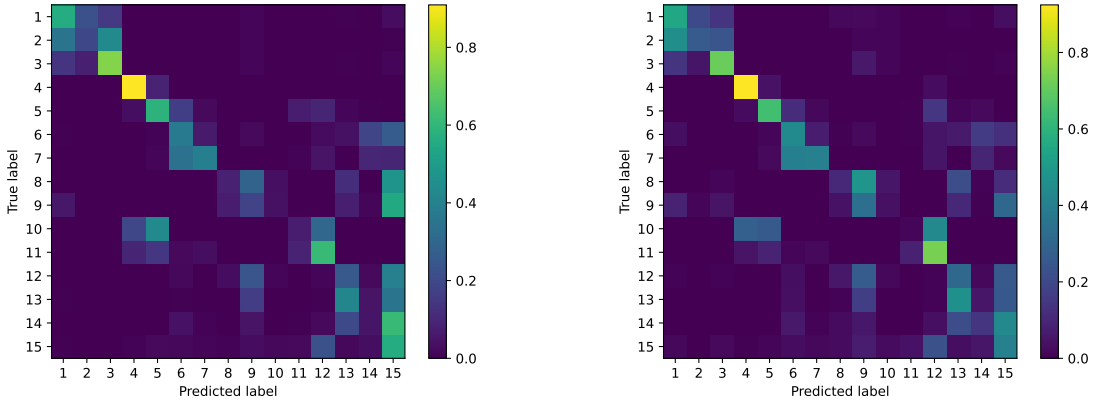
Figure 4.5: Analysis on each individual driver.

Enc	Routes	Seq	d_{model}	Depth	15-1	15-2	15-3	15 Avg.	Avg.
LSTM	4	200	256	2	78.34	13.92	79.84	57.37	34.43
	3				78.42	4.75	44.61	42.59(↓ 25%)	39.41
	2				62.33	3.77	23.1	29.73(↓ 47%)	37.84
	1				49.18	4.4	14.22	22.6(↓ 60%)	41.95
Trm	4	200	256	2	78.13	25.85	69.32	57.77	18.69
	3				75.01	20.11	43.4	46.18(↓ 20%)	20.45
	2				63.94	14.57	40.7	39.74(↓ 31%)	21.39
	1				48.22	7.79	16.44	24.15(↓ 58%)	22.9

Table 4.6: Accuracy on the single-driver problem.

performance as the amount of data increases. To explore this, we vary the data amount from driver 15. We train the model using one, two, three, and four routes while always keeping three routes for testing to evaluate its performance. The results are presented in Table 4.6.

We observe that the accuracy of both LSTM and Transformer drops by more than 50% when using only one route compared to when using four routes for training. Thus, we



(a) 4 Routes for training

(b) 2 Routes for training

Figure 4.6: Confusion matrix on the single-driver problem.

can confirm that, as expected, more data for a specific driver leads to better performance on that driver. However, we also investigate whether this increase in data for one driver causes a drop in accuracy for other drivers. For this, we plot in Figure 4.6 the confusion matrix using 4 routes and 2 routes in the training set. It becomes apparent that with 4 routes, the model tends to misidentify more cases as driver 15. The average accuracy for all drivers indicates that focusing on gathering more data for a particular driver can have a detrimental effect on overall performance. For example, when using only 2 routes for training, the model predicts driver 9 more accurately but predicts driver 15 less accurately. Additionally, it is possible that some misidentified cases might be predicted as other drivers instead of driver 15, even when using less data for driver 15. For driver 7, the model predicts fewer instances as driver 15 but more as driver 6 when using 2 routes for training.

4.3.5 Ablation Studies

During the inference phase, we conducted an analysis of feature contribution by manipulating certain features in the input data. Specifically, we replaced the values of some features with their mode value and collected the results. We set the real Lidar bounding box features to all zeros, implying that no objects, including cars, trucks, and bicycles, were detected. This allowed us to examine how the presence of surroundings contributes to identifying a particular driver.

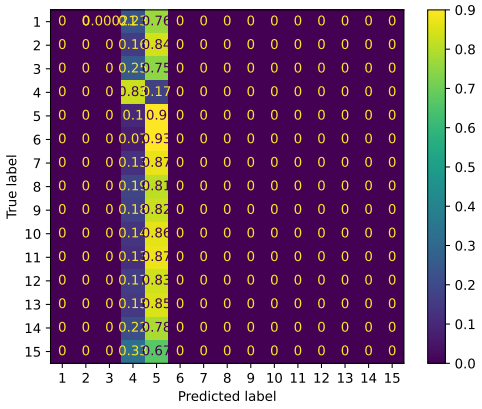


Figure 4.7: w/o Lidar data

In Figure 4.7, which represents the confusion matrix with the lidar replacement, the output prediction is always 5. This outcome suggests that lidar data plays a crucial role in the model's performance, and the absence of surroundings significantly impacts its ability to make accurate predictions.

However, it is noteworthy that driver no. 4 can still be identified despite the lidar replacement. This observation is likely due to the fact that this driver's routes were recorded on a day with good traffic conditions, leading to driving patterns that are relatively independent of their surroundings. In contrast, for most other drivers, the presence of surroundings captured by lidar data appears to be vital for effective identification.

