A TWO-STAGE HYBRID DEEP LEARNING FRAMEWORK WITH REINFORCE-LEARNED TEMPORAL DILATED CONVOLUTIONS FOR PREDICTING VEHICLE LEFT-TURN SPEED AT PEDESTRIAN CROSSINGS

A Thesis Submitted to the Committee on Graduate Studies
in Partial Fulfillment of the Requirements for the Degree of Master of Science in the
Faculty of Arts and Science

TRENT UNIVERSITY

Peterborough, Ontario, Canada

Applied Modeling and Quantitative Methods M.Sc. Graduate Program

September 2025

Abstract

Title: A Two-Stage Hybrid Deep Learning Framework with Reinforce-Learned Temporal Dilated Convolutions for Predicting Vehicle Left-Turn Speed at Pedestrian Crossings

Author: Hamza Attarwala

Predicting vehicle speed at critical road segments, such as pedestrian crossings during left-turn maneuvers at signalized intersections, is essential for improving traffic safety and supporting autonomous driving systems. This thesis presents a novel two-stage hybrid deep learning framework enhanced with reinforcement learning to forecast vehicle left-turn speed at pedestrian crossings.

Using a multivariate time series dataset of vehicle speed and acceleration, the final three seconds of data are intentionally removed to simulate real-world decision-making prior to reaching pedestrian crossings. In stage one, a Convolutional Neural Network (CNN) imputes the removed values. Stage two uses the imputed data to forecast speed, combining Temporal Convolutional Networks (TCNs) and Long Short-Term Memory (LSTM) networks as feature extractors, followed by a Random Forest Regressor (RFR) for robust speed predictions.

Reinforcement learning is employed to dynamically adjusts the TCN's dilation rate, improving temporal pattern capture. Experimental results show the proposed framework outperforms standalone, hybrid, and state-of-the-art models.

Keywords: Left-Turn Maneuvers, Time series Forecasting, Temporal Convolutional Network, Long Short-Term Memory, Random Forest, Data Imputation, Reinforcement-Learning, Dynamic Dilation

Acknowledgement

I would like to express my deepest gratitude to all those who have supported me throughout the journey of completing this thesis.

First and foremost, I am profoundly grateful to my supervisor, Dr. Quazi Rahman, for his invaluable guidance, insightful feedback, and unwavering support. His expertise and encouragement have been instrumental in shaping this work. I would also like to extend my thanks to Dr. Mostafa Tawfeek, whose expert knowledge in the Transportation domain significantly enhanced the quality of this work. His contributions were pivotal in navigating complex aspects of the research.

I also gratefully acknowledge Digital Research Alliance of Canada for providing the computational resources that enabled the successful execution of the experiments conducted in this study. Their infrastructure was essential to the completion of this work.

Next, I would like to extend my special thanks to my family and friends, whose love, patience, and encouragement have been my greatest source of strength. Their belief in me has been a constant motivation.

Lastly, I would like to acknowledge the financial support provided by Trent University, which made this research possible.

Thank you all for your contributions and support.

Table of contents

List of Tables

List of Figures

1 Introduction

Pedestrian safety has been a central focus for many governments and organizations worldwide. The Insurance Institute of Highway Safety (IIHS) reported that pedestrian fatalities accounted for 18% of all crash-related deaths in 2024 [1]. Between 2018 and 2020, Statistics Canada indicated that approximately 21% of pedestrian fatalities occurred at intersections [2].

Given the rise in pedestrian accidents and fatalities, it is essential to implement standardized speed and safety measures. According to several studies, the probability of pedestrian fatality increases with higher vehicle speeds [3, 4]. Consequently, excessive vehicle speeds significantly raise the risk of vehicle-pedestrian collisions [5] and increases overall pedestrian crash risk [6].

To mitigate these risks, previous research has introduced the concept of *Target* speed, whose adaption has led to notable reductions in pedestrian and bicycle crashes in both urban and suburban areas [7]. Additionally, numerous studies have proposed various speed models and trajectory planning approaches to enhance traffic and pedestrian safety [8].

With the introduction of Connected and Autonomous Vehicles (CAVs), drivers of such vehicles no longer need to manually control the speed and maneuvering of their vehicles. Instead, CAVs determine appropriate speeds based on driver behaviors, thereby improving comfort and enhancing trust in system reliability [9–11]. However, intersections remain complex environments that require advanced decision-making to ensure safe navigation, even with CAVs. Several studies have focused on modelling left-turn maneuvers at signalized intersections by calibrating kinetic models, planning left-turn trajectories, and analyzing path dispersion of left-turning and opposing through movements [15].

Using basic speed and acceleration data of the vehicles at an intersection and employing a combination of hybrid deep learning and ensemble models, this dissertation attempts to predict the left turning speed of vehicles at pedestrian crossings to provide an advisory/target speed using a proposed two-stage framework. This advisory/target speed can be relied to the driver in a semi-AV environment, where the vehicle gives the driver a warning about the speed or suggest a specific speed. The novel two-stage framework, that includes an imputation and forecasting model, attempts to forecast the speed of the vehicles using real-world intersection data. Furthermore, a reinforcement learning agent is then used to improve the ability of the forecasting model to forecast the target speed of vehicles over the pedestrian crossings with the highest accuracy.

A time series refers to a sequence of data points collected at successive, equally spaced intervals over time. Since vehicle speed and acceleration at intersections are recorded at regular time intervals, predicting vehicle speed can be formulated as a time series forecasting problem. This chapter begins with a brief introduction and background on time series forecasting, followed by a section outlining the organization of the thesis.

1.1 Time Series Forecasting

In order to understand time series forecasting, it is essential to understand time series data. Time series data is the collection of observations over time. Intuitively, time series data can be interpreted as sequence of data points that are taken at equal intervals of time. Example of time series data can include records of daily stock prices, weekly sales data, hourly amount of precipitation etc.

Forecasting on the other hand is any statement made about the future. The term forecasting has been largely associated with certain situation such as predicting weather, stock prices, or number of goals that may be scored by a soccer player in a year. Time series forecasting is the process of analyzing time series data using statistics and modelling to make an informed and strategic prediction. Forecasting a time series can be dated back to 800BC, however, it was only formally disciplined in 1927, when a British statistician, Udny Yule, used it for their analysis of sunspots [16].

Unlike independent and identically distributed (iid) data, where each observation is assumed to be independent and future values are not influenced by past ones, time series data typically exhibit auto-correlation. This means that future values are often related to their past observations. Consequently, effective time series forecasting requires methods that can identify and interpret the patterns formed by these auto-correlations before generating predictions. The patterns within a time series can generally be categorized as seasonal short-term patterns, cyclical short-term movements, and long-term trends. Seasonal short-term patterns refer to periodic fluctuations that occur at fixed intervals within a year. Cyclical patterns, while similar to seasonal ones, may not occur at strictly regular intervals and can span longer periods. Long-term patterns describe the overall progression of the time series over an extended period. These trends may be linear or non-linear and can indicate an increasing, decreasing, or stable trajectory.

For a forecasting model to accurately predict future values in a time series, it should be capable of learning from historical data in order to:

1. Identify the meaningful regularities and patterns, such as lagged relationships, present in the past data.
2. Ensure that the identified patterns provide relevant information for making accurate forecasts.
3. Distinguish between meaningful patterns and noise, ensuring that irrelevant fluctuations or noisy data do not distort the forecast.

To ensure that the forecast in this thesis follows the above stated conditions, not only well established statistical models but also traditional machine learning, deep learning and hybrid models that are widely adopted in literature are utilized. These are further discussed in detail in Chapter 3.

1.2 Thesis Organization

Chapter 2 presents a literature review of recent works related to target speed and trajectory prediction, specifically for vehicles turning left at signalized intersections. It also discusses studies in the field of time series forecasting, as well as approaches to dynamically selecting optimal dilation rates in Temporal Convolutional Networks (TCN), which form part of the forecasting model in the proposed two-stage framework.

Next, Chapter 3 explains the various forecasting methods used in this thesis. It first describes in detail the real-world dataset employed in this study, and the preprocessing steps taken to create an ideal real-world scenario, by removing certain number of timestamps, for forecasting speed over pedestrian crossings. Next, the different forecasting models used to predict the target speed of vehicles over pedestrian crossings are then described. Following this,the statistical and traditional machine learning models employed for forecasting-such as the Random Forest Regressor (RFR)-are explained in detail.

Chapter 3 then continues by presenting the deep learning models utilized to forecast the time series, namely Convolutional Neural Networks (CNN) and Long Short-term Memory (LSTM). It then describes in detail the hybrid models used for forecasting, which are created by combining various deep learning methods. Before introducing the proposed forecasting model, an extension of CNN, the Temporal Convolutional Network (TCN), is discussed in detail. This is followed by the introduction of the proposed hybrid deep learning TCN-LSTM-RFR model. The chapter concludes by presenting the results of the forecasting models and methods discussed.

Chapter 4 builds upon the proposed hybrid TCN-LSTM-RFR model described in Chapter 3 and proposes a novel two-stage framework that uses this hybrid forecasting model after imputing the removed speed and acceleration data from the time series. It begins by describing the various imputation methods used to handle the removed data, including classic imputation methods, attention-based imputation models, state-of-the-art (SOTA) approaches, and deep learning models configured for imputation. The CNN imputation model used in the two-stage framework is also described in detail. Next, the novel two-stage framework that integrates the CNN imputation model and the TCN-LSTM-RFR forecasting model is then presented in detail.

Chapter 4 also discusses the reinforcement learning methods used for selecting the optimal dilation rate in the TCN model. It first introduces the fundamentals of reinforcement learning, followed by a detailed explanation of the various components of the reinforcement learning agent. The specific reinforcement learning based architecture with dynamic dilation used in this thesis is then described. Finally, the chapter concludes with a summary of all the components discussed.

Lastly, Chapter 5 presents an overall review of thesis and highlights its major contributions. It also discusses potential future work that could be undertaken to further advance the research presented in this thesis.

2 Related Work

This chapter presents previous research that has been done related to this thesis, which includes target speed prediction for left-turning vehicles at signalized intersections, the various forecasting methods that have been used for time series, and the selection of dynamic dilation rates of Temporal Convolutional layers. Additionally, the concept and background of target speed is also introduced.

Target speed for vehicles is defined as *the highest operating speed at which vehicles should ideally operate on a roadway in a specific context, consistent with the level of multimodal activity generated by adjacent land uses, to provide both mobility for motor vehicles and a safe environment for pedestrian, bicyclists, and public transit users* [7]. This not only emphasizes the physical capabilities of the vehicles or the design of the road, but also the interaction between vehicles and the surrounding environment, ensuring the safety and comfort of all users.

Predicting the *Target Speed* for vehicles and offering it as an advisory speed to drivers of autonomous vehicles (AV) can enhance driver confidence in the vehicle and clarify the maneuvers executed by the AV. However this posses some challenges as mentioned below:

1. Each driver exhibit different behaviours while driving, and are influenced by their preferences, experience level, and situation awareness. [17]
2. Speed predictions involve analyzing factors such as vehicular behaviour, lane-level dynamics, and overall traffic flow. These are difficult to model and challenging due to variability in data, and the modelling techniques needed for each level. [18]
3. Traditional methods to predict speed often reply on infrastructure such as cameras and sensors. However, these equipment may not provide comprehensive coverage of traffic flow dynamic, and hence hinder the accuracy of speed prediction. [19]

These issues become even more significant when speed prediction is performed at signalized intersections. Additional challenges arise due to factors such as variability in traffic volume patterns [20], variability in the delays vehicles experience at signalized intersections [21], and data imperfections such as missing data or noise [18]. Consequently, developing accurate and reliable speed prediction models in such environments requires robust approaches that can account for these complexities and ensure both operational efficiency and safety.

To overcome these challenges, this thesis focuses on using basic vehicle data, specifically speed and acceleration values, to provide drivers with an advisory or target speed for left-turning vehicles at signalized intersection over the pedestrian crossing. This thesis addresses the key challenge of predicting the target speed of vehicles making a left-turn maneuver at pedestrian crossings. It introduces a two-stage framework in which the optimal dilation rate of the Temporal Convolutional Network (TCN) model, a component of the proposed architecture, is selected using reinforcement learning.

2.1 Left-turn Target Speed Prediction

To estimate naturalistic driving at intersections, several studies developed speed control and modelling tools using real-world trajectories and naturalistic driving data that simulate real-world driving, especially for left-turning vehicles at intersections [12, 22]. The key objective of these speed control models were to replicate the natural deceleration and acceleration patterns of the vehicles as they approach, and navigate through the intersection. The models would identify acceleration inputs and optimize stopping and turning at intersections, providing insights to driver behaviour which can improve safety systems. Extending this study to improve the accuracy of target speed predictions by the speed control model, Wolfermann et al.[23] proposed a model for generating stochastic speed profiles for left and right-turning vehicles at intersections. Through a generalized analysis, the authors were successful in developing a model that formed a basis for understanding speed behaviours. This study highlighted the importance of accurately predicting vehicle speeds and acceleration during turning, which contributed to safer navigation of vehicles through intersections.

Furthermore, a recent study [24] proposed a high-dimensional lane-level travel speed feature extraction and mapping, that aimed to provide a short-term travel speed prediction on urban expressways. They utilized 3D convolutions and attention modules that acted as enhanced feature extractors for traffic dynamics. Using this method, the authors were able to successfully overcome the uncertainty and volatility of traffic flows at the lane level. Additionally, Polverino et al.[25] developed a vehicle speed management algorithm, that suggested drivers of vehicles a target speed in real-time to achieve lower fuel consumption. Complementing this study, Nesa and Yoon [26] attempted to explain and interpret results of different AI-based speed prediction model using various regression-based models such as XGBoost, Random Forest, and LASSO.

In additions to speed control and speed profile models, Sander [27] conducted a study to evaluate an Automated Emergency Braking (AEB) system that is specifically designed to avoid collisions at intersection while vehicles make a left-turn. The AEB system was tested in a virtual simulation, where it was noted that the system was able to identify the critical speed threshold of the vehicles in the simulation where the collision risk escalated. Alongside the AEB system, a multi-vehicle decision making system designed for left-turn behaviour in autonomous vehicle was also proposed [28]. This system integrated conflict resolution methods, speed control strategies, and decision-making algorithms to guide autonomous vehicles through intersections safely. Collision avoidance system such as these, along side speed control system, play a crucial role in predicting and managing vehicle speeds at intersections, especially while making turns, due to the increased risk of vehicle-vehicle and vehicle-pedestrian collisions. These collision avoidance system also assist in maintaining a target speed for vehicles, that helps in reducing pedestrian and bike crashes [7].

With recent advancements in intelligent vehicles, it is necessary that trajectory planning in addition to speed control and collision avoidance system are also developed. These trajectory planning models are required in intelligent vehicles as they are used particularly to perform complex maneuvers such as left-turns at intersections while keeping safety at its at most priority. For this purpose, Long et al. [14] proposed a trajectory planning algorithm that adapts based on constrains such as velocity, acceleration and road boundary, to enhance safety, energy saving and comfort. The algorithm developed addressed multiple constrains, including vehicle left-turn, longitudinal, and lateral velocity. Moreover, a comprehensive framework for predicting driver intentions at intersection was developed using real-world data and advanced data mining strategies [29]. The model used key vehicles features such as location, velocity, and intersection layout, encoded through high-resolution maps. This model was able to demonstrate the effectiveness of data-driven models in forecasting vehicle behaviour, and target speeds at pedestrian crossings at intersections.

Complementing this, with recent advancements in image interpretations and generations system using machine learning and deep learning techniques, several vision-based approaches have been developed. Shirazi and Morris[30] introduced a vision-based vehicle tracking system that aims to enhance the analysis of vehicle behavior at intersections, particularly during turning maneuvers. The images were obtained using a single camera setup that used motion based tracking, and were coupled with trajectories to analyse the vehicle speed and turning movements accurately. Additionally, a homogeneous method to estimate real-world 3D coordinates, which were then used to model speed patterns and categorize vehicles as "Stop," "Slow," "Normal," or "Speeding" was also utilized. In another study by Abdelhalim et al.[31] a three-step system framework was proposed that was used for real-time estimations and vehicle trajectory predictions that used vehicle speed estimates that were obtained using video inference.

2.2 Forecasting Methods for Time Series

Time series forecasting has been extensively studied across various domains and can generally be classified into three main categories: classical forecasting methods, machine and deep learning prediction methods, and hybrid time series forecasting methods [32]. Before discussing these approaches, it is essential to understand the different types and characteristics of time series data.

A time series can be either univariate or multivariate. A univariate time series consists of a single variable measured over time, for example, recording the daily temperature of a city over one year. In contrast, a multivariate time series involves the simultaneous measurement of two or more variables over the same time interval. An example of this could be recording temperature, wind speed, and humidity in a city throughout a year.

Furthermore, time series data may exhibit the property of stationarity, which refers to a time series whose statistical properties, such as mean and variance, remain constant over time and do not depend on the time at which the time series is observed. However, this may not be the case in every time series as some time series also exhibit non-stationarity, i.e., the time series do not have constant statistical properties, and exhibit varying means and variance.

Classical forecasting methods rely on mathematical and statistical models, which are often dependent on the properties of the time series. One of the earliest classical forecasting model is the Autoregressive (AR) model, which was introduced to describe time-dependent processes. An AR model captures the relationship between a time series and its past (lagged) observations by expressing the current value as a linear combination of previous values.

Building upon this, the Moving Average (MA) model was developed for analyzing and forecasting univariate time series. Unlike the AR model, which relies on past values, the MA model utilizes past forecast errors to make predictions.

To further improve forecasting performance, the AR and MA models were combined to form the Autoregressive Integrated Moving Average (ARIMA) model. In addition to incorporating both AR and MA components, ARIMA includes a differencing step, which transforms a non-stationary time series into a stationary one by computing the differences between consecutive observations. The ARIMA model has been widely used for forecasting time series, and has been used in field such as economics, health-care, and weather [33–35].

These classical forecasting models can capture linear relationships, and produce good results when used over smaller dataset. However, they do not perform well over large, and complex dataset that do not follow stationarity [32]. Hence, to overcome these limitations of classical forecasting models, various machine (ML) and deep learning (DL) methods are also used to forecast time series data. These ML and DL methods are great at capturing non-linear relationship between different variables and features in relation to time within the time series, and can use these captured relationships for forecasting. Machine learning models such as Support Vector Machines (SVM), Random Forest Regressors (RFR), and deep learning models such as Artificial Neural Networks (ANN), Recurrent Neural Network (RNN), specifically Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), Convolutional Neural Networks (CNN), and Temporal Convolutional Networks (TCN) are widely used for forecasting of time series. Recently, several transformer-based model have also been utilized to forecast time series and have been able to achieve result with superior performance [36].

Nevertheless, real-world time series often present several challenges: (1) it is not always straightforward to determine whether a time series is stationary or non-stationary, (2) most time series are neither purely stationary nor purely non-stationary but rather a mixture of both, and (3) in practice, a single model cannot effectively capture all patterns present in different time series simultaneously. Stationarity plays a critical role in time series modelling, as many traditional forecasting techniques such as ARIMA rely on the assumption that the underlying series is stationary. However, real-world data frequently violate this assumption due to the presence of varying trends and seasonality. These non-stationary components can obscure the true data-generating process and make it difficult for a single model to learn consistent and stable patterns. Consequently, single-model approaches may struggle to accurately estimate error confidence intervals, introducing bias in autocorrelation structures, and resulting in poor model fitting, and ultimately produce misleading forecasts. To address these limitations, it becomes necessary to adopt multiple models that can handle different structural aspects of the time series. In this context, hybrid models are employed, which combine classical statistical approaches with machine learning or deep learning methods, or integrate multiple machine or deep learning models. By leveraging hybrid modeling, it becomes possible to capture both stationary and non-stationary components more effectively, thereby improving forecasting accuracy and enhancing the model's generalization capabilities.

2.3 Dynamic Dilation in Temporal Convolutional Network

Temporal Convolutional Networks (TCNs) have been widely used for the task of time series forecasting by various studies previously [37, 38], and use the concept of dilation in their architecture (further discussed in Section 3.5.1). Some recent studies have explored methods to dynamically select or adapt dilation rates in the convolution layers of the TCN during training. Chu et al. [39] proposed a Multi-Kernel Temporal Dynamic Dilation Convolution combined with an Adaptive Decision Spatio-temporal neural Ordinal Differential Equation model. This approach, designed for traffic flow forecasting, enabled the network to learn features across diverse temporal scales by dynamically adjusting its receptive field. By adapting the dilation rates, the model was able to better capture temporal dependencies and respond to changing traffic data over time. Furthermore, combining dynamic dilation with an Ordinal Differential Equation improved forecasting performance by introducing greater flexibility and precision in modelling continuous-time dynamics, making it especially valuable for forecasting vehicle movements in intersection scenarios.

Additionally, to address constraints caused by shared dilation, which limits the ability to capture diverse spatial content at different locations, Yao et al. proposed Adaptive Dilation Convolutional Neural Networks (ADCNN) [40]. This method introduced a novel attention-based module to guide dilation selection at each spatial location, improving segmentation accuracy while maintaining computational efficiency comparable to standard dilated convolutions.

In another study, an extended version of TCN called Adaptive TCN (AdaTCN) was proposed [41] to improve flexibility and effectiveness in language modeling tasks. AdaTCN used several learnable sparse binary masks applied to one-dimensional convolution layers for sequence modeling. However, results showed that the baseline TCN outperformed AdaTCN and produced better overall results.

2.4 Summary

This chapter covered the essential topic of *Target Speed*, defined as "the highest operating speed at which vehicles should ideally operate on a roadway in a specific context" [7]. Adopting target speed would provide clarity and explainability to the drivers of autonomous vehicles (AV) on the actions taken by the vehicle. The chapter also indicated the issues with speed prediction and the complexity of speed prediction over pedestrian crossings.

Previous works have primarily focused on developing speed control and collision avoidance systems that predict speed. However, these models are often trained using free-flowing traffic data or data that contains limited driver demographics. Additionally, some studies rely on simulated traffic data, which lacks real-world validation and raises concerns about practical effectiveness. Furthermore, vision-based models that utilize images and videos, along with sensory data to predict vehicle speeds, tend to be computationally expensive and sensitive to tracking errors. Despite the increasing adoption of deep learning in time series forecasting, existing approaches have not extensively explored the use of hybrid deep learning architectures combined with dynamic dilation techniques for speed prediction. This gap limits the potential to effectively model complex temporal dependencies and adapt to varying traffic patterns, especially under diverse real-world driving conditions.

Hence, this thesis builds up on previous works, and proposes a framework that is able to predict the speed of vehicles over pedestrian crossings at signalized intersections. Chapter 3 will cover basic forecasting theory and forecasting methods. The primary focus of this chapter will be presenting the proposed two-stage framework that includes imputation of the removed speed and acceleration values, as well as forecasting using a Reinforced-learned dynamical dilated TCN-LSTM-RFR model, which will be explained in detail.

3 Hybrid Deep Learning Architecture for Forecasting

This chapter begins by describing the dataset used to forecast the speed of vehicles making left-turns at pedestrian crossings, followed by the preprocessing steps applied to simulate real-world scenarios by removing a portion of the speed and acceleration timestamps. Next, it introduces the evaluation metrics employed to assess, compare, and rank the performance of the forecasting models.

Following this, the traditional machine learning and deep learning methods used for comparison with the proposed hybrid model are explained. An overview of the Temporal Convolutional Network (TCN) is then provided, which serves as a foundation for the hybrid approach. The chapter then presents the hybrid TCN-LSTM-RFR model in detail, describing each of its three components, namely TCN, LSTM, and RFR, along with their mathematical formulations, before detailing the integration strategy that combines them into a unified predictive model.

Furthermore, this chapter discusses the forecasting results obtained using the hybrid TCN-LSTM-RFR model on data that excludes the last three seconds of vehicle speed and acceleration. Finally, the chapter concludes with a discussion summarizing the key findings and insights from the work presented in this chapter.

3.1 Data

While several open-source trajectory datasets exist, most have limitations for the specific problem of modelling target speed during left turns over pedestrian crossings. Drone-based datasets such as INTERACTION [42], inD/rounD [43], or pNEUMA [44] provide high-resolution trajectories of vehicles and pedestrians, but typically lack pedestrian signal phases, driver control inputs, and often contain relatively few unprotected left-turn–pedestrian conflict events. Similarly, autonomous vehicle sensor datasets such as Waymo Open Motion

[45–47], Argoverse [48], or nuScenes [49] offer rich multi-agent trajectories and HD maps, but again rarely capture the detailed pedestrian crossing signal states or vehicle dynamics at the pedestrian crossing itself. Pedestrian-focused datasets such as JAAD [50] or PIE [51] emphasize intent recognition rather than continuous traffic interactions, and therefore provide limited utility for turn-speed modelling. To overcome these limitations, this research utilizes the NGSIM dataset [52], which offers detailed vehicle trajectory data suitable for analyzing turn-speed behavior at intersections over the pedestrian crossing.

The NGSIM dataset [52] was obtained from the U.S. Department of Transportation - Federal Highway Administration. Four intersections of Lankershim Boulevard, Los Angeles, California were recorded using five synchronous video cameras in 2005. The cameras were placed on several high rise buildings surrounding the intersections and captured vehicles that entered and exited the intersections during the morning peak hours. The trajectories of the vehicles captured were transcribed using a software application. Utilizing the transcribed trajectories of the vehicles, speed and acceleration data of the vehicles were obtained. Additionally, the speed of the vehicles specifically over the pedestrian crossings were also identified and obtained.

Since the focus of this thesis is on predicting the speed of left-turning vehicles, only vehicles that entered and exited the intersection by making a left turn were considered for this study. All other vehicles that did not fall into this category were excluded. Upon examination, about 190 vehicles were identified that fit the criteria and were included in the study. For each vehicle, 10 timestamps of speed and acceleration data were simultaneously recorded. This data was collected from the moment the vehicles entered the intersection until they exited after completing the left-hand turn. The image of the four intersection that were recorded are shown in Figure 3.1.

It should be noted that not all vehicles tracked at the intersections have the same number of timestamps. This discrepancy is likely due to two main reasons:

1. Variation in time spent at the intersection - Some vehicles were able to make a left turn immediately upon entering the intersection due to a green advance left-turn signal,

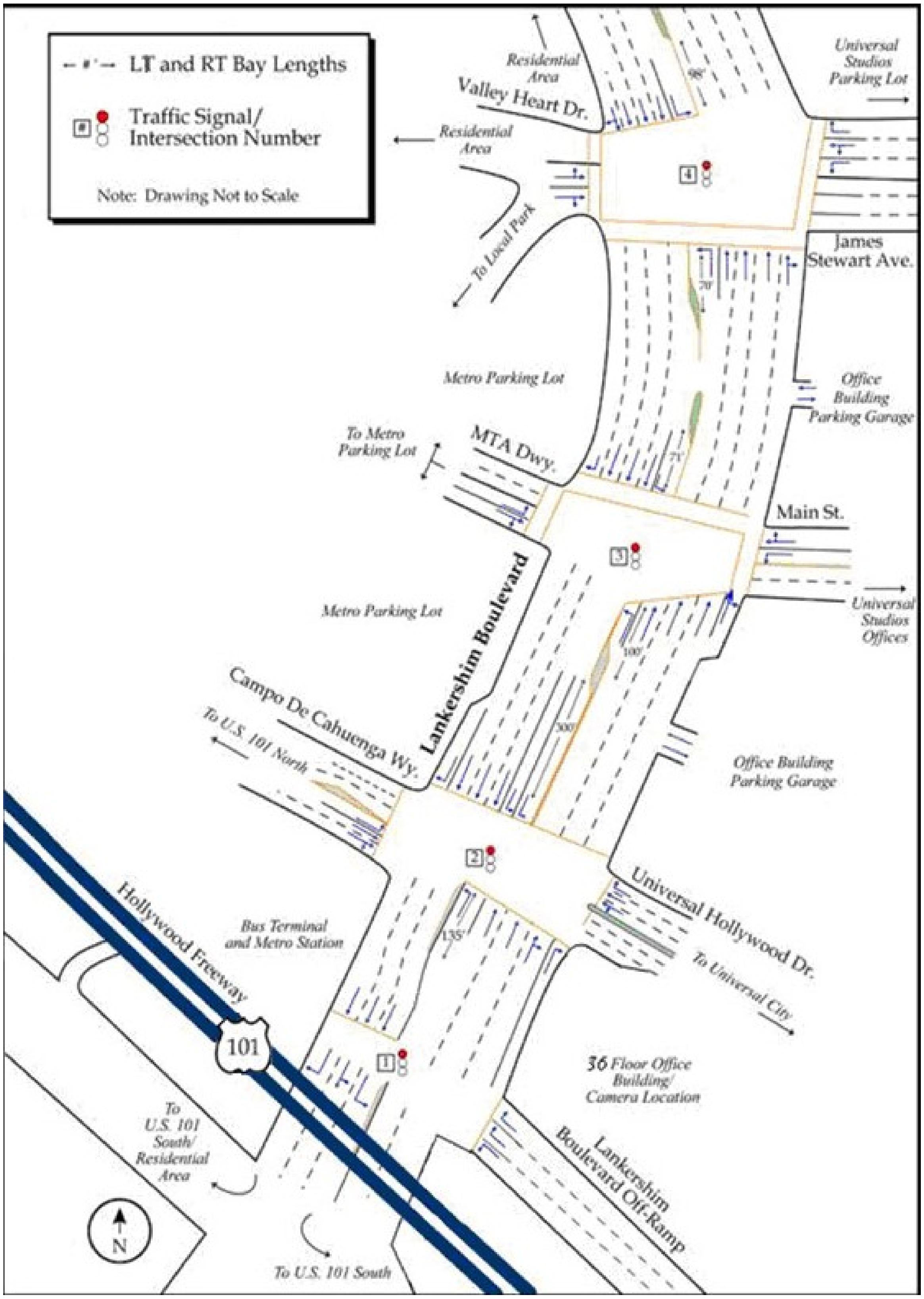

Figure 3.1: Image of the intersections that were recorded on Lankershim Boulevard. Vehicles going north on Lankershim Boulevard, that turned on Campo De Cahuenga Wy, MTA Dwy, or Valley Heart Dr, and vehicles going shouth that turned on Lankershim Boulevard Off-Ramp, Universal Hollywood Dr, Main St, or James Stewart Ave, were used

while others had to wait until the intersection was clear.

2. Asynchronous entry and exit - Vehicles did not enter or leave the intersection at the same time, leading to differences in the number of recorded timestamps

Due to the above reasons, it was essential to pre-process and standardize the number of timestamps for the vehicles in the dataset. Some vehicles had timestamps as little as 50, whereas other vehicles had over 1600 timestamps. In order to standardize, a cutoff of 200 timestamps was selected, where, any vehicles that had less than 200 timestamps were not taken into account for the study.

Despite implementing the aforementioned measures, another issue persisted. Specifically, the vehicles remaining in the study lacked a uniform timestamp that recorded their speed over the pedestrian crossings. For instance, vehicle x entered the intersection and reached the pedestrian crossing at timestamp t. However, when another vehicle y entered the intersection, it spent more time inside the intersection compared to vehicle x, and hence ended up reaching the pedestrian crossing at timestamp $t + 10$. To overcome such scenarios, it was essential to preprocess the data to standardize the timestamps of all the vehicles in the dataset. This alignment ensured that the speed and acceleration timestamps for all vehicles were consistent, such that the timestamp at which they reached the pedestrian crossings remained the same.

To align the timestamps, the vehicle that reached the pedestrian crossing first, after entering the intersection, was identified. The timestamp at which this vehicle reached the pedestrian crossing was used as a temporal reference. Subsequently, the timestamps of all other vehicles in the dataset were adjusted relative to this temporal reference. The preprocessing ensured that the final timestamp value for all vehicles in the dataset corresponded to their speed at the pedestrian crossings.

The main purpose of the proposed model architecture and framework is to be used in autonomous vehicles (AV) to predict a typical safe vehicle speed at pedestrian crossings. This speed would then be relayed to the drivers as an advisory speed. To ensure that drivers have enough time to adjust their speeds or take necessary actions, the advisory speed should be provided to the driver of the vehicle a few seconds before the vehicle reaches the pedestrian crossing. Therefore, to train and test the model, the last 3 seconds (30 timestamps) of speed and acceleration data were removed for all vehicles in the dataset.

3.2 Evaluation Metrics

The result of forecast are highly dependent on the context in which they are made. These context are often unpredictable, and hence to measure the quality of a forecast, a forecasting metric is required. The forecasting metric chosen for this thesis is Mean Absolute Percentage Error (MAPE). MAPE is defined in Equation 3.1

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right| \times 100 \tag{3.1}$$

where, A_t are the actual values, F_t are the forecast values, and n is the number of steps in the forecast.