Chapter 5

B2 - Lane Positioning

This chapter delves into the concept of "lane positioning," which refers to how a vehicle is positioned within a lane—either in the center, on the right, or on the left. Drivers choose specific lane positions to adapt to potential issues and create adequate space between their vehicles and potential hazards. Typically, lanes on highways or streets are about twelve feet wide (3.65m), whereas the average vehicle's width is six feet, allowing for approximately six feet of maneuvering room within the lane.

Lane positioning is relevant not only to motorcyclists and cyclists but also to car drivers who need to learn how to position themselves correctly within a lane. It's not simply a matter of staying in the center or keeping to the right; different driving situations call for different lane positions.

This behavior, referred to as "Subject Vehicle Position in the lane," focuses on how the driver positions their vehicle within the lane while taking into account driving speeds and road conditions. The DBL model utilizes various factors such as speed, lateral and longitudinal acceleration, time of day, weather, and road environment to determine the optimal lane position, considering driving preferences. Maintaining the correct lane position is crucial, as slight adjustments to the right or left within the lane may be necessary to achieve the safest position in certain situations.

In this chapter, a machine-learning approach is presented to understand this behavior. The section explores its relation to other behaviors in the project, describes the approaches used, explains the data preprocessing specific to this behavior, and finally presents the model and experimental results in later sections.

5.1 Problem

Here we precisely define the problem of lane positioning. The predicted output is the vehicle's position in the following time steps, formalized as a sequential regression task.

5.1.1 Input Sequence Definition

Both the features and the window size of input sequences for lane positioning are the same as those used for driver identification, as described in Section 4.1.1.

5.1.2 Output Sequence Definition

To implement the regression task for lane positioning, we normalized the lane width to eliminate the influence of lane width variations. After normalization, the regression task involved predicting the vehicle's position within a range of [0, 1]. In this scale, zero indicates the vehicle positioned at the far left of the lane, while one signifies the vehicle's position at the far right of the lane. The normalization process is as follows:

$$Target = \frac{Dist2Left}{Dist2Left + Dist2Right} \tag{5.1}$$

where $Dist2Left$ and $Dist2Right$ represent distances to the left and right lane markings, which can be obtained from VLT data. The original values for these two features can be positive or negative, indicating the relevant position of the vehicle with respect to the lane markings. We use the absolute value of these two distances to ensure that the "Target" value falls between 0 and 1.

After the normalization process, we obtain the target value for lane positioning. For each contiguous $f * T_0$-frame sequence as input, the output or prediction is a $f * T_0/5$ sequence, which is 1/5 of the input sequence. Since there is only one target value, the shape of the prediction is either $[f * T_0/5]$ or $[f * T_0/5, 1]$. The overlapping strategy for input and output sequences is the same as described for driver identification in Section 4.1.1, using a stride of 1.

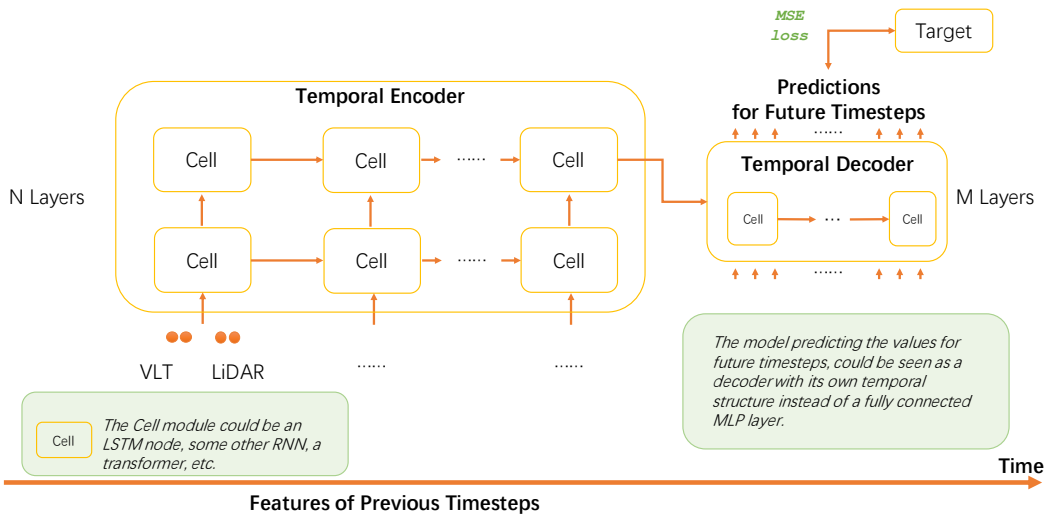


Figure 5.1: General model structure of B2.

5.2 Relation to Driver Identification

The approach employed for lane positioning shares several components with models used for driver identification. We adopt an encoder-decoder framework for lane positioning, where the encoders are the same structures used for driver identification, including Transformers and LSTMs. The input features used are also the same as those for driver identification, consisting of 174 VLT features and 21 LiDAR features. We recognize that lane positions may vary from one driver to another and can serve as a distinguishing signal related to a specific driver. By using the same input features, we can compare the difficulty of this regression task with the classification task of driver identification. The dataset division also follows the same train-test partitioning used for the classification task, allowing us to analyze each individual driver's performance.