MAPE is a measurement of error between the actual and the forecasts value, hence a MAPE score of 0% indicates that the forecast was perfect. Since MAPE expresses the error as a percentage, a lower MAPE value indicates that the accuracy of the forecast is high, and on the contrary, a higher MAPE value suggest that the accuracy of the forecast is low.

Although MAPE is a standard forecasting metric, it does have its limitations. Firstly, the MAPE value is undefined when actual values are 0, as this leads to a division by zero error. However, this limitation is not a concern for this thesis, as none of the actual values, i.e., the speed of the vehicles over the pedestrian crossings, are 0. Secondly, MAPE might not be the best metric for comparing the forecasting accuracy across datasets that have different scales or unit. This would have been a matter of concern for this thesis if the speed and acceleration dataset were being evaluated separately. Nevertheless, because the speed and acceleration data are used in combination for this study, the limitation concerning different scales or unit also does not effect the results and their evaluations.

3.3 Forecasting using Traditional Machine Learning Methods

Prior to the introduction of machine learning models, statistical models such as Linear Regression (LR), ARIMA or Seasonal ARIMA were widely popular for the task for time series forecasting. These model would attempt to forecast values of a time series using a linear relationship. However, in the late 1970s and early 1980s, it became increasingly clear that linear models were not adaptable for many real-world applications [53]. With recent advancements in Artificial Intelligence (AI), traditional machine learning models have been adopted by various studies for the purpose of forecasting time series data. They have been considered to be serious contenders to statistical models [54]. Some of these traditional machine learning models are namely Decision Tree (RT), Support Vector Machines (SVR), and Random Forest Regressor (RFR). These traditional machine learning models follow a basic concept of forecasting a time series of interest y based on its previous time series values x. Although they do not model temporal dependencies inherently, they can learn relationships based on the autocorrelation structure present in the lagged input features, and use these learned temporal patterns to predict future values.

By adopting machine learning techniques, the time series patterns are learned by approximating both linear and non-linear functions, offering robust performance in high-dimensional spaces. These are important for this thesis as the time series data is multivariate - i.e. it contains more than one variable that are tracked - and requires the ability to handle and learn intricacies from the temporal patterns.

3.3.1 Linear Models

A Linear regression model, in the simplest case, attempts to forecast a variable y by using a single predictor variable x. It attempts to find a linear function that fits the data as shown in Equation 3.2

$$y_t = \beta_0 + \beta_1 x_t + \epsilon_t \tag{3.2}$$

where y_t is the forecast value, β_0 and β_1 are the coefficients, and are denoted as the intercept, and the slope of the line, respectively, and ϵ_t is the error.

However, when the dataset includes 2 or more variables that the linear model needs to take into account, the linear regression model attempts to fit a model that follows the general form of

$$y_t = \beta_0 + \beta_1 x_{t1} + \beta_2 x_{t2} + \beta_3 x_{t3} + \cdots + \beta_k x_{tk} + \epsilon_t \tag{3.3}$$

where, all variables x_1 to x_k are used for forecasting the next timestamps in the time series.

An advantage of linear model compared to other models when used for time series forecasting is that they are highly interpretable and are often have a low computation cost associated to them. Auto-regressive integrated moving average (ARMIA) are widely used linear models for the purpose of forecasting using past observations. They consist of parameters p - number of time lags, d - the degree of differencing (difference of the present and past values), and q - the order of the moving averages. Additionally, extension of ARIMA called Seasonal ARIMA (SARIMA) have also been used extensively for time series forecasting, where the time series exhibits seasonality.

In this study, linear models were fitted using both speed and acceleration data. The linear model attempted to fit the data to forecast can be expressed as the following equation

$$y_t = \beta_0 + \beta_1 \cdot \text{speed} + \beta_2 \cdot \text{acceleration} \tag{3.4}$$

Nevertheless, for the linear model to perform well, and provide suitable forecast predictions, the data over which the model is fitted, should observe linearity, and should be stationary. More often than not, this is not the case for most time series, as there are several factors that effect these attributes of the time series, which is the case in this study as well. The linear models were not be suitable for this study, as the speed and accelerations of vehicles (as seen in Section 3.3.5) do not follow properties that are suitable of a linear model (non-linearity), which is attributed to the complex and ever changing environment of intersections.

3.3.2 Decision Tree

To address the challenges of non-linearity and irregularities in time series, several modern tree based models can overcome problems of forecasting non-linear and non-stationary time series data. Decision tree (DT), being one-such tree based models, can be used to forecast the speed of the vehicles over pedestrian crossings.

Decision tree have been primarily been used as a non-parametric supervised learning approach which are used for both classification and regression. As the name suggest, they form a tree structure, and break down the data into smaller sub-problems. The final tree that is constructed after breaking down the data contains a root node, an internal node, and leaf nodes. Since time series forecasting is a regression based problem, the inputs to the decision tree are the lagged time series values to predict future values. The decision tree is trained to use the input feature values $X_t = [y_{t-1}, y_{t-2}, \dots, y_{t-k}]$ to produce an output y_t.

Once trained, the decision tree model fits the data and forecasts the future value y_t using the following equation.

$$y_t = f(X_t) = \sum_{m=1}^{M} c_m \cdot \mathbb{I}(X_t \in R_m) \tag{3.5}$$

where, the decision tree partitions the input space into M leaf regions $(R_1, R_2, \dots, R_M)$ based on how the feature are split. $X_t \in \mathbb{R}^p$ is the feature vector of the past values, c_m is usually the mean of y_t in the region $\mathbb{R}_m$, and $\mathbb{I}(X_t \in R_m)$ being the indicator function which would be equal to 1 if X_t falls in region R_m, or otherwise 0.

For this thesis, as with the linear models, the decision tree was provided both speed and acceleration values X_t of the vehicles to predict their speed over pedestrian crossings

y_t. The ability of the decision tree to interpret non-linear and non-stationary data was the motivation of using it for the purpose of forecasting the speeds of the vehicles. Furthermore, an extension of a Decision tree, called Random Forest Regressor (RFR), was also utilized in this thesis to forecast the time series (See Section 3.3.4).

3.3.3 Support Vector Machines

Support Vector Machines (SVMs) are powerful and versatile machine learning tools that have been successfully applied to various prediction tasks, including time series forecasting. Originally developed primarily for classification, SVMs have been extended to regression problems through Support Vector Regression (SVR), enabling them to predict continuous outcomes [55]. In the context of time series forecasting, SVR is trained on transformed data where lagged values of the time series are used as input features to predict future observations [56].

A key advantage of SVMs lies in their ability to capture non-linear relationships using kernel functions, which implicitly maps the input data into higher-dimensional feature spaces. This capability allows the model to approximate complex, non-linear functions that may be present in real-world time series data. When combined with suitable preprocessing and feature selection, SVMs have been shown to perform competitively for time series forecasting, particularly in cases involving non-linear and multivariate dynamics [56].

Support Vector Regressors (SVR) for time series forecasting follow a fundamental equation derived from the Support Vector Machine theory, which is

$$y_t = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(\mathbf{x}_i, \mathbf{x}_t) + b \tag{3.6}$$

where, $\mathbf{x}_t$ is the input vector of the lagged values, $\mathbf{x}_i$ are the support vectors, (α_i, α_i^*) are the dual coefficients that are obtained while training, K is the kernel function, b is the bias term, and n is the number of support vectors.

In terms of using SVRs for speed forecasting using lagged or past speed and values, the input vector is constructed using k lagged values, i.e. $x_t = [\text{speed}_{t-k}, \dots, \text{speed}_{t-1}, \text{accel}_{t-k}, \dots, \text{accel}_{t-1}]$.

3.3.4 Random Forest Regressor

When applied to time series forecasting, decision trees frequently suffer from overfitting, resulting in poor predictive performance. However, an extension to the decision tree, called Random Forest (RF), offers a robust approach to forecasting a time series. A Random Forest model is a ensemble machine learning model, that is widely used for various classification and regression tasks. It is created by aggregating the prediction of multiple decision trees, which prevents the model from overfitting the data, and enhances it generalization ability [57].

A Random Forest, as with Decisions trees and Support Vector Machines, can be used for time series forecasting after the time series to be converted into a supervised learning problem. Additionally, it also is able to map complex relationship between features in datasets that have high dimensionality.

Unlike, statistical and linear models, Random Forest do not have a defined explicit function, and rather use the lagged values of the time series to forecast the future values. This can be expressed as

$$y_t = \frac{1}{M} \sum_{m=1}^{M} T_m(\mathbf{x}_t) \tag{3.7}$$

where, M is the number of tree in the forest, and $T_m(\cdot)$ is the prediction if the m-th decision tree of the forest.

In this study, both lagged speed and acceleration values were simultaneously used as inputs for the Random Forest Regressor to forecast the time series. Owing to its ensemble approach, i.e., combining predictions from multiple decision trees, the Random Forest is often capable of achieving higher forecasting accuracy [58], which was also observed in the results of this study as shown in Section 3.3.5.

3.3.5 Results using Traditional Machine Learning Methods

Following the experiment using traditional machine learning models to predict vehicle speeds during left turns at signalized pedestrian crossings, the resulting data is presented in Table 3.1.

Table 3.1: MAPE values using Traditional Machine Learning Models

Models	**Forecasting MAPE values**
Linear Regression (LR)	28.44
Decision Tree (DT)	19.64
ARIMA	17.68
SARIMA	17.63
Random Forest (RFR)	**11.66**
Support Vector Regressor (SVR)	12.34

Noticeably, Random Forest was able to perform and produce the most accurate results for the speeds of vehicles over pedestrian crossings. This proved that Random Forest was able to detect and model the intricacies of the time series, and was able to produce forecast values with the lowest Mean Absolute Percentage Error (MAPE) value of 11.66%. Additionally, Random Forest is able to learn the patterns using both the lagged speed and acceleration time series to predict the speed of the vehicles. Furthermore, as expected, due to the non-linear and non-stationary properties of the time series, the linear regression model was not suitable as it resulted in forecast values with the highest MAPE value of 28.44%.

Surprisingly, ARIMA and SARIMA were able to outperform Decision Tree with a difference in MAPE value of approximately 2%. However, this might be due to the fact that ARIMA and SARIMA are specifically designed for time series data, and inherently are able to model temporal dependencies better than when compared to Decision Tree. Even when the time series data does not observe linearity or stationarity, it may be able to partially account for the structural pattern, making them outperform decision tree in the task of time series forecasting.

Support Vector Regressor was the most accurate model after Random Forest with a MAPE value of 12.34%. This satisfied the hypothesis that mapping the input time series data into a higher dimensional space helps the model in learning the pattern in the time series, and indeed can use the learned pattern to produce results which are better than forecast values produced by linear or statistical models.

3.4 Forecasting using Neural Network Methods

Neural networks are mathematical models that were inspired from the design of the neural network inside the human brain [59]. They consist of interconnected node, that essentially act as neurons, which are organized into layers and control the amount of information that is passed between them as well as assign weights to the information that is being passed. As the information is passed from one layer to the next, the neural network starts to identify patterns and features which when added up, help in identifying complex representations. For time series data, as with traditional machine learning methods, the time series first needs to be transformed into a supervised learning problem prior to training the neural network. While doing so, it is essential to maintain the temporal sequence of the time series.

Historically, Recurrent Neural Networks (RNNs) have been used widely for the purpose of time series forecasting [60]. They keep track of hidden states while maintaining temporal dependencies in the data. Nevertheless, they do struggle with a problem called the *vanishing gradient* which restricts their capacity to learn long-term dependencies. Moreover, Convolutional Neural Networks, which are generally used in image classification tasks, have also been utilized for the purpose of time series forecasting [61]. They are useful in extracting patterns and features from images and can be also be extended to extract temporal patterns and feature from time series data.

Following is a discussion of the various neural network methods that were used for this study.

3.4.1 Convolutional Neural Networks

As mentioned before, Convolutional Neural networks (CNN) are primarily used for analyzing image data. In general, they are applied directly over the pixels of the image and are used to extract key patterns and features from them such as edges, textures, and other complex pattern that can be easily recognized by humans. The CNN, to extract patterns from images, applies a two-dimensional ($2D$) filter (or also known as kernel) over the image and is able to extract the features from images by using the convolution operation. The convolution operation performs an element-wise multiplication that involves the filter values and the pixel values of the image. Once the convolution filter has been applied on a certain section of the image, the filter is then moved around the image using a stride s, in an attempt to capture the entire image.

Similar to a Fully Connected Neural Network (FCNN), where each node in a layer is connected to every node in the next layer, a CNN also includes fully connected layers, where the output of the convolution layers are connected to one or more fully connected layers.

However, when a CNN is used for purposes such as time series forecasting, its architecture needs to be modified in order to be suitable for its application over a time series. In such cases, rather than utilizing a $2D-$CNN, the time series is treated as a one-dimensional ($1D$) image, and hence a CNN with a $1-$dimension layer is applied over the time series. This way when the convolution operation is used over the time series, it can analyse the sequence and extract the crucial patterns and features from the time series.

When the convolution operation is applied to a time series, it involves a dot product between a filter matrix $\mathrm{k} \in \mathbb{R}^{h \times d}$, where h and k are the length and width of the filter, and the input time series $\mathbf{x}$. However, in the case of a one-dimensional CNN, where the input has only one feature dimension ($d = 1$), the filter can be simplified and represented as a vector in $\mathbb{R}^h$. The input time series includes the lagged time series $\mathbf{x}$, which can be formally expressed for a uni-variate time series as

$$\mathbf{x} = [x_1, x_2, x_3, \dots, x_T] \in \mathbb{R}^T \tag{3.8}$$

where T is the length of the time window or also known as the sequence length, and each $x_t \in \mathbb{R}$ represents the observed value at time step t.

Nevertheless, as this study includes a multi-variate time series that includes speed and acceleration values of the vehicles, it is essential to define the input vector for a multi-variate time series, which can be expressed as

$$\mathbf{x} = [x_1, x_2, x_3, \dots, x_T] \in \mathbb{R}^{T \times C} \tag{3.9}$$

where, C is the number of multi-variate features, and and each $x_t = [x_t^{(1)}, x_t^{(2)}, \dots, x_t^{(C)}] \in \mathbb{R}^C$ represents the observed value at time step t.

Formally, the input vector for a input time series for both, a uni-variate and multi-variate time series, can expressed as

$$\mathbf{x}_{i:i+h+(s-1)} = \mathbf{x}_i \oplus \mathbf{x}_{i+s} \oplus \cdots \oplus \mathbf{x}_{i+h+(s-1)} \tag{3.10}$$

where i represents the i^{th} time step, and $\oplus$ denotes the concatenation operation applied over a sliding window of size h with stride s, resulting in a segment of the input time series used as input to the convolution filter. In the multivariate case, each $\mathbf{x}_t$ is a vector of features at time t, while in the univariate case, each x_t is a scalar value.

The dot product between filter k and input x results in

$$y_t = (\mathbf{x} * \mathbf{w})_t + b = \sum_{i=0}^{h-1} w_i \cdot x_{t-i} + b \tag{3.11}$$

where, y_t is the output of the convolution filter at time step t, $\mathbf{w} = [w_0, w_1, \dots, w_{h-1}]$ are the weights of the $1D-$convolution filter of size h, b is the bias term, and x_{t-i} is the input at the previous time step. The $*$ denotes the convolution operation between the input time series vector $\mathbf{x}$, and the weights of the filter k.

ReLu Activation Function

Additionally, the convoluted output of the convolution layer is passed onto a Rectified Linear Unit (ReLU). ReLU is an activation function that adds non-linearity to the input, which allows flexibility and enables the convolution network to effectively learn more complex and abstract patterns from within the time series. This can be expressed as follows

$$y_t = \mathrm{ReLU}\left(\sum_{i=0}^{k-1} w_i \cdot x_{t-i} + b\right) \quad \text{where} \quad \mathrm{ReLU}(z) = \max(0, z) \tag{3.12}$$

A note worthy observation is that the result after passing the convoluted output through the ReLU activation function is a single scalar value, represented in the feature map. Hence, the complete feature map can be expressed as

$$\mathbf{y}_t^{(j)} = \mathrm{ReLU}\left(\sum_{i=0}^{k-1} w_i^{(j)} \cdot x_{t+i} + b^{(j)}\right), \quad \text{for} \quad t = 1, \dots, T' \tag{3.13}$$

In the above feature map formulation, $\mathbf{x}$ represents the input time series, $\mathbf{w}^j$ represents the filter weights of the j^{th} filter, b^j is the bias term of the j^{th} filter, and $\mathbf{y}_t^{(j)}$ is the resulting feature map for the filter j, and T' is the output sequence length after the convolution.