5.3 Model Overview

A general model structure for behavior B2 is illustrated in Figure 5.1. For lane positioning, we follow an encoder-decoder structure, where the encoder can be either LSTM or Trans-

formers, similar to the driver identification task. The decoder can be a simple MLP or a more complex structure, such as LSTM or Transformers, to learn the relationships among different future time steps.

5.4 Experiment Settings

We inherit some parts from the driver identification task since both behaviors share the same data and can use similar encoders. However, we made some changes in the width and depth of the models due to computational complexity and training time limitations. For the models, we employ an encoder-decoder structure, using LSTM and Transformer as encoders for feature comparison. The decoder can correspondingly be an LSTM or Transformer decoder or a simpler MLP decoder. We simplify the model by setting $N = M$ in Figure 5.1 when the decoder is the same structure as the encoder (e.g., LSTM-encoder and LSTM-decoder, Transformer-encoder and Transformer-decoder) to minimize hyper-parameters. For a simple MLP decoder, it is a 1-layered MLP, similar to the one used in the driver identification task.

5.4.1 Hyper-parameters

Regarding the hyper-parameters, we made some adjustments for behavior B2. The width is set to 64 and 128, instead of 128, 256, and 512 used in B1, due to the increased complexity of models in B2, which could lead to longer and unacceptable training times with wider models. For depth, we continue to use 1, 2, and 4. Additionally, we explore the impact of history on lane positioning by trying sequence lengths of 200, 600, and 1200. However, most of the experiments are conducted under a sequence length of 200 due to time constraints. The training epoch is set to 10 for all models, and other hyper-parameters like batch size, learning rate, and head number of Transformers remain the same as in the driver identification task.

5.4.2 Loss Function

As lane positioning is a regression task, we apply MSE loss to measure the error between predictions and targets. MSE loss measures the mean squared error (squared L2 norm) between each element in the input x and target y. The loss can be described as:

$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = (x_n - y_n)^2 \tag{5.2}$$

Encoder	Decoder	avg(train)	avg(test)	std
LSTM	MLP	0.042	0.041	0.004
Trm	MLP	0.042	0.041	0.004
LSTM	LSTM	0.013	0.013	0.002
Trm	Trm	0.013	0.013	0.002
w/o teacher forcing				
Trm	Trm	0.001	0.014	0.003
w teacher forcing				

Table 5.1: We collect MSE loss for every single driver and average them. We also show the standard deviation of MSE losses amongst drivers. Results are under the best hyper-parameters.

5.5 Results and Discussion

In this section, we present the results and discuss their implications, providing valuable insights into the findings and their significance in the context of the research question.

5.5.1 Main Results

From Table 5.1, it is evident that all models achieve similar test losses as training losses, suggesting minimal overfitting. This contrasts with the driver identification task, where the corresponding models often exhibit test accuracy below 50% compared to their performance on the training set. When comparing the results of a simple MLP decoder with those of LSTM and Transformer decoders, the latter achieves lower Mean Squared Error (MSE) loss on the test set.

This is not surprising that the MLP simultaneously predicts values for all future time steps, without considering the relationships among them, while LSTMs and Transformers take such relations into account. This indicates that relationships between future time steps play a significant role, necessitating the design of a suitable structure to learn and capture them effectively. With an MLP decoder, there is not much difference between a Transformer encoder and an LSTM encoder, which is different from the driver identification task. It shows that for the lane positioning task, modeling input is not as important as decoding the information.

The results for the Transformer-encoder and Transformer-decoder with and without

Enc	Dec	Depth	MSE
LSTM	MLP	1	0.041
		2	0.041
		4	0.041
LSTM	LSTM	1	0.013
		2	0.013
		4	0.013

(a) Effect on Depth. Experiments were conducted under a width of 64.

Enc	Dec	Width	MSE
LSTM	MLP	64	0.041
		128	0.041
		256	0.042
		512	0.042
LSTM	LSTM	64	0.013
		128	0.013
		256	0.013
		512	0.013

(b) Effect on Width. Experiments were conducted under a depth of 2.

Table 5.2: Analysis on width and depth.

teacher forcing are also presented. Teacher forcing, a technique commonly used in machine translation to prevent prediction biases and expedite convergence, exhibits a different outcome in B2. In this case, teacher forcing leads to worse performance. A plausible explanation for this could be the strong dependency on future time steps, where Transformers fail to correct mistakes accumulated from previous predictions due to a lack of learning in the presence of teacher forcing.

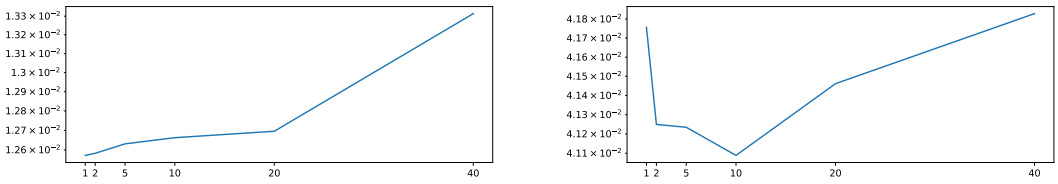
The small value of standard deviation in the results implies that there is not much difference in predicting different drivers, suggesting a consistent performance across various drivers.