Max Pooling

Along with the convolution layer and the ReLU activation function, the CNN also utilizes pooling layers. These pooling layers are used to reduce the spatial dimension of the input while retaining only the essential and important features. Max Pooling is a type of pooling layer that selects the maximum element from region of the feature map covered by the filter. By doing so, the output of the max pooling layer would contain the features and patterns of the most prominent features of the previous feature map.

If the feature map has length n_h, and the pooling operation uses a window of size p with stride s, then the max pooling is applied approximately $\lfloor \frac{n_h - p}{s} + 1 \rfloor$ times, each time selecting the maximum value within the pooling window.

This reduction in dimensionality helps decreasing the complexity of the CNN model, and helps in avoiding the overfitting of data. It provides an abstract representation, and introduces a factor where small shifts in the input do not drastically effect the models capability to producing an accurate output.

By combining convolution layers with ReLU activation functions and max pooling layers, a CNN model can effectively extract meaningful features from a time series. These extracted features can then either be used directly by the CNN for forecasting future values, or passed to subsequent layers that further process the information to make an accurate forecast of the values in the time series.

3.4.2 Long Short-Term Memory Networks

A Long Short-Term Memory (LSTM) network is a type of Recurrent Neural Network (RNN) designed to overcome the problem of *vanishing gradients* when learning from and modelling sequential data. RNNs are well-suited for sequential data because they can preserve and model the temporal structure of the data. Maintaining this temporal structure is essential, as it contains important information about the context and order of the sequence, which is crucial for tasks such as forecasting, where these properties significantly influence the output. In general, LSTMs have been widely used in applications such as natural language processing, speech recognition, and anomaly detection. Since time series data also rely heavily on sequence and context, LSTMs are particularly suitable for the task of time series forecasting.

LSTMs incorporate a series of gates, namely the input gate, forget gate, and output gate. These gates allow the network to retain long-term dependencies within a sequence by utilizing a memory cell. The input gate controls the amount of new information added to the memory cell by combining the current input and the previous hidden state. The forget gate determines which information in the memory cell should be discarded and which should be retained, allowing the model to selectively forget irrelevant information. Finally, the output gate decides what information from the memory cell should be output and passed on to the next LSTM cell as the new hidden state.

In each gate, the current input and the previous hidden state are combined and passed through a `sigmoid` activation function. The `sigmoid` output is then used to scale the information, which is often further processed by a `tanh` activation function to determine what information is stored, discarded, or retained as the hidden state.

In an LSTM, the gates at time step t manage how information flows through the network and how the hidden state h_t is updated. The hidden state h_t is computed based on the current input $\mathbf{x}'_t$, the previous hidden state h_{t-1}, the forget gate f_t, the output gate o_t, and the updated memory cell state C_t, which together determine what information is retained, discarded, or passed on to the next time step.

The forget gate vector $\mathbf{f}_t$ produces a value between 0 and 1 using the sigmoid function, indicating how much information in the cell state should be retained. A value closer to 0 indicates the information needs to be forgotten, whereas a value closer to 1 indicates that the information should be remembered and stored in the cell state. This is done using the previous hidden state h_{t-1} and the current input at time step t, x'_t, and concatenation them together, which enables the LSTM to capture both past and current context. Additionally, the weight matrix $\mathbf{W}_f$ which is learned while training the LSTM, is used to transform the concatenated input. Finally, a bias factor b_f is also added to the forget gate. This forget gate $\mathbf{f}_t$ can be formally expressed as

$$f_t = \sigma(\mathbf{W}_f[h_t \oplus \mathbf{x}'_t] + \mathbf{b}_f) \tag{3.14}$$

Similar to the forget gate f_t, the input gate i_t also uses the current input x'_t and the previous hidden state h_{t-1}, which are concatenated together and transformed using the weight matrix of the input gate $\mathbf{W}_i$ with a bias factor b_i. The input gate extends this and also uses the current input x'_t and the previous hidden state h_{t-1} with a `tanh` function to create a candidate cell state C'_t. They are expressed as follows

$$i_t = \sigma(\mathbf{W}_i[h_t \oplus \mathbf{x}'_t] + \mathbf{b}_i) \tag{3.15}$$

$$C'_t = tanh(\mathbf{W}_c[h_t \oplus \mathbf{x}'_t] + \mathbf{b}_c) \tag{3.16}$$

The LSTMs updates the cell states using the forget gate f_t and the past time step cell state C_{t-1}, which scales the amount of previous memory in order to limit the information that is needed to be kept. On the other hand, the input gate i_t and the candidate memory cell C'_t are used to scale the candidate memory in order to accept the information. These are then combined together to retain the useful past information and integrate relevant new information. This is defined as

$$C_t = f_t \oplus C_{t-1} + i_t \oplus C'_t \tag{3.17}$$

Finally, the output gate o_t also utilizes the current input x'_t and the previous hidden state h_{t-1}, which are concatenated together and transformed using the weight matrix of the output gate $\mathbf{W}_o$ with a bias factor b_o. This result of the output gate then undergoes element-wise multiplication using the current cell state C_t after it has been passed through the `tanh` activation function. The $tanh(C_t)$ converts the cell state into a form such that when combined with the output of the output gate it decides how much the information is suitable enough for passing it as the hidden state h_t for the next step in the model. These can be expressed as

$$o_t = \sigma(\mathbf{W}_o[h_t \oplus \mathbf{x}'_t] + \mathbf{b}_o) \tag{3.18}$$

$$h_t = o_t \oplus tanh(C_t) \tag{3.19}$$

Bidirectional Long Short Term Memory (BiLSTM)

An extension for a LSTM, called a Bidirectional LSTM or BiLSTM, consists of 2 layers of LSTM nodes [62]. For time series data, a BiLSTM will process it in the forward direction in the first layer, where as it will be processed in the backward or reverse direction in the second layer.

This is done in an attempt to capture the long-term dependencies of the time series, in both the forward and backward direction. This way, the BiLSTM would be able to capture time-dependent representation within the time series better than a unidirectional LSTM, that processes the time series only in one direction.

3.4.3 Autoencoders

Autoencoders (AEs) are a type of neural network primarily used for unsupervised learning tasks, such as feature extraction and dimensionality reduction. They are designed to learn efficient data representations by reconstructing their input as accurately as possible.

The architecture of an autoencoder typically consists of three main components: the encoder, the latent space (also referred to as the *bottleneck*), and the decoder. The encoder compresses the input data into a lower-dimensional representation within the latent space. This latent space captures a dense, abstract representation of the original input, retaining only the most critical information. The decoder then utilizes this compressed representation to reconstruct the original input data, employing a series of linear transformations and non-linear activation functions.

In essence, autoencoders reduce the dimensionality of the input data by compressing it into a more compact form, which not only reduces its complexity but also forces the network to learn and preserve the most crucial features while discarding redundant or non-essential information. As a result, the latent representation serves as an abstract, information-rich encoding of the original data.

The dimensionality of the latent space plays a crucial role in determining the quality and amount of information preserved. When the latent space is smaller than the input space, the model is encouraged to learn a more compact and efficient encoding, which can be beneficial for applications such as noise reduction or anomaly detection. Conversely, a larger latent space allows the model to capture more detailed and complex features of the input data, though at the risk of retaining unnecessary information.

Finally, the decoder reconstructs the input by mapping the latent representation back into the original high-dimensional space. This process involves learning a function that can accurately translate the compressed features into a close approximation of the original data. The decoder uses linear transformations to model affine relationships and activation functions to introduce non-linearities, allowing it to capture complex patterns and structures present in the data.

While utilizing autoencoders for time series data, the autoencoders accepts an sequence of time series observation and attempts to learn the complex representations by capturing the key dynamics of the time series. For an input time series $\mathbf{x} = [x_1, x_2, \ldots, x_T]$ where $x_t \in \mathbb{R}^d$, T is the length of the sequence and d is the number of feature of the time series, the encoder mapping of the input can be expressed as follows

$$\mathbf{z} = f_{enc}(\mathbf{x}; \theta_{enc}) \tag{3.20}$$

where, f_{enc} is the type of encoder(for this study, an LSTM was utilized), θ_{enc} are the parameters of the LSTM, and $z \in \mathbb{R}^k$, where $k \ll T \cdot d$.

This latent representation $\mathbf{z}$ that is obtained from the encoder, is then used by the decoder in an attempt to recreate the original input $\mathbf{x}$. This reconstruction can be expressed as

$$\hat{\mathbf{x}} = f_{dec}(\mathbf{z}; \theta_{dec}) \tag{3.21}$$

where, f_{dec} is the decoder function (also an LSTM), θ_{dec} are the parameters of the LSTM, and $\hat{\mathbf{x}} = [\hat{x_1}, \hat{x_1}, \ldots, \hat{x_T}]$ is the reconstructed time series.

In this thesis, the Autoencoder was used to reconstructed time series, and extend its model architecture to forecast the time series. On the other hand, the autoencoder was also combined with another separate LSTM to create a hybrid architecture, where the autoencoder would perform as a feature extractor, and the extracted features were used by the LSTM to forecast the time series.

3.4.4 Results using Neural Network Methods

The results of using Neural Network based models for predicting the speed of vehicles over pedestrian crossings at signalized intersection are presented in Table 3.2

Table 3.2: MAPE values using Neural Network Models

Models	**Forecasting MAPE values**
CNN	15.32
LSTM	13.70
BiLSTM	13.98
AE	15.32
CNN-LSTM	**12.76**
AE-LSTM	13.39
CNN-BiLSTM	13.24

Along with using the various Neural Network based models by themselves, this study also utilized models where 2 or more models were combined to create hybrid deep learning models. For instance, along with separately using a Autoencoder and a LSTM model for forecasting, they were also combined together to form a hybrid deep-learning model Autoencoder-LSTM (AE-LSTM). Similarly, a CNN model was combined with a LSTM and BiLSTM model respectively, to create hybrid deep learning models, namely, CNN-LSTM, and CNN-BiLSTM. All hybrid models created were combined with a fully connected layer (FC) and a single node as the output layer.

Among all the neural network, deep learning, and hybrid models that were used to forecast the speed of the vehicles, the hybrid CNN-LSTM model was able to produce the result with the lowest Mean Absolute Percentage Error (MAPE) value of 12.76%. Additionally, it was also noted that all hybrid models produced MAPE values that were lower compared to the standalone models.

Furthermore, BiLSTM performed poorly when compared to the standard LSTM model, in both the standalone and hybrid models. In the standalone model, the BiLSTM produced forecast values with MAPE value 13.98%, when compared to the standalone LSTM which had a 0.28% less error, with a MAPE value of 13.70%. When combined with a CNN to form hybrid models, the CNN-BiLSTM model was able to forecast with a MAPE value of 13.24%, which was 0.48% higher compared to the CNN-LSTM model which produced forecast with a MAPE value of 12.76%.

All models were trained for 100 epochs. The CNN-LSTM model comprised for 4 CNN layers and 3 Max Pooling layer. Each convolution layer consisted of a ReLU activation function. On the other hand, the second half of the hybrid model consisted of 4 LSTM layers each using a `tanh` activation function. To determine the result of the CNN-LSTM model, hyperparameters such as the batch size, number of filter, kernel size, pool size in the pooling layers, and number of nodes in the hidden layer had to be determined. Additionally, hyperparameters such as number of LSTM units, dropout rate, and learning rate of the model also had to be determined. To optimize model performance, both Grid Search and Bayesian hyperparameter optimization techniques were employed. The method that identified the model with the lowest MAPE was selected as the preferred approach for tuning the model's hyperparameters.

For both Grid Search and Bayesian optimization, the CNN model was evaluated using various combinations of hyperparameters: the number of filters was set to 16, 32, 64, or 128; kernel sizes of 3, 5, and 7 were tested; pool sizes of 2, 3, and 4 were considered; the number of nodes in the dense layer was set to 100, 200, or 300; and a batch size of 16, 32, and 64 were set. Similarly, the LSTM model was tuned using different values for its hyperparameters: the number of LSTM units was set to 64, 128, or 256. Additionally, both models were tested with dropout rates of 0.1, 0.3, and 0.5, and learning rates of 0.01, 0.001, and 0.0001.

Between the grid search and Bayesian optimization technique, the Bayesian optimization technique produced optimal hyperparameters for the model. The hyperparameter that were chosen for the CNN model were, 64 filters, a kernel size of 3, a pool size of 2, and 100 nodes in the dense layer, and a batch size of 32. One the other hand, 128 LSTM units were chosen for the LSTM layer, whereas, a dropout rate of 0.3, and a learning rate of 0.001 were chosen for both the CNN and LSTM models.

3.5 Hybrid Model: TCN-LSTM-RFR

From the discussions in section 3.3.5 and section 3.4.4, we note the following:

1. Table 3.1 and Table 3.2 show that among all models, including traditional machine learning and neural network/deep/hybrid models, the Random Forest model was the best performing model with the lowest MAPE value of 11.66%.
2. Table 3.2 shows that among the neural network/deep learning based model, the hybrid CNN-LSTM architecture was the best performing model with a MAPE value of 12.76%.

Taking this into account, the CNN component of the hybrid CNN-LSTM model was effective in identifying important patterns and features within the time series data, particularly short-term patterns. These extracted short-term features were then passed to the LSTM, which is well-suited for capturing long-term dependencies. By combining these capabilities, the model was able to forecast vehicle speeds at pedestrian crossings with improved accuracy

It was observed that enhancing the feature extraction process, particularly in capturing short-term patterns within the time series, could significantly improve the performance of the LSTM model. A more precise extraction of these patterns would enable the LSTM to generate forecasts with greater accuracy and precision. To achieve this, the convolution component of the CNN-LSTM architecture was replaced with a more advanced variant known as the Temporal Convolutional Network (TCN), which is specifically designed to better capture temporal dependencies in sequential data. Additionally, since the Random Forest Regressor (RFR) alone outperformed the neural network-based models, it was also considered for inclusion in the hybrid model.

The upcoming chapters will discuss the TCN in detail, and then provide a detailed version of the complete hybrid TCN-LSTM-RFR architecture that provided the speed of the vehicles over pedestrian crossings with the lowest MAPE value among all models.

3.5.1 Temporal Convolutional Networks

Temporal Convolutional Networks (TCN), are an extension of CNNs, that focus on extracting features and pattern from a time series while maintaining the temporal structure of the time series. Unlike, RNN where the data is processed sequentially, a TCN utilizes $1D-$convolution layers with causal and dilated convolution to model the temporal patterns and features of the data.

In contrast to standard convolution, the *causal* property of the TCN ensures that when the time series undergoes convolution, it does not include future timestamps in doing so. It ensures that the output of the convolution layer at a certain timestamp is only dependent on its past and present timestamps. Hence, this property of TCNs make it suitable for task related to time series, such as time series forecasting.

For instance, when modeling a time series, it is crucial to prevent information leakage by ensuring that the output at the current time step t depends only on past values $[t-1, t-2, \dots, t-n]$ and should not add future value such as $[t+1, t+2, \dots, t+T]$. However, this requirement is failed while using standard convolution layer as, for a filter of size k, the output y_t for a timestamp t is computed as

$$y_t = \sum_{i=o}^{k-1} w_i \cdot x_{t+i-[k/2]} \tag{3.22}$$

Hence, given an input sequence $\mathbf{x} = [x_0, x_1, x_2, x_3, x_4]$, and filter size of $k = 3$, where $\mathbf{w} = [w_0, w_1, w_2]$ are the weights of the filter, the standard convolution output at $t = 2$

would cover

$$y_2 = w_0 \cdot x_1 + w_1 \cdot x_2 + w_2 \cdot x_3 \tag{3.23}$$

where, x_3 is a future input with respect to $t = 2$.

However, this issue is mitigated by using causal convolution as they do not consider future timestamps $t + 1$ when computing the output at the current timestamp t. The computed output y_t at timestamp t using causal convolution with filter size k can be expressed as follows

$$y_t = \sum_{i=0}^{k-1} w_i \cdot x_{t-i} \tag{3.24}$$

where, y_t is only computed using $x_t, x_{t-1}, x_{t-2}, \dots, x_{t-k+1}$, ensuring no information leakage from the future.

Therefore, given and input sequence $\mathbf{x} = [x_0, x_1, x_2, x_3, x_4]$, and filter size of $k = 3$, where $\mathbf{w} = [w_0, w_1, w_2]$ are the weights of the filter, the causal convolution output at $t = 2$ would cover

$$y_2 = w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2 \tag{3.25}$$

where, x_0, x_1 and x_2 are the past and present inputs, maintaining the temporal structure, and causality.

When forecasting, as mentioned previously, the output of a specific timestamps is dependent on its present and previous timestamps. This can be achieved in convolution layers by adjusting the size of the receptive field. A smaller receptive field would only allow the convolution layer to keep track of timestamps that are close to the timestamps whose output is being computed. Increasing the receptive field allows a larger number of timestamps to be taken into account while applying the convolution operations, consequently allowing the convolution layers to extract not just short-term, but also long-term features and patterns.

However, this has a considerable impact on the computational resources and time that are used while performing the convolution operation. The reason for this is because, as the size of the receptive filed grows larger, more number of weights are involved and hence more amount of computation need to be done. Thus, to avoid these issue, the concept of *dilation* in convolution layers was introduced. A dilated convolution introduces gaps between the filter elements, allowing the network to cover a wider temporal context using fewer layers. In essence, dilation is the number of timestamps that are skipped between the elements of the filter.

Formally, for a $1D$ input sequence x, a dilated convolution with filter w of size k, and a dilation factor d at a timestamp t can be expressed as

$$y_t = \sum_{i=0}^{k-1} w_i \cdot x_{t-d\cdot i} \tag{3.26}$$

where, the dilation factor d controls the spacing between the elements of the filter in the input. When $d = 1$, the convolution is reduced to a normal causal convolution layer, but as the value of d is increased, the convolution layer starts to skip the timestamps of the input, and allows the receptive filed to grow larger in size.

Due to dilation in convolution layers, the output at timestamp t is influenced by inputs up to $d(k-1)$ steps in the past. This design enables the casual convolution layer to observe further back in time and gain significant information about the long-term patterns of the time series. Nevertheless, when multiple convolution layers are stacked with the same dilation factor d, the receptive field becomes spares, and as a result, the model fails to incorporate short-term information at timestamp t due to its failure to observe timestamps between t and $t-d$.

To mitigate this limitations of fixed dilation rates, multiple causal convolution layers are stacked with progressively increasing dilation factors. This architectural design enables the model to effectively capture both short-term and long-term pattern and features from the time series. In line with common practice in the literature, and as adopted in this study, the dilation rate is typically doubled at each successive causal convolution layer.

3.5.2 Complete Hybrid TCN-LSTM-RFR Architecture

As the TCN model has been proven to be better at extracting features compared to a CNN [63], the CNN from the hybrid CNN-LSTM model was replaced with a TCN. Hence, the new hybrid model was a combination of a Temporal Convolutional Network, and a Long Shot-Term Memory network (TCN-LSTM). The combined hybrid TCN-LSTM model also included a Fully Connected (FC) layer, which received the output of the LSTM model as an input. The TCN guarantees that every output from each timestamp of the time series only depended on its present and past timestamp values, without any information leakage, ensuring that the entire time series is convoluted while maintaining its temporal structure. The LSTM units are combined together for each timestamp to produce hidden state vectors to form a unified representation. The unified output are then passed down to a fully connected layer which enables the further transformation and integration of the temporal features.

Since RFR produced forecast values with the lowest MAPE value among both traditional and neural network based methods, it was also incorporated with the hybrid TCN-LSTM model to created a hybrid traditional learning and deep learning architecture. The TCN-LSTM act as feature extractors, where the TCN extracts short-term patterns and features from the time series, and using those short-term patterns the LSTM is able to retrieve the long-term dependencies of the time series. When both the short and long-term patterns and dependencies have been extracted, they are passed through a fully connected layer which combines the output of both the TCN and the LSTM, and the output of this fully connected layer is then passed onto the Random Forest Regressor to make the final forecast of the vehicle's speed over pedestrian crossings. The hybrid model employed a dual-headed architecture, where the two time series inputs, speed and acceleration, were processed through separate branches.

When the two time series are passed as inputs to the model, first the TCN for each timestamp t, with dilation rate d, and filter size k, produces an output h_t, which can be expressed as

$$\mathbf{h}_t = \sum_{i=0}^{k-1} \mathbf{W}_i \cdot \mathbf{x}_{t-d\cdot i} + \mathbf{b} \tag{3.27}$$

where, $\mathbf{W}_i \in \mathbb{R}^{C_{in} \times C_{out}}$ are the causal convolution filter weights, $\mathbf{x}_{t-d\cdot i}$ are the present and past input timestamps of the time series, and $d = 2^l$ is the dilation rate at layer l.

This output of the TCN $\mathbf{h}_t$ is then passed on to LSTM, which produces an output for speed and acceleration each $\mathbf{z}_s$ and $\mathbf{z}_a$ which are formulated as

$$\mathbf{z}_s = \text{LSTM}(\text{TCN}_s(\mathbf{X}_s)) \tag{3.28}$$

$$\mathbf{z}_a = \text{LSTM}(\text{TCN}_a(\mathbf{X}_a)) \tag{3.29}$$

which are then combined and passed to a fully connected layer with a ReLU activation layer

$$\mathbf{z} = [\mathbf{z}_s; \mathbf{z}_a] \in \mathbb{R}^d \tag{3.30}$$

$$\mathbf{h}_{\text{dense}} = \text{ReLU}(\mathbf{W}_d \cdot \mathbf{z} + \mathbf{b}_d) \tag{3.31}$$

$$h_{\text{drop}} = \text{Dropout}(\mathbf{h}_{\text{dense}}) \tag{3.32}$$

Finally, h_{dense} which are the feature embedding that have been extracted by the TCN and LSTM are passed to the RFR model $\mathcal{F}$

$$\hat{y} = \mathcal{F}(h_{\text{dense}}) \tag{3.33}$$

where $\mathcal{F}$ consist of M decision trees $T_1, T_2, T_3, \ldots, T_M$, each predicting $\hat{y}_i$, using which the final predict $\hat{y}$ is made.

$$\hat{y} = \frac{1}{M} \sum_{i=1}^{M} T_i(h_{\text{dense}}) \tag{3.34}$$

3.5.3 Results using TCN-based Hybrid Models

The result of forecasting the speed of the vehicles over pedestrian crossings using the hybrid TCN-LSTM-RFR model along with a few other hybrid model created using Temporal Convolutional Networks (TCN) are listed in Table 3.3

Table 3.3: MAPE values using hybrid TCN-LSTM-RFR and other TCN models

Models	**Forecasting MAPE values**
TCN	32.67
TCN-LSTM	11.78
TCN-BiLSTM	12.90
TCN-BiLSTM-RFR	11.79
TCN-LSTM-RFR	**11.14**

The hybrid TCN-LSTM-RFR model achieved the most accurate forecasting performance among all evaluated models, which include both standalone and hybrid, traditional and deep learning, by attaining the lowest Mean Absolute Percentage Error (MAPE) of 11.14%. Notably, the hybrid deep learning model TCN-LSTM outperformed its CNN-LSTM counterpart, achieving a MAPE of 11.78%, which represents a 0.98% improvement. This result highlights the effectiveness of incorporating a TCN in place of a standard Convolutional Neural Network (CNN), as TCNs leverage causal and dilated convolutions to more effectively capture temporal dependencies and extract meaningful patterns from time series data. Furthermore, the integration of a Random Forest Regressor (RFR) into the TCN-LSTM architecture contributed to an additional performance gain of 0.64%, underscoring the benefit of combining deep learning with ensemble-based regression techniques for enhanced forecasting accuracy.

It should be noted that, due to the randomly initialized weights in neural networks, training the same model configuration on the same data can result in different final trained

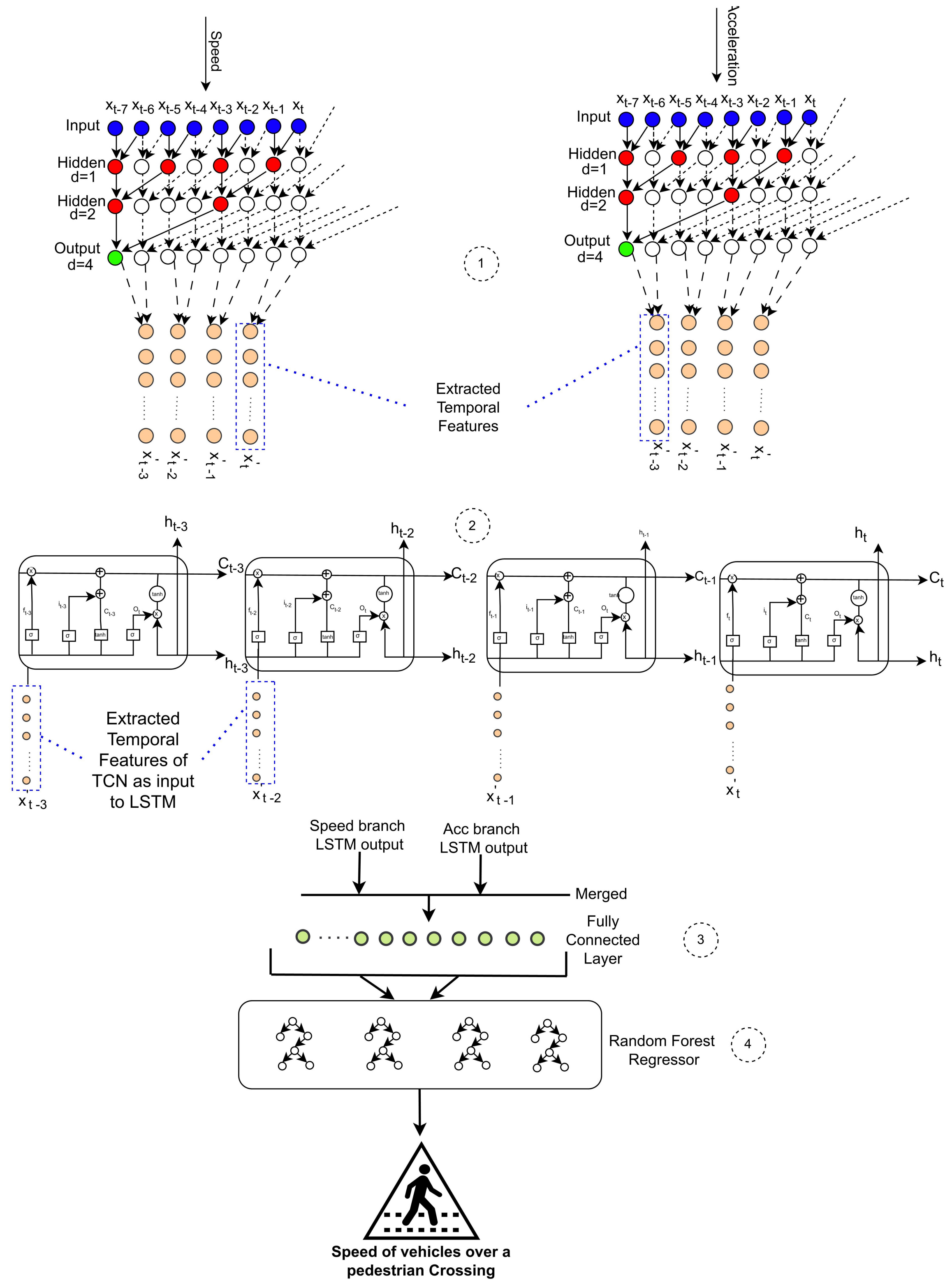

Figure 3.2: Hybrid TCN-LSTM-RFR model architecture for forecasting vehicle speed over a pedestrian crossing. Inputs, speed and acceleration, enter the TCN (Stage 1) to extract short-term temporal features. In Stage 2, these features are passed to the LSTM to capture long-term dependencies. Stage 3 merges outputs from both branches via a fully connected layer, and Stage 4 uses an RFR to forecast vehicle speed.

weights. As a result, in each run, hyperparameters were selected using 5-fold cross-validation, and the models were retrained on the full training set before evaluation. The mean results are presented in Table 3.3.

Given the hybrid nature of the model, it was crucial to optimize it using the most effective set of hyperparameters. To achieve this, both grid search and Bayesian optimization techniques were employed. For the TCN component of the model, in addition to the standard hyperparameters used in the CNN, such as filter size, number of filters, batch size, learning rate, dropout rate, and pooling size, additional TCN specific parameters like dilation rate and the number of TCN blocks were also tuned. Similarly, hyperparameters for the LSTM layers, RFR, and the fully connected (FC) layers were carefully selected.

As with the other deep learning architectures in this study, the hybrid TCN-LSTM-RFR model was trained over 100 epochs. For the TCN model, a comprehensive evaluation of various hyperparameter combinations was performed: the number of filters was varied among 16, 32, 64, and 128; kernel sizes of 3, 5, and 7 were tested; and pooling sizes of 2, 3, and 4 were explored. The dilation rate in the TCN layers was progressively increased by a factor of 2, such that the first TCN layer had a dilation rate of 1, followed by 2 and 4 for the second and third layers, respectively. The number of TCN blocks was also tested with values of 3, 4, and 5.

The hyperparameters for the LSTM component in the hybrid model were consistent with those used in the CNN-LSTM model. Specifically, the number of LSTM units was set to 64, 128, or 256. For the fully connected layers, the number of neurons was varied among 100, 200, and 300, with batch sizes of 16, 32, and 64. As for the Random Forest Regressor, hyperparameters included the number of estimators (set to 50, 100, 200, and 300), maximum tree depth (None, 5, 10, 15, and 20), minimum samples required to split a node (2, 3, 5, and 6), and minimum samples per leaf node (1, 2, and 3).

After hyperparameter tuning and model optimization, Grid Search achieved a lower MAPE than Bayesian Optimization. Therefore, Grid Search was used to select the final hyperparameters for the hybrid TCN-LSTM-RFR model. The chosen hyperparameters are listed in Table 3.4.

Table 3.4: Parameter Optimization Configurations : TCN-LSTM-RFR Using Grid Search

Model	**Parameters**	**Options**
TCN-LSTM	Number of filters	16, 32, **64**
	Kernel size	**3**, 5, 7
	Pooling Size	**2**, 3, 4
	Number of blocks	**3**, 4, 5
	Batch Size	16, **32**, 64
	LSTM units	64, **128**, 256
	Dropout Rate	0.1, **0.3**, 0.5
	Learning rate	0.01, **0.001**, 0.0001
FC	Nmmber of nodes	**100**, 200, 300
RFR	Number of estimators	50, 100, 200, **300**
	Maximum depth	None, 5, 10, **15**, 20
	Min samples to split node	**2**, 3, 5, 6
	Min samples per leaf node	**1**, 2, 3

3.6 Summary

The chapter presented a hybrid traditional and deep learning model that is a combination of Temporal Convolutional networks, Long Short-Term Memory, and a Random Forest Regressor (TCN-LSTM-RFR) model. The data with which the model was evaluated was first introduced. The dataset was obtained from the U.S. Department of Transportation - Federal Highway Administration. For every vehicle that entered and exited 4 intersection on Lanskerhim Boulevard, their speed and acceleration data was recorded. Specifically, their speed over pedestrian crossings was also noted in this dataset. The data was pre-processed such that it was converted into a supervised learning problem, as well as to simulate real-world scenarios, the last 3 seconds of data for all vehicles was excluded from the dataset. The problem that needed to be solved was to forecast the target speed for all vehicles over pedestrian crossings. Hence, an evaluation metric, Mean Absolute Percentage Error (MAPE), was chosen for this study.

First experiment involved using statistical and traditional machine learning models to forecast the speed of the vehicles over pedestrian crossings. The models considered for this study were linear and statistical models (LR, ARIMA, SARIMA), Decision Tree (DT), Random Forest Regressor (RFR), and Support Vector Machine (SVR). Among them, the Random Forest Regressor was able to produce the best forecast value with the lowest MAPE value of 11.66%.

Next, the experiments involved using Neural Network based models to forecast the speed of the vehicles over pedestrian crossings. The models considered for this study were Convolutional Neural Networks (CNN), Long Short-term Memory (LSTM), and extension of a LSTM, called a BiLSTM, and Autoencoders (AE). Along with that, a combination of deep learning models (hybrid models), such as CNN and LSTM (CNN-LSTM), CNN and a BiLSTM (CNN-BiLSTM), and Autoencoder and LSTM (AE-LSTM) were also used for this study. Among them the hybrid CNN-LSTM was able to produce the best forecast value with the lowest MAPE value of 12.76%

Noticing that perhaps a better feature extractor than a CNN might improve the final forecast value of the CNN-LSTM, the CNN from the CNN-LSTM model was replaced with an extension of a CNN called a Temporal Convolutional Network (TCN). The TCN model included using causal-dilated convolution layer instead of standard convolution which enabled to extracted both short and long-term pattern effectively from the data.