5.5.2 Analysis on Hyper-parameters

From Table 5.1, we find that there is a difference between a simple MLP decoder and complicated decoders and a similarity within that category. Therefore, we conduct experiments on LSTM-MLP and LSTM-LSTM to analyze the effect of hyper-parameters of network width and depth of these two architecture types. Among all the combinations, MSE changes by 0.0001 at most, indicating that the lane positioning task is not sensitive to these hyper-parameters. With an LSTM decoder, the loss doesn't even change varied from different widths and depths. It shows that the LSTM decoder is more resilient to hyperparameters changing.

Enc	Dec	Driver Embed	MSE
LSTM	MLP	true	0.041
LSTM	MLP	false	0.041
LSTM	LSTM	true	0.013
LSTM	LSTM	false	0.013

Table 5.3: Effect of driver embedding.



(a) LSTM-LSTM (b) LSTM-MLP

Figure 5.2: Loss on different time steps.

Effect of driver embedding

The input is combined with feature embedding and driver embedding through an add operation. Table 5.3 shows the result with driver embedding and without. The average test MSE is almost the same under these two scenarios. It indicates that different drivers probably follow a similar lane positioning pattern and it's not quite useful to include driver information. On the other hand, it's not easy to differentiate drivers only through their driver data according to the results of driver identification, and thus the driver embedding learned through lane positioning may not be very helpful.

5.5.3 Loss on Different Time Steps

We predict the position of the vehicle for the future 40 time steps given the previous 200 time steps, and we are interested in whether MSE loss is the same on different future time steps. We plot MSE loss on the time step 1, 2, 5, 10, 20, and 40 in Figure 5.2. Figure 5.2a is the result of LSTM-LSTM. MSE loss gradually rises with the future time steps increasing, indicating that there is an error accumulation. However, it's not the same in LSTM-MLP, as it has the lowest error on the 10th time step. An MLP decoder predicts all the future time steps simultaneously, different from an LSTM decoder. The following

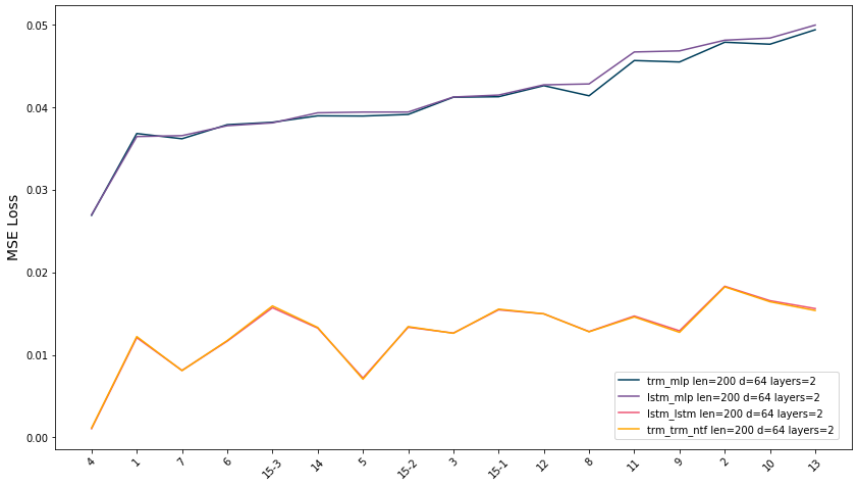


Figure 5.3: Results for each individual driver.

first-time step might not be easiest to predict, probably the second or the following 10. For time steps 10, 20, and 40, loss increases when we predict further future, the same as LSTM-LSTM.

5.5.4 Analysis on Each Individual Driver

Similar to driver identification, one of our focuses is to analyze the performance of structures on each individual driver. We reorder the IDs according to their performance on LSTM-MLP in Figure 5.3. MLP-based decoder structures show a similar pattern, and LSTM-LSTM and Trm-Trm-ntf (ntf means without teacher forcing) share a similar pattern. This is probably because some drivers have a behavior of consistency on lane positioning and thus it's easier for such complicated decoders to learn the relations.

Chapter 6

B3 - Lane Keeping

Lane keeping is an advanced driver assistance system (ADAS) feature that employs a combination of sensors, algorithms, and control mechanisms to ensure a vehicle maintains its intended path within a lane while predicting and adjusting both steering wheel angle and vehicle speed simultaneously. This technology enhances safety, reduces driver workload, and contributes to smoother and more efficient driving experiences.

The system utilizes various sensors, such as cameras, LiDAR, radar, and ultrasonic sensors, to continuously monitor the vehicle's surroundings and lane markings. By analyzing real-time data, the system predicts the optimal steering wheel angle required to keep the vehicle centered within the lane. This prediction is based on factors like lane curvature, road geometry, and vehicle dynamics.

Simultaneously, the lane keeping system considers the vehicle's speed in relation to the road conditions and traffic flow. Through predictive algorithms, it adjusts the vehicle's speed to maintain a safe and appropriate following distance from other vehicles, pedestrians, and obstacles. This dynamic speed control helps prevent sudden deceleration or acceleration, contributing to a smoother and more harmonious flow of traffic.

The integration of steering and speed control enables the lane-keeping system to make seamless and coordinated adjustments, ensuring the vehicle stays within the lane while responding to changing road scenarios. This predictive approach anticipates potential lane departure or speed variations, preemptively taking corrective actions to prevent unwanted lane changes or abrupt speed changes.