The hybrid TCN-LSTM model was able to improve the forecast value producing a MAPE value of 11.78%. Nevertheless, noticing that the Random Forest Model was able to produce a MAPE that was lower compared to the hybrid TCN-LSTM model, it was incorporated into the hybrid model to create the hybrid traditional and deep learning TCN-LSTM-RFR model. The hybrid model was able to produced the best result compared to all other models that were used in this study with a MAPE value of 11.14%.

4 Two-stage Framework with Imputation and Dynamic Dilation in Temporal Convolutional

This chapter presents the two-stage framework that is used to predict the speed of the vehicles over pedestrian crossings. The framework consists of an imputation stage, followed by a forecasting stage that utilizes the hybrid TCN-LSTM-RFR model described in Section 3.5. As previously mentioned, for this framework to be effectively used in autonomous vehicles (AVs) and to provide drivers with sufficient time to take necessary actions, the advisory speed must be communicated to the driver of the AV before the vehicle reaches a pedestrian crossing. To simulate this scenario, 3 seconds of speed and acceleration data were removed from the dataset. However, imputing this 3 seconds of removed data and integrating it with the training dataset for the forecasting model could potentially enhance the forecast accuracy [64]. Therefore, the proposed framework first performs imputation and then passes the resulting data to the hybrid forecasting model.

Furthermore, this chapter also includes a reinforcement learning approach for dynamically selecting the dilation rate in each layer of the TCN that is used in the forecasting stage. This chapter begins by introducing the models used for imputing the removed data, followed by a discussion of the imputation results. It then outlines the evaluation of the two-stage framework using various methods and presents the forecasting performance of the models in predicting vehicle speed over pedestrian crossings using the imputed data. Finally, it explains the reinforcement learning approach used to select the optimal dilation rates in each layer of the TCN. The last section of this chapter provides a brief discussion that summarizes the work that was done in this chapter.

4.1 Imputation of Speed and Acceleration Data

Imputation is the process of replacing missing data with alternative values. Missing data poses significant challenges for data analysis, as it complicates data processing and reduces the performance of models trained on incomplete datasets [65].

Typically, when a dataset contains missing values, studies often opt to exclude the affected records. However, imputation offers a valuable alternative, although its implementation can be complex. When performing imputation, it is essential to preserve the integrity of the data. Moreover, the method of imputation should be determined by the relationships between the missing values and the available data.

There are three main types of missing data:

1. Missing Completely at Random (MCAR): The probability of missing data is the same for all samples, with no relationship to other variables.
2. Missing at Random (MAR): Missing values are distributed randomly but are related to other observed variables.
3. Missing Not at Random (MNAR): Missing data is related to unobserved factors or events that influence the missingness.

In this study, data was deliberately removed to simulate real-world scenarios, resulting in a classification of Missing Not at Random (MNAR). Given the temporal nature of the data, preserving the temporal dependencies during imputation is crucial. Additionally, it is important to maintain the trends and seasonality of the time series data while performing imputation.

Due to the constraints stated, common imputation methods, such as Last Observation Carried Forward (LOCF), Next Observation Carried Backward (NOCB), Mean/Median/Mode imputation, or K-Nearest Neighbor (KNN) imputation are not suitable for this study, and require advanced machine/deep learning models to impute the removed data.

The following sections will discuss the various models used for imputation in this study and present the results of the imputation process.

4.1.1 Classical and Hybrid Deep Learning Methods

To impute the removed data in the time series, both vehicle speed and acceleration data were utilized. The imputation process was performed for both speed and acceleration variables simultaneously. The models used had a multi-headed architecture, allowing them to accept two inputs and produce two outputs at the same time. Specifically, both speed and acceleration data were fed into the model, which first imputed the vehicle speed and then the acceleration.

LSTM, BiLSTM, TCN-LSTM, and TCN-BiLSTM

Several recurrent and hybrid models were used to impute the last 3 seconds (30 timestamps) of speed and acceleration for each vehicle. These included standalone LSTM and BiLSTM models, as well as hybrid architectures like TCN-LSTM and TCN-BiLSTM. All models were connected to a fully connected layer that transformed the hidden representations into final output predictions.

The LSTM-based models modeled the temporal dependencies in the sequence, while BiLSTM extended this by processing the lagged timestamps in both forward and backward directions to improve the modeling of temporal relationships.

In the hybrid TCN-LSTM model, adapted from the TCN-LSTM-RFR architecture described in Section 3.5.2, the RFR component was excluded due to its limitation in producing multiple time-step time series forecasts. The TCN, described in section 3.5.1, produced a temporal sequence which was then passed to the LSTM to model long-term dependencies. The LSTM output was subsequently passed to a fully connected layer to generate the imputed outputs of 30 timestamps.

The TCN-BiLSTM model followed a similar structure but replaced the LSTM with a BiLSTM to better capture bidirectional dependencies. Instead of receiving the original lagged time series directly, the BiLSTM took the output of the TCN layers as input.

All these models shared the same multi-branch input design and jointly imputed both speed and acceleration timestamps.

CNN and CNN-LSTM

Furthermore, a classical one-dimensional CNN, similar to the one used for forecasting as explained in section 3.4.1, was also used to impute the removed timestamps. Like the other models, it was structured with two input branches (lagged speed and acceleration) and connected to a fully connected layer for dual-output generation of 30 speed and acceleration timestamps. Unlike TCNs, CNNs use standard convolution layers, i.e. without causal or dilated layers.

Additionally, to enhance temporal modeling, a hybrid CNN-LSTM model, similar to one used for forecasting, was also employed for imputation. The CNN component extracted short-term temporal features from the input, which were then passed to the LSTM to capture long-term dependencies. The LSTM's output was finally mapped to impute speed and acceleration values through a fully connected layer.

4.1.2 Attention based Methods for Imputation

After evaluating the classic and hybrid deep learning methods for imputing speed and acceleration data, this study also explored models that included attention and self-attention mechanisms in their architectures. Attention mechanisms have been widely used in the field of Natural Language Processing (NLP), and because time series data are also sequential, they can be effectively used for imputing the removed values in this context. An attention mechanism helps a model focus on the most relevant parts of the input data, allowing it to prioritize important information while ignoring less significant details. By doing so, attention layers enable the model to capture the most meaningful patterns and improve the accuracy of its predictions [66].

On the other hand, self-attention, which has a similar relationship as attention mechanism with input and output sequence, allows a model to process the sequence as whole, capturing dependencies between all tokens in the same sequence [66]. The key difference between an attention and self-attention mechanism is that, an attention attempts to capture relationships *between* sequences, while a self-attention mechanism attempts to capture relationship within *same* sequence.

With respect to that, this study used two models; A sequence to sequence architecture, that was connected to an attention mechanism, called a Seq2Seq Attention model(Seq2Seq-Attention) [67], and another called Temporal Fusion Transformer(TFT) [68] that utilized a multi-headed self attention mechanism to impute the removed data. Both these models accept speed and acceleration data as inputs as 2 separate branches, and configured to output a sequence of 30 imputed speed and acceleration timestamps.

Sequence to Sequence Attention

The dual-branched Seq2Seq-Attention model [67] leverages the temporal features of both time series (speed and acceleration) and uses its encoder-decoder structure to create latent representations of the data. The encoder consists of LSTM layers that process each input time series in parallel. The latent representations produced by each encoder are then combined to form a shared initial state. This concatenation enhances the representation of patterns in the lagged speed and acceleration time series, which is then passed to the attention layers. The attention mechanism assigns higher weights to the most important parts of these combined latent features, allowing the model to focus on the most relevant information.

After this, the shared representation is passed through a series of decoders, which are also built using LSTM layers. These decoders take the output from the encoder and combine it with the context vector produced by the attention mechanism. This combined information is then used to reconstruct the time series and impute the removed data. By dynamically learning to weigh different parts of the input sequence, the attention mechanism helps the decoder focus on the most relevant temporal patterns at each time step, enabling accurate reconstruction and imputation of the removed values.

The sequence to sequence encoder-decoder model along with the attention mechanism can be described formally as

$$\mathbf{h}_t^{(v)}, \mathbf{c}_t^{(v)} = \mathrm{LSTM}_{\mathrm{vel}}(v_t, \mathbf{h}_{t-1}^{(v)}, \mathbf{c}_{t-1}^{(v)}) \tag{4.1}$$

$$\mathbf{h}_t^{(a)}, \mathbf{c}_t^{(a)} = \mathrm{LSTM}_{\mathrm{acc}}(a_t, \mathbf{h}_{t-1}^{(a)}, \mathbf{c}_{t-1}^{(a)}) \tag{4.2}$$

$$\mathbf{h}_0^{(dec)} = \mathrm{Concat}(\mathbf{h}_T^{(v)}, \mathbf{h}_T^{(a)}) \tag{4.3}$$

$$\mathbf{c}_0^{(dec)} = \mathrm{Concat}(\mathbf{c}_T^{(v)}, \mathbf{c}_T^{(a)}) \tag{4.4}$$

$$\mathbf{h}_s^{(dec)}, c_s^{(dec)} = \mathrm{LSTM}_{dec}(\tilde{x}_k, \mathbf{h}_{k-1}^{(dec)}, \mathbf{c}_{k-1}^{(dec)}) \quad \text{where} \quad \tilde{x}_k = \{\tilde{v}_k, \tilde{a}_k\} \tag{4.5}$$

where, $\mathbf{h}_t^{(v)}$ and $\mathbf{h}_t^{(a)}$ are the hidden state encoder outputs for speed v and acceleration a, $\mathbf{c}_t^{(v)}$ and $\mathbf{c}_t^{(a)}$ are the cell states encoder outputs for speed and acceleration, which are concatenated with each other $\mathbf{h}_0^{(dec)}$, $\mathbf{c}_0^{(dec)}$ before being passed to the decoder with input at timestamp k as $\tilde{v}_k$ and $\tilde{a}_k$.

The attention weight $\alpha_{s,t}$ in the Seq2Seq-Attention model are calculated as

$$e_{k,t} = \mathbf{v}^T \tanh(\mathbf{W}_1 \mathbf{h}_k^{(dec)} + \mathbf{W}_2 \mathbf{h}_t^{(enc)}) \tag{4.6}$$

$$\alpha_{s,t} = \frac{\exp(e_{k,t})}{\sum_{t'=1}^{T} \exp(e_{k,t'})} \quad \text{(Softmax over timestamps)} \tag{4.7}$$

where, the context vector c_k is the weighted sum of the encoder output

$$c_k = \sum_{t=1}^{T} \alpha_{k,t} \mathbf{h}_t^{enc}$$

The context vector is then combined with the output of the decoder to reconstruct and impute the removed time series $\hat{y}_k$ at timestamps k

$$z_k = \text{Concat}(C_k, \mathbf{h}_k^{(dec)}) \tag{4.8}$$

$$\hat{y}_k = \mathbf{W}_o \mathbf{Z}_k + \mathbf{b}_o \tag{4.9}$$

Temporal Fusion Transformer

Temporal Fusion transformer (TFT), a multi-headed attention-based architecture is primarily used for time series forecasting [68]. It can also be extended to be used for the task of time series imputation, as it is describe in the rest of this section.

In order to learn the temporal patterns and features from the time series, the TFT utilizes recurrent layers to capture local short-term patterns and extends its ability to model long-term dependencies using a self-attention mechanism. Its architecture includes specialized components that help it select relevant features from the time series, while gating layers suppress irrelevant patterns. These design elements enable the model to focus on the most important information, leading to improved performance and more accurate results.

Since TFTs were originally designed for forecasting, they need to be modified to handle the task of imputation. To do so, some preprocessing of the data is required. For this study, the removed values of the time series were replaced with `NaN`, representing the `missing` values in the dataset. To train the TFT, the data marked with the `missing` values is used, and the MAPE loss value is used to make the TFT focus on imputing the removed speed and acceleration values.

TFTs require three type of inputs: Static covariates, time-varying known inputs, and time-varying observed inputs. In this study, since the static covariates (Vehicle id's) did not play a role in either forecasting or imputation, they are not sent as part of the input to the model. On the other hand, the observed lagged speed and acceleration values, and the observed (removed speed and acceleration) values were sent as inputs.

The gated residual network (GRN), which serves as the gating mechanism in the TFT architecture, enables the model to learn when to skip certain layers. This helps the TFT perform effective feature selection and reduces model complexity. Additionally, the gating mechanism helps prevent the model from overfitting the data. The output of the GRN is then passed through Softmax layers and undergoes non-linear transformations. This process of utilizing GRNs, softmax layer, and undergoing non-linear transformation is known as the Variable Selection Network, and allows the TFT to focus only on the most relevant variables in the time series by assigning them importance weights.

Next, the TFT uses an LSTM encoder-decoder architecture to process the time series. The encoder processes the known and observed inputs, while the decoder processes the future known inputs. This helps the TFT capture the temporal dependencies of the time series, which are enhanced by the multi-headed self-attention mechanism. The output of the decoder is passed to the multi-headed self-attention mechanism, enabling the model to learn the long-term dependencies of the time series.

Finally, using the combined output of encoder-decoder and self-attention mechanism, a final GRN layer uses the captured temporal pattern and dependencies, which enables it to impute the removed speed and acceleration values of the vehicles from the data.

4.1.3 State-Of-The-Art (SOTA) Methods

In addition to classic and hybrid deep learning models, as well as attention-based models for imputing speed and acceleration values, this study also employs several state-of-the-art (SOTA) models to impute the removed values in the time series. These SOTA imputation models include Bidirectional Recurrent Imputation for Time Series (BRITS) [69], Self-attention-based imputation for time series (SAITS) [70], and Bayesian Online Multivariate Time Series for imputation with functional decomposition (BayOTIDE) [71].

BRITS

BRITS, is a SOTA model that is used for time series imputation. It utilizes recurrent neural network (RNNs) for imputing missing values from the time series. The method proposed includes using a bidirectional recurrent dynamic system, where the missing values are treated as variables of the RNN graph. The variables are then updated using the RNN during backpropagation. The algorithm utilized by BRITS first attempts to find the hidden state decay, depending on how long it has been since the last value was observed. This can be mathematically represented as

$$\gamma_t = \exp(-\max(0, \mathbf{W}_\gamma \cdot \Delta_t + \mathbf{b}_\gamma)) \tag{4.10}$$

$$\tilde{h}_{t-1} = \gamma_t \odot \mathbf{h}_{t-1} \tag{4.11}$$

where, $\mathbf{W}_\gamma, \mathbf{b}_\gamma$ are the learnable parameters, Δ_t is the time gap vector between each feature since the last observation and h_{t-1} is the hidden state at the previous time step. The symbol $\odot$ represents element-wise multiplication.

Using, the temporal decay and hidden state, the model imputes the missing values using regression, and then combines it with observed values. This *completed* time series is then used to update the hidden state of the RNN, and then use the total loss value to make the updates. This is represented as

$$\hat{x}_t = \text{MLP}(\tilde{h}_{t-1}) \tag{4.12}$$

$$x_t^c = \mathbf{m}_t \odot \mathbf{x}_t + (1 - \mathbf{m}_t) \odot \hat{x}_t \tag{4.13}$$

$$\mathbf{h}_t = \text{RNN}(x_t^c, \tilde{h}_{t-1}) \tag{4.14}$$

where, $\hat{x}_t$ is the imputed value at timestamp t, $\mathbf{m}_t \in \{0, 1\}^D$ is a mask vector, and x_t^c is the complete time series using the imputed values. The values of the mask vector $\mathbf{m}_t$ depends whether $x_{t,d}$ is observed $m_{t,d} = 1$ or $x_{t,d}$ is missing $m_{t,d} = 0$.

For this study, the BRITS model was provided the speed and acceleration values, to impute the removed timestamps. Due to BRITS's ability to handle multivariate time series for imputation, no configuration were necessary and hence output produced by the model was comparable to the other model that were used to impute the removed data.