Overall, lane keeping with simultaneous steering angle and speed prediction represents a significant advancement in driving automation. By combining accurate sensor inputs,

intelligent algorithms, and precise control mechanisms, it enhances vehicle stability, reduces the risk of collisions, and enhances overall driving comfort and efficiency, making it a critical component of the evolution towards fully autonomous vehicles.

6.1 Problem

In this section, we provide an exact definition of the lane-keeping problem. The objective is to predict the speed and steering wheel angle of a vehicle for future time intervals, treated as a sequential regression task.

6.1.1 Input Sequence Definition

The input sequences for lane keeping are the same as those used for lane positioning and driver identification, as described in Section 4.1.1.

6.1.2 Output Sequence Definition

In order to carry out the regression task for lane keeping, we standardize the representation of speed and steering wheel angle. Speed is scaled in units of km/h, while the angle is measured in degrees. To achieve this, we perform normalization by dividing the speed by 120 and the angle by 360.

Upon completion of the normalization process, we establish the target value for lane-keeping prediction. For every consecutive sequence of $f * T_0$ frames provided as input, the resulting output or prediction becomes a sequence of size $f * T_0/5$, equivalent to one-fifth of the input sequence length. Due to the single target value, the prediction format appears as either $[f * T_0/5]$ or $[f * T_0/5, 1]$. The approach for overlapping input and output sequences mirrors that described for driver identification and lane positioning in Section 4.1.1, employing a stride value of 1.

6.2 Relation to Lane Positioning

Numerous elements are inherited from the lane positioning framework. We adopt an encoder-decoder framework, with both encoders and decoders replicating those utilized

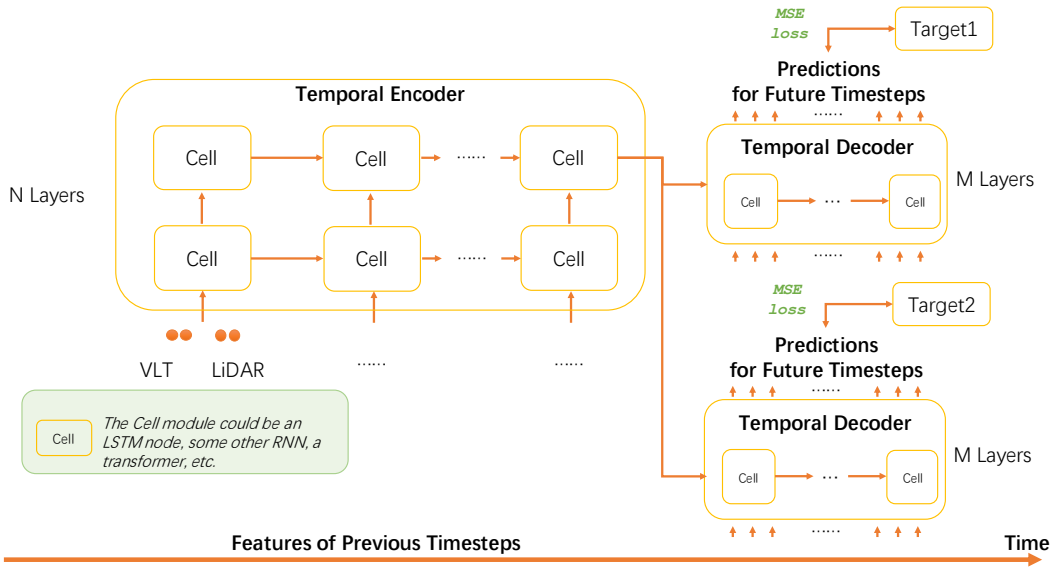


Figure 6.1: General model structure of B3.

in lane positioning. The division of the dataset into training and testing subsets remains consistent, allowing for the individual assessment of each driver's performance. However, adjustments are made to the input features due to potential similarities with target features, which could potentially lead to hints and facilitate unfair advantages. As a result, we utilize 142 VLT features and 21 LiDAR features for the lane-keeping task.

6.3 Model Overview

A broad framework for behavior B3 is depicted in Figure 6.1. In the context of lane keeping, we adhere to an encoder-decoder architecture, mirroring the configuration employed in the lane positioning task. Here, we incorporate two separate decoders to concurrently anticipate both the vehicle's speed and the steering wheel angle. The decoder itself may manifest as a straightforward MLP, or it could manifest as a more intricate design, such as LSTM units or Transformers. This complexity allows the model to capture interdependencies among various upcoming time intervals.

Encoder	Decoder	avg(train)	avg(test)	std
LSTM	MLP	0.049	0.047	0.013
Trm	MLP	0.046	0.040	0.011
LSTM	LSTM	0.049	0.047	0.014
Trm	Trm	0.049	0.047	0.014
w/o teacher forcing				
Trm	Trm	0.004	0.157	0.020
w teacher forcing				

Table 6.1: We collect MSE losses for every single driver and average them. We also show the standard deviation of losses amongst drivers. Results are under the best hyper-parameters.

6.4 Experiment Settings

We inherit all the model structure and hyper-parameter settings from lane positioning.

6.4.1 Loss Function

The loss consists of the MSE loss of vehicle speed and steering wheel angle.

$$l = \alpha * l_{speed} + (1 - \alpha) * l_{angle} \tag{6.1}$$

where α is a hyper-parameter to balance l_{speed} and l_{angle}. Here we simplify $\alpha = 0.5$.