SAITS

Another SOTA model that was utilized in this study was the SAITS imputation model. SAITS uses a self-attention mechanism to impute the missing values in multivariate time series. It learns the missing values from a weighted combination of some diagonally-masked self attention that enable it to capture both the temporal dependencies of the time series as well as the feature correlation between the time steps. It also includes a weighted combination design that assist the diagonally-masked self attention by assigning weights using its attention map. This enables it to accurately impute missing data and while significantly reducing the time for training and imputing the missing values of the time series.

SAITS leverages a transformer-style self attention to capture the temporal dependencies across the time steps, and to do, it utilizes a complex model architecture. It first creates a mask vector/matrix similar to how it is created by the BRITS imputation model, however, SAITS in addition to using a mask, it also replaces the missing values in the time series with 0's or using mean imputation before time series is sent to the model.

It then uses two encoder blocks, each containing multi-headed self-attention layers. The first encoder block attempts to learn the temporal dependencies from the time series, and creates a hidden representation. These hidden representation are then passed down to the second encoder block, which refines these representations. The attention layers help SAITS capture the relationship between the timestamps and the variables of the time series.

The diagonal mask help in ensuring that each feature only attends to the other and not to itself, preventing any data leakage when imputing the missing values. SAITS, instead of using the just one of the encoders output, uses a combination of the results from both encoder to impute the missing values.

Due to SAITS ability to handle multivariate time series, no modification were made to its architecture. Both lagged speed and acceleration time series were sent as inputs to the model for it to provide the imputed values for both the speed and acceleration time series.

BayOTIDE

The last SOTA model that was used to impute the speed and acceleration values for this study was BayOTIDE. It conceptualizes a multivariate time series as a combination of weighted groups that contain low-rank temporal factors, which contain different patterns. It attempts to overcome the challenge of using local horizons to impute the missing values, and rather uses the global trends and periodic patterns to impute the missing data from a time series.

BayoTIDE treats the problem of imputing missing data as a probabilistic problem, where it assumes that the missing values are latent variables that are associated with uncertainty. In an attempt to impute the missing values, BayOTIDE uses functional decomposition such that the multivariate time series are assumed to be generated using a combination of smooth latent functions. Doing so, helps BayOTIDE capture the temporal features and trends from the time series, as well as the relationship between the variables of the multivariate time series.

BayOTIDE assumes that the observed time series can be modeled as

$$x_{t,d} \sim \mathcal{N}(\phi_t^T \mathbf{w}_d, \sigma^2) \tag{4.15}$$

where, ϕ_t is latent basis vector at time t, $w_d \in \mathbb{R}^r$ are the dimension-specific weights at time series variable d, and σ^2 is the observation noise variance.

After decomposing the time series, a Bayesian prior is placed over the coefficients $w_{d,k}$, such that $w_d \sim \mathcal{N}(0, \lambda^{-1} I)$, where λ^{-1} is the prior covariance matrix. The missing values at t are then imputed by computing the expected value under the predictive distribution

$$\hat{x}_{t,d}^{\text{missing}} = \mathbb{E}_q[\phi_t^T \mathbf{w}_d] \tag{4.16}$$

Because the model natively supports multivariate time series, its architecture also remained unchanged for imputing the final 30 timestamps of vehicle speed and acceleration. To ensure a fair comparison with the other model, the model was provided speed and acceleration through separate input branches; the network then produced imputed values for each variable over those last 30 time steps.

4.1.4 Imputation Results

Figure 4.1 and Figure 4.2 show the results of the various models that were used to impute the removed data of the time series.

The model that is part of the first-stage in the two-stage framework, needs to be able to impute both speed as well as acceleration data with the highest accuracy. However, as seen from Figure 4.1, and Figure 4.2, it can be concluded that a standalone TCN, and a TFT are not suitable for this purpose. Evaluation metrics show that the TFT yields the highest error rate for speed imputation, while the standalone TCN performs poorly when imputing acceleration values. Additionally, SOTA methods like BRITS and SAITS also have a considerably high error rate while imputing speed of the vehicles. Due to these reason, the TCN, TFT , SAITS and BRITS models are not considered as part of the first-stage in two-stage framework, and hence are not ideal to be used for the imputing both speed and acceleration values.

As illustrated in Figure 4.3 and Figure 4.4, both the CNN and CNN-LSTM models demonstrate the capability to impute the removed speed and acceleration data with high accuracy, producing values closely aligned with the ground truth. Specifically, the CNN

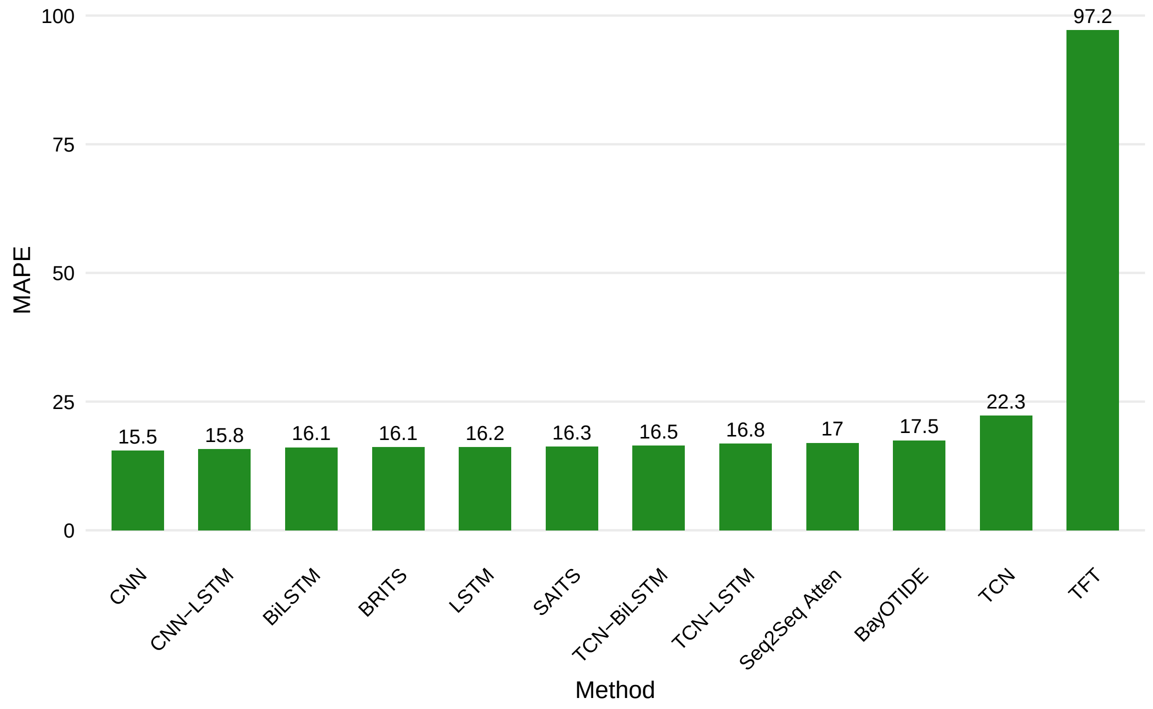

Figure 4.1: Imputation performance of classic and hybrid deep learing, state-of-the-art, and attention-based models on speed data

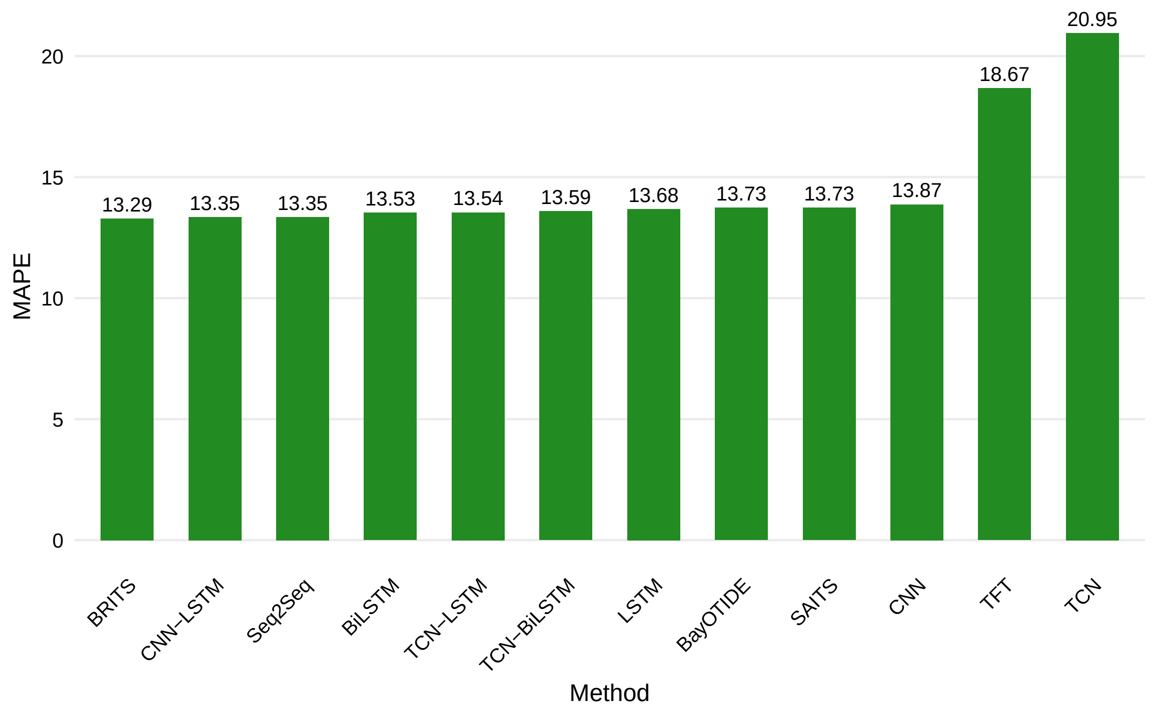

Figure 4.2: Imputation performance of classic and hybrid deep learing, state-of-the-art, and attention-based models on acceleration data

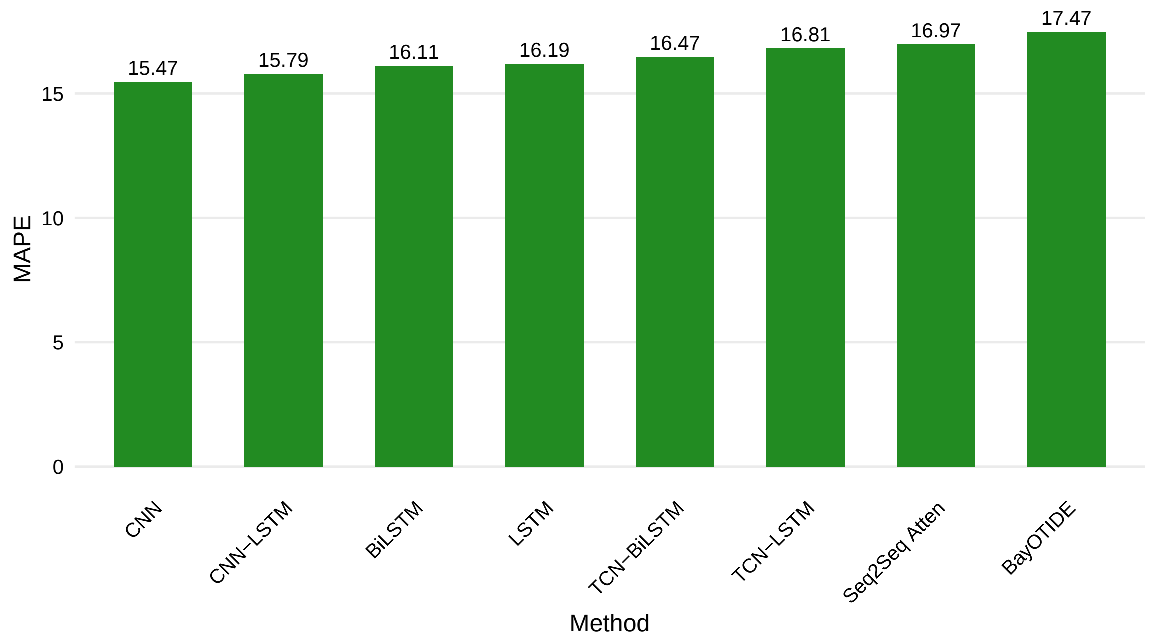

Figure 4.3: Imputation performance using classic and hybrid deep learing, state-of-the-art, and attention-based models on speed data that were able to produce considerably well with respect to both speed and acceleration

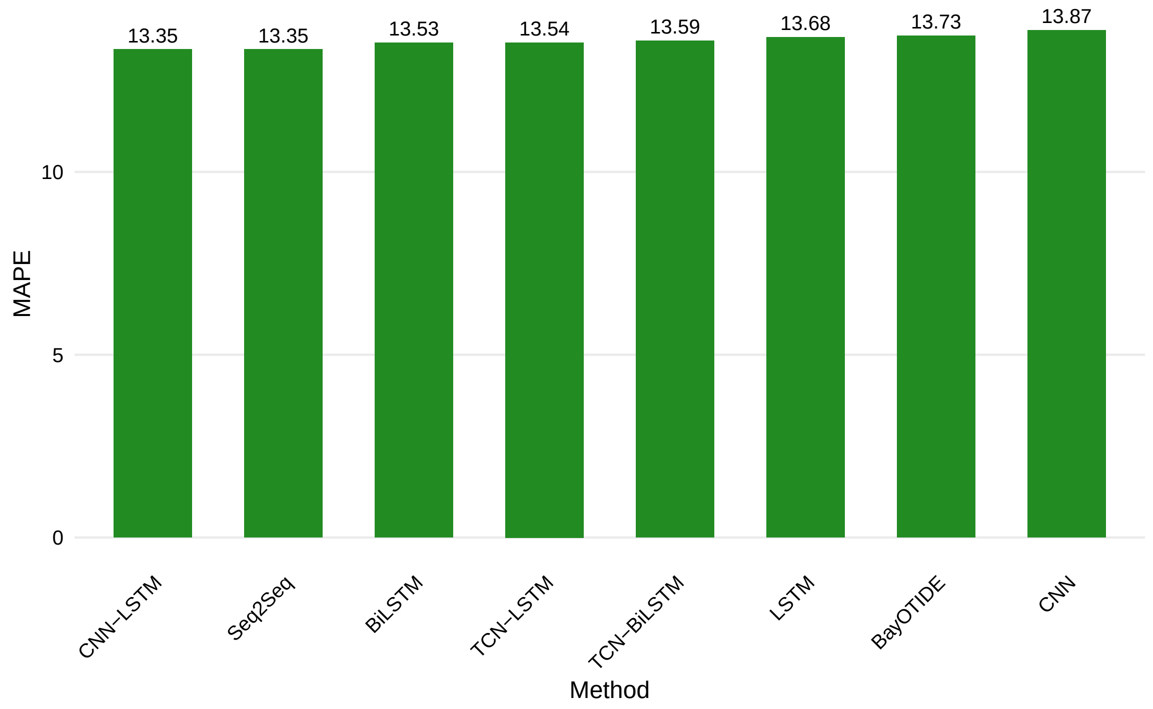

Figure 4.4: Imputation performance using classic and hybrid deep learing, state-of-the-art, and attention-based models on acceleration data that were able to produce considerably well with respect to both speed and acceleration

model achieved a Mean Absolute Percentage Error (MAPE) of 15.47 for speed imputation, and 13.87 for imputation of acceleration. In comparison, the CNN-LSTM model yielded a MAPE of 15.79 for speed, and 13.35 for acceleration. However, as further discussed in Section 4.2.4, the CNN model emerged as the more suitable choice for imputing the removed speed and acceleration values. Consequently, it was selected as the model for the first stage of the proposed framework.

4.2 Forecasting using Imputed Data

The second stage of the two-stage framework utilizes the imputed values generated in the first stage to forecast vehicle speeds at pedestrian crossings. Following the imputation process, the dataset expanded to include not only the original 170 speed and acceleration timestamps, but also an additional 30 imputed timestamps. This resulted in a complete time series of 200 timestamps for both speed and acceleration values per vehicle.

The combined dataset, comprising both original and imputed values, was then employed for forecasting. This approach was designed to address the challenges encountered by models when attempting to forecast without access to the final 3 seconds of data. By modelling a real-world scenario, the imputation phase aimed to fill temporal gaps using the CNN model, thereby enhancing the forecasting stage. It was hypothesized that the inclusion of these imputed values enabled the forecasting models to achieve improved accuracy and reduced error rates.