6.5 Results and Discussion

In this section, we present the results and discuss their implications, providing valuable insights into the findings and their significance in the context of the research question.

The main results from the models in the lane-keeping task are highlighted in Table 6.1. When we closely examine the loss values for both the training and test datasets, we find that most models don't show signs of overfitting. This similarity with the lane positioning task is because both tasks involve predicting values and heavily rely on previous information.

However, it's important to mention that Transformers, when used with teacher forcing, tend to overfit in the lane-keeping task due to accumulating errors. Interestingly, unlike

Encoder	Decoder	Loss	Loss on Speed	Loss on Angle
LSTM	MLP	0.047	0.083	0.011
Trm	MLP	0.040	0.070	0.011
LSTM	LSTM	0.047	0.083	0.011
Trm	Trm	0.047	0.083	0.011
w/o teacher forcing				
Trm	Trm	0.157	0.197	0.082
w teacher forcing				

Table 6.2: Losses for predictions of speed and angle.

what we observed in the lane positioning task, making the decoder more complex, especially with MLP decoders, doesn't lead to better outcomes. This could be due to the differences between a simple MLP decoder and a time-dependent decoder, where the latter leverages patterns within the time series. It's possible that the strong connections within labels might be less clear, and Transformers and LSTM decoders might struggle to capture such intricate patterns.

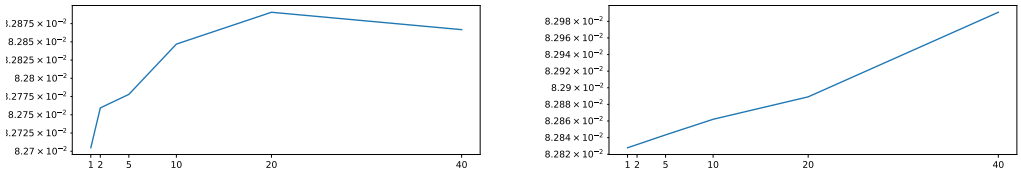
The standard deviation, which is around 0.013, indicates differences in results among different drivers, emphasizing the variations that are specific to each driver.

6.5.1 Analysis on Speed and Angle

The calculated overall loss comes from both how fast the vehicle is moving and the direction it's facing. Usually, the part of the loss related to speed is the biggest. Among the models, the Transformer-MLP is the best in terms of loss because it does really well in speed-related loss. Surprisingly, its performance in predicting steering angles is similar to other models. On the other hand, the Transformer with teacher forcing is the weakest model. It doesn't predict speed and direction very well.

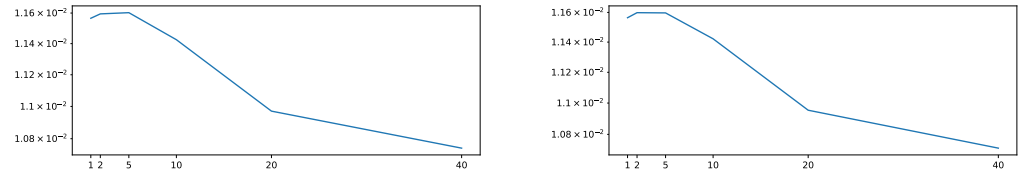
Loss on different time steps

We're working on predicting where the vehicle will be in the next 40 time steps, and we're particularly interested in how evenly the Mean Squared Error (MSE) loss spreads out across different time periods in the future. By separating the loss into parts for speed and



(a) LSTM-MLP (b) LSTM-LSTM

Figure 6.2: Loss for prediction of speed on different time steps.



(a) LSTM-MLP (b) LSTM-LSTM

Figure 6.3: Loss for prediction of angle on different time steps.

angle, we can make separate graphs for the losses for speed and angle at different time steps: 1, 2, 5, 10, 20, and 40. You can see these graphs in Figure 6.2 and Figure 6.3.

When it comes to the loss of speed, both the LSTM-MLP and LSTM-LSTM models show a similar pattern where the loss increases as we predict further into the future. For LSTM-LSTM, the loss consistently and slowly goes up, which is similar to what we saw in the lane positioning task. On the other hand, LSTM-MLP has a big increase in loss in the first 20 time steps, followed by a slight decrease in the following steps.

Now, looking at the loss on angles (shown in Figure 6.3), we notice something interesting: both curves decrease as time goes on. This is quite different from the behavior of speed loss. Angle loss doesn't contribute as much to the total loss. It seems that angle loss happens when the vehicle is turning. This leads to sudden changes in angles in the next frames, which makes it harder for the model to predict these turn-related decisions. The model's inability to predict these turns might be why it makes mistakes in the following frames.

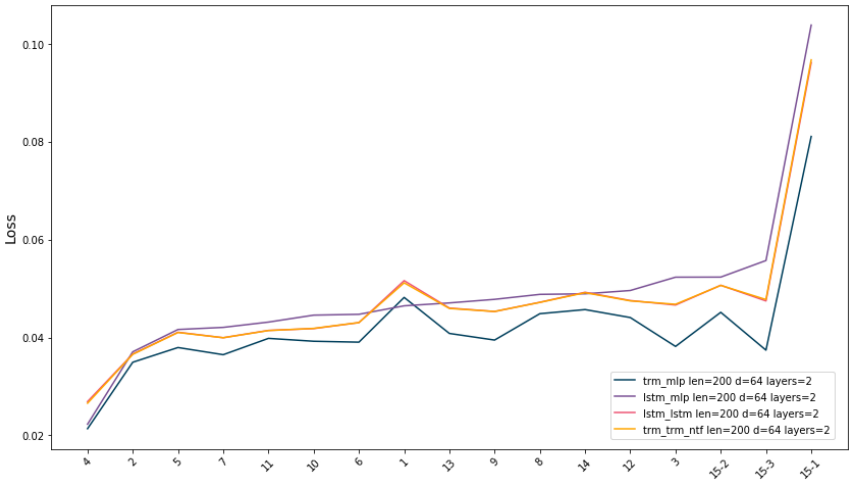


Figure 6.4: Results for each individual driver.

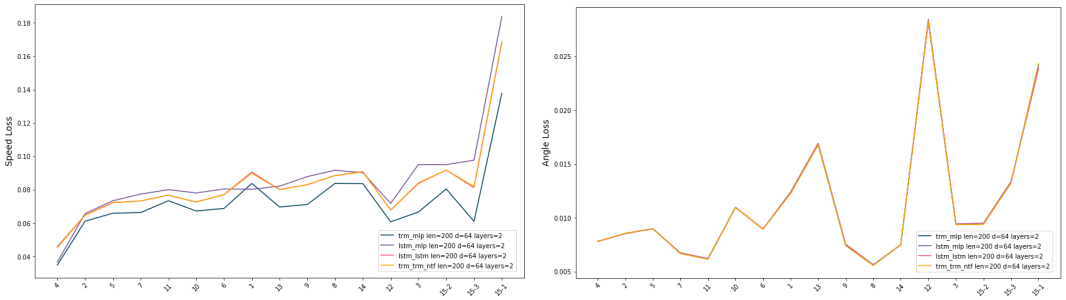
6.5.2 Analysis on Each Individual Driver

As we continue our research into driver identification and lane positioning tasks, we are focused on thoroughly examining how different model structures perform with different drivers. In Figure 6.4, we have reorganized the driver IDs based on their performance according to the LSTM-MLP model. What we consistently notice across all the models is a clear pattern in predicting each specific driver's behavior. Notably, the predictions for Driver 4 are impressively accurate.

What makes this particularly interesting is that, unlike what we observed in the tasks of lane positioning and driver identification, the models show exceptional skill in predicting the behavior of driver 4 across all the tasks. This unexpected and consistent performance for driver 4 is a significant and noteworthy finding.

By breaking down the different parts of the loss, we've created a visual representation of the losses in speed and angle for each driver, as shown in Figure 6.5. It's worth noting that the trends we see in speed loss are quite similar to those in the overall loss, mainly because speed-related factors play a dominant role.

When we focus on angle loss, a striking observation emerges: the model struggles to predict angles accurately for driver 12. This aligns with our earlier findings that the models faced challenges in correctly identifying driver 12. The difficulty in predicting angles for this driver is likely due to the complex and unique driving behaviors exhibited by them. This connection between predictive difficulty and identification accuracy highlights the



(a) Speed Loss (b) Angle Loss

Figure 6.5: Speed and Angle loss for each individual driver.

intricate nature of the specific driving behaviors that vary from one driver to another.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we compare LSTMs and Transformers with multi-modal time-series driving data on a variety of tasks, e.g., classification and regression tasks. For the classification task, driver identification, LSTMs outperform Transformers on both overall accuracy and multi-way top-k accuracy. For the regression tasks, a complicated decoder such as an LSTM or a Transformer is needed for the lane positioning task but is not necessary for predicting speed and steering wheel angle. It's worth mentioning that teacher forcing is harmful to these two regression tasks as errors are accumulated during inference. As for the depth and width of the model, we should carefully pick the proper depth and width if we choose to fit the classification task for both LSTMs and Transformers while it's not quite important to do so in regression tasks.

For driver identification, we train models on all data and test on highway data only. Not surprisingly, models are not able to identify drivers on highways well as most drivers share a similar driving behaviour when driving on highways and very few actions happen then. For better standardization, we should have a balanced dataset. More data on one particular person helps learn the behaviours of that person but can be harmful to the model to learn from others.

For regression tasks of driving data, models that consecutively predict future time steps such as Transformer decoders or LSTM decoders have a higher error in predicting further future time steps on both lane positioning and lane-keeping tasks. However, it's different when using models that predict future time steps simultaneously, e.g., an MLP decoder.

7.2 Future Work

In this thesis, we investigate three different driving behaviours, driver identification, lane positioning, and lane keeping. However, these three are not independent of each other. For example, the driving style vector learned from driver identification can be used for the other two downstream tasks for a personal driving system. As the input of three tasks is in the same format, a multi-task learning framework can be applied, that is, using the same encoder and changing the decoders for different outputs. Driving behaviour learning aims to mimic human behaviours and thus it's reasonable to think about how imitation learning can be utilized for it.

References

[1] Mozhgan Nasr Azadani and Azzedine Boukerche. Siamese Temporal Convolutional Networks for Driver Identification Using Driver Steering Behavior Analysis. IEEE Transactions on Intelligent Transportation Systems, 23(10):18076–18087, 2022.

[2] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. Advances in Neural Information Processing Systems, 28, 2015.

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to End Learning for Self-Driving Cars. arXiv preprint arXiv:1604.07316, 2016.

[4] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car. arXiv preprint arXiv:1704.07911, 2017.

[5] Zhilu Chen and Xinming Huang. End-to-end Learning for Lane Keeping of Self-driving Cars. In 2017 IEEE Intelligent Vehicles Symposium (IV), pages 1856–1860. IEEE, 2017.

[6] Xinping Gu, Yunpeng Han, and Junfu Yu. A Novel Lane-Changing Decision Model for Autonomous Vehicles Based on Deep Autoencoder Network and XGBoost. IEEE Access, 8:9846–9863, 2020.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. Neural Computation, 9(8):1735–1780, Nov 1997.

[8] Christopher Innocenti, Henrik Lindén, Ghazaleh Panahandeh, Lennart Svensson, and Nasser Mohammadiha. Imitation Learning for Vision-based Lane Keeping Assistance. In 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), pages 425–430. IEEE, 2017.

[9] Basma Khelfa and Antoine Tordeux. Lane-Changing Prediction in Highway: Comparing Empirically Rule-based Model MOBIL and a Naïve Bayes Algorithm. In 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), pages 1598–1603. IEEE, 2021.

[10] Jinkyu Kim and John Canny. Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention. In Proceedings of the IEEE International Conference on Computer Vision, pages 2942–2950, 2017.

[11] Minghao Liu, Shengqi Ren, Siyuan Ma, Jiahui Jiao, Yizhou Chen, Zhiguang Wang, and Wei Song. Gated Transformer Networks for Multivariate Time Series Classification. arXiv preprint arXiv:2103.14438, 2021.

[12] Qi Liu, Xueyuan Li, Shihua Yuan, and Zirui Li. Decision-Making Technology for Autonomous Vehicles: Learning-Based Methods, Applications and Future Outlook. In 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), pages 30–37. IEEE, 2021.

[13] Yonggang Liu, Xiao Wang, Liang Li, Shuo Cheng, and Zheng Chen. A Novel Lane Change Decision-Making Model of Autonomous Vehicle Based on Support Vector Machine. IEEE Access, 7:26543–26550, 2019.

[14] Keisuke Mori, Hiroshi Fukui, Takuya Murase, Tsubasa Hirakawa, Takayoshi Yamashita, and Hironobu Fujiyoshi. Visual Explanation by Attention Branch Network for End-to-end Learning-based Self-driving. In 2019 IEEE Intelligent Vehicles Symposium (IV), pages 1577–1582. IEEE, 2019.

[15] Society of Automotive Engineers (SAE). Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles, 2018.

[16] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles. IEEE Transactions on Intelligent Vehicles, 1(1):33–55, 2016.

[17] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-Modal Fusion Transformer for End-to-End Autonomous Driving. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 7077–7087, 2021.

[18] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and Decision-Making for Autonomous Vehicles. Annual Review of Control, Robotics, and Autonomous Systems, 1(1):187–210, 2018.

[19] Santokh Singh. Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey. Technical report, 2015.

[20] Charlott Vallon, Ziya Ercan, Ashwin Carvalho, and Francesco Borrelli. A Machine Learning Approach for Personalized Autonomous Lane Change Initiation and Control. In 2017 IEEE Intelligent Vehicles Symposium (IV), pages 1590–1595. IEEE, 2017.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. Advances in Neural Information Processing Systems, 30, 2017.

[22] Xiao Wang, Jinqiang Wu, Yanlei Gu, Hongbin Sun, Linhai Xu, Shunsuke Kamijo, and Nanning Zheng. Human-Like Maneuver Decision Using LSTM-CRF Model for On-Road Self-Driving. In 2018 21st International Conference on Intelligent Transportation Systems (ITSC), pages 210–216. IEEE, 2018.

[23] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end Learning of Driving Models from Large-scale Video Datasets. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2174–2182, 2017.

[24] Jingbo Yang, Ruge Zhao, Meixian Zhu, David Hallac, Jaka Sodnik, and Jure Leskovec. Driver2vec: Driver Identification from Automotive Data. arXiv preprint arXiv:2102.05234, 2021.

[25] Jinsoo Yang, Seongjin Lee, Wontaek Lim, and Myoungho Sunwoo. Human-like Decision-Making System for Overtaking Stationary Vehicles Based on Traffic Scene Interpretation. Sensors, 21(20):6768, 2021.

[26] Zhengyuan Yang, Yixuan Zhang, Jerry Yu, Junjie Cai, and Jiebo Luo. End-to-end Multi-Modal Multi-Task Vehicle Control for Self-Driving Cars with Visual Perception. In 2018 24th International Conference on Pattern Recognition (ICPR), pages 2289–2294. IEEE, 2018.

[27] Qi Zeng, Guangqiang Wu, and Libo Mao. A Support Vector Machine-Based Truck Discretionary Lane Changing Decision Model. In 2021 20th International Conference on Advanced Robotics (ICAR), pages 435–440. IEEE, 2021.

[28] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-to-end Interpretable Neural Motion Planner. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8660–8669, 2019.

[29] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pages 11106–11115, 2021.

APPENDICES

Appendix A

Driver Identification

n	Subset number	top-k
2	–	top-1
3	50	top-1, top-2
5	200	top-1, top-2, top-3
15	full set	top-1, top-2, top-3, top-5

Table A.1: Set up for n-way top-k accuracy.