To confirm this hypothesis, the study conducted experiments using the combined dataset while using the hybrid TCN-LSTM-RFR model. In addition to the hybrid TCN-LSTM-RFR model, two attention-based models were also employed to comparatively evaluate the performance of different forecasting models in predicting vehicle speeds at pedestrian crossings during the second stage of the two-stage framework.

A detailed discussion of the various models that were used in the experiments are presented in the next sections. Furthermore, the result of their forecast value using the combined dataset is also presented and discussed in detail.

TCN-LSTM-RFR

As detailed in Section 3.5.2, the hybrid TCN-LSTM-RFR model comprises of a Temporal Convolutional Network (TCN) layer, followed by a Long Short-Term Memory (LSTM) layer, which is then connected to a fully connected layer. The output from this sequence is subsequently passed through a Random Forest Regressor (RFR) to produce the final forecast of vehicle speed over pedestrian crossings.

The primary distinction between the TCN-LSTM-RFR model described in Section 3.5.2 and the one used in this experiment lies in the input data. In the experiment done in Chapter 3, the model was provided with a dataset in which the final 3 seconds of speed and acceleration data were not present. In contrast, the model used in this stage of the framework received a complete dataset comprising both the original 170 speed and acceleration timestamps and an additional 30 imputed timestamps. These imputed values, generated during the first stage of the framework, allowed for a more complete time series input, where the imputed data modeled the speed and acceleration values of the vehicles prior to reaching the pedestrian crossings. These modeled imputed values replicated a real-world scenario of the vehicle as it approached pedestrian crossings.

The hybrid model maintained a multi-headed architecture, wherein speed and acceleration data were fed into the model through two distinct branches. Each branch incorporated both the original and the imputed timestamps, enabling the model to learn not only the underlying dynamics of the original time series but also to leverage the imputed series, which preserved the same trends and seasonal patterns. This dual input approach enhanced the model's capacity to extract richer feature representations and to capture long-term dependencies. Consequently, the model could make accurate predictions three seconds prior to the vehicle reaching pedestrian crossings and not just milliseconds before reaching the pedestrian crossings.

The full two-stage process to forecast the speed of the vehicle over the pedestrian crossings using the CNN imputation model to first impute the last three seconds of speed and acceleration data, followed by the hybrid TCN-LSTM-RFR forecasting model for a input speed $\mathbf{X}_s = \{x_{s1}, x_{s2}, x_{s3}, \dots, x_{sT}\}$ and acceleration $\mathbf{X}_a = \{x_{a1}, x_{a2}, x_{a3}, \dots, x_{aT}\}$ can be mathematically expressed and formulated as

$$\hat{\mathbf{Y}}_s = \mathcal{F}_{FC}(\mathcal{F}_{\text{CNN}}(X_s; \Theta_{\text{CNN}})) = \mathcal{F}_{FC}(\text{Conv1D}(\mathbf{X}_s; W_{\text{CNN}}, b_{\text{CNN}})) \tag{4.17}$$

$$\hat{\mathbf{Y}}_a = \mathcal{F}_{FC}(\mathcal{F}_{\text{CNN}}(X_a; \Theta_{\text{CNN}})) = \mathcal{F}_{FC}(\text{Conv1D}(\mathbf{X}_a; W_{\text{CNN}}, b_{\text{CNN}})) \tag{4.18}$$

$$\mathbf{z}_s = \text{LSTM}(\text{TCN}_s(\mathbf{X}_s \oplus \hat{\mathbf{Y}}_s)) \tag{4.19}$$

$$\mathbf{z}_a = \text{LSTM}(\text{TCN}_a(\mathbf{X}_a \oplus \hat{\mathbf{Y}}_a)) \tag{4.20}$$

$$\mathbf{z} = [\mathbf{z}_s; \mathbf{z}_a] \in \mathbb{R}^d \tag{4.21}$$

$$\mathbf{h}_{\text{dense}} = \text{ReLU}(\mathbf{W}_d \cdot \mathbf{z} + \mathbf{b}_d) \tag{4.22}$$

$$h_{\text{drop}} = \text{Dropout}(\mathbf{h}_{\text{dense}}) \tag{4.23}$$

$$\hat{y} = \mathcal{F}(h_{\text{dense}}) \tag{4.24}$$

$$\hat{y} = \frac{1}{M} \sum_{i=1}^{M} T_i(h_{\text{dense}}) \tag{4.25}$$

Figure 4.5 illustrates the two-stage framework with CNN imputation model, and TCN-LSTM-RFR forecasting model. Stage 1, that is represented by the yellow box, presents the imputation stage of the framework. In this stage the lagged speed and acceleration timestamps of the vehicles are passed as inputs to the CNN imputation model, that simultaneous outputs the imputed speed and acceleration timestamps for each vehicle. These imputed speed and acceleration timestamps are coloured in red and purple respectively in the figure. These imputed timestamps are combined with the original speed and acceleration values and are provided as input to Stage 2 (highlighted in blue box), which is responsible for forecasting the speed of the vehicle over the pedestrian crossings. In Stage 2, the combined input are passed to the TCN model, after which, it is passed to the LSTM, and then through a fully connected layer. Finally, the output of the fully connected layer is passed to a Random Forest Regressor to forecast the speed of the vehicles over the pedestrian crossings.

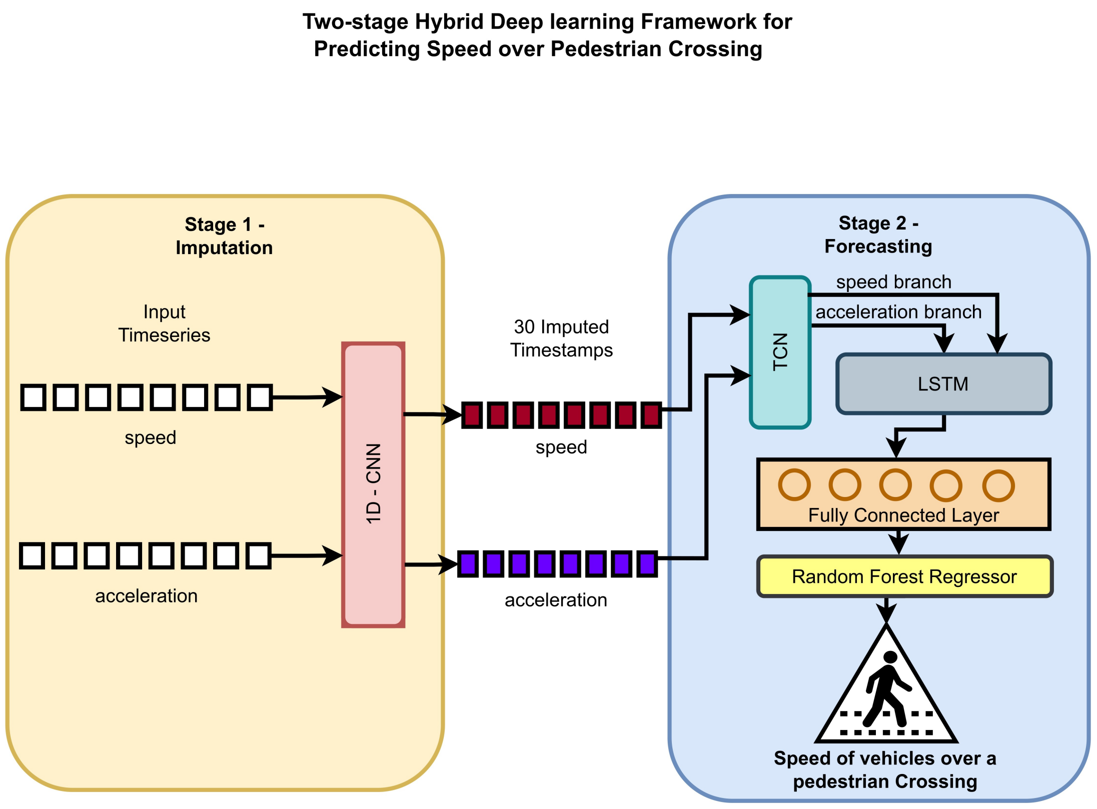

Figure 4.5: Two-stage Framework with CNN imputation model, and TCN-LSTM-RFR Forecasting model

4.2.1 Attention-based Methods for Forecasting

Temporal Fusion Transformers

In addition to the hybrid TCN-LSTM-RFR model, the combined- original values and imputed values- dataset was also evaluated using a Temporal Fusion Transformer (TFT). As mentioned in Section 4.1.2 TFTs are a type of transformer model that are designed mainly for the purpose of time series forecasting. Due to the augmentation of the dataset that included the imputed three seconds of timestamps, this study utilized a TFT to form a comparison between the hybrid TCN-LSTM-RFR model, and evaluate the effectiveness of imputation of the data on the final forecast value.To perform an accurate comparison, the TCN-LSTM portion of the hybrid deep learning model is replaced with a TFT, to form a hybrid Temporal Fusion Transformer and a Random Forest Regressor (TFT-RFR) model.

The architecture of the TFT model is similar to the one used for the task of imputing the removed data from the time series. The input to the TFT remained multi-headed, i.e. lagged speed and acceleration values along with the imputed values from the CNN imputation model were sent as inputs to the TFT. These inputs are passed to a variable selection layer where the crucial and most representative features of the time series are extracted, which are then forward to the LSTM encoder-decoder layer. Using the representation created from the encoder layer, the multi-headed attention layer is able to add weights to the important time stamps, which are then used by the GRN to forecast the values of the time series.

The TFTs tasked with time series forecasting for a multivariate time series with the original as well as imputed data $\mathbf{X}_s = \{x_1, x_2, x_3, ..., x_T\}$ combined with a Random Forest Model, can be mathematically expressed as

$$\hat{\mathbf{Y}} = \mathcal{F}_{FC}(\mathcal{F}_{\text{CNN}}(X; \Theta_{\text{CNN}})) = \mathcal{F}_{FC}(\text{Conv1D}(\mathbf{X}; W_{\text{CNN}}, b_{\text{CNN}})) \tag{4.26}$$

$$\mathbf{H}_0 = \text{GRN}(\mathbf{X} \oplus \hat{\mathbf{Y}}; \Theta_{\text{GRN}}) \tag{4.27}$$

$$\mathbf{H}_{\text{enc}} = \text{LSTM}_{enc}(\mathbf{H}_0; \Theta_{\text{enc}}) = [h_1^{enc}, \ldots, h_T^{enc}] \tag{4.28}$$

$$\mathbf{H}_{\text{attn}} = \text{MHAttn}(\mathbf{H}_{\text{enc}}; \Theta_{\text{attn}}) = \sum_{i=1}^{T} \alpha_{ti} \cdot \mathbf{v}_i \tag{4.29}$$

$$\hat{\mathbf{Y}}_{\text{out}} = \mathcal{F}_{\text{RFR}}(\text{GRN}(\mathbf{H}_{\text{attn}} \oplus \mathbf{H}_0)) \tag{4.30}$$

$$\tag{4.31}$$

where, $\hat{\mathbf{Y}}_{\text{out}}$ is the forecast speed value of the vehicle over the pedestrian crossings, α_{ti} is the softmax function that helps the multi-headed attention layer apply weights to the appropriate time stamps using the help of the value vector $\mathbf{v}_i$.

Hybrid Sequence to Sequence Attention and RFR

Along with the hybrid TCN-LSTM-RFR, and the TFT-RFR model, another attention-based model-called Sequence to Sequence Attention (Seq2Seq-Attention)- was used to compare the forecast speed values of the vehicles over the pedestrian crossings. Similar to the Seq2Seq-Attention model used for imputing the removed data in Section 4.1.3, this model was used to forecast the speed of the vehicles over the pedestrian crossings. The key difference between the model used for imputation and the one used for forecasting is that its architecture was modified to pass the output of the fully connected layer through a Random Forest Regressor to predict the speed of the vehicle.

Figure 4.6 illustrates the architecture of the Seq2Seq-Attention model that is used to forecast the speed of the vehicles. The grey dotted borders envelop the sequence to sequence (Seq2Seq) model architecture, where several LSTM layers encode and decode the input time series. On the left side of the image, the input time stamps highlighted in yellow are passed onto the LSTM layers (highlighted in blue) to create latent representations or hidden states of the time series. The hidden representation are created not only using the input times stamps, but also by combining the hidden state produced from the previous time stamp. These hidden states are then passed onto the attention mechanism that calculates the attention weights, and using these encoded hidden states produces a context vector. The context vector is a weighted summary of the inputs, and helps the model identify the important timestamps within the time series.

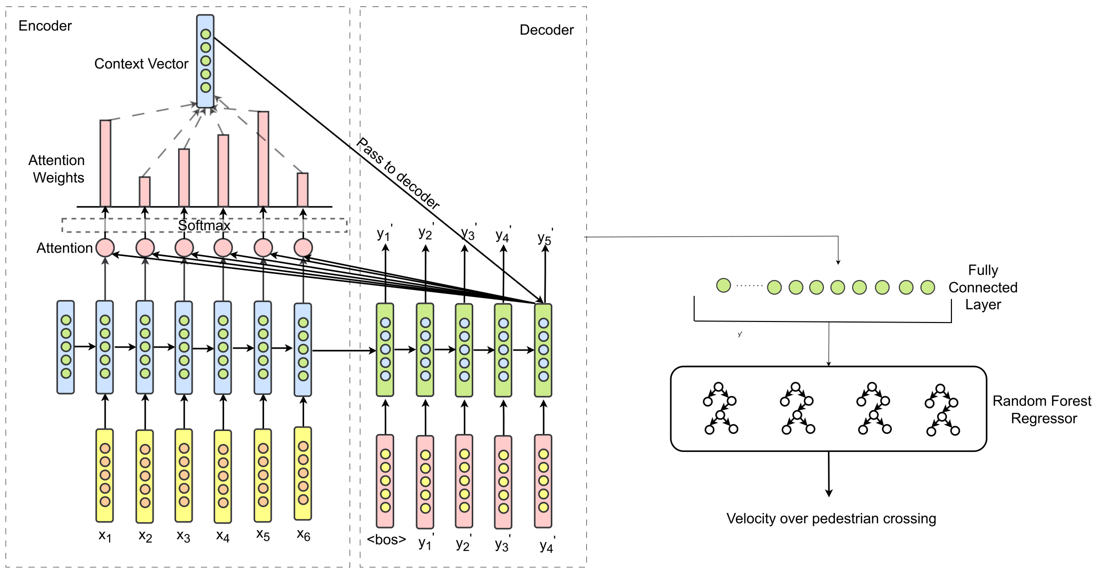

Figure 4.6: Sequence to Sequence Attention Architecture.

The context vector produced from the hidden states after being passed through the attention mechanism, is passed onto the decoder LSTM layers (highlighted in green) in an attempt to reconstruct the time series and decode the encoded input time series. The first decoder layer, along with the context vector is also provided the output of the last encoder layer. Following on, the output at each time step is passed as an input to the next decoder layer. Finally, when all the time stamps have been decoded, the output of the Seq2Seq model is then passed through a fully connected layer, where the output of the speed and acceleration branches are combined together before being passed onto the Random Forest (RFR) model to output the predicted speed of the vehicles over pedestrian crossings.

4.2.2 Forecasting Results

The evaluation of the speed forecast value depends on how well the models are able to analyse and extract features and representations from the original as well as imputed data of the time series. Furthermore, the feature extracted from the imputed time series would only be adequate and influential if the imputed data is interpreted correctly by the model. For this reason, along with the CNN imputation model, other model were also utilized to form a benchmark for the models used for imputation. Since the hybrid TCN-LSTM-RFR model was able to produce the best forecast value without the imputed data (illustrated in Table 3.3), it was chosen to evaluate the effectiveness of the imputation models.

Table 4.1 contains the forecasted speed value using the hybrid TCN-LSTM-RFR model as the forecaster, while using different models for imputation.

Table 4.1: MAPE values of forecasted speed using imputations models along with hybrid TCN-LSTM-RFR as forecaster

Imputation Models	**MAPE**
LSTM	11.82
CNN	**10.89**
CNN-LSTM	12.42
BiLSTM	11.31
TCN	13.16
TCN-LSTM	11.63
TCN-BiLSTM	11.43
Seq2Seq	12.10
TFT	18.38

Based on the results, it can be concluded that the CNN-based imputation model outperforms the other approaches in reconstructing the removed speed and acceleration data. It not only yields the lowest error rates for the imputed values but also demonstrates superior performance in forecasting vehicle speeds over the pedestrian crossings.

Furthermore, to emphasize the importance of forecasting speed following imputation, the models were extended to impute not only the last three seconds of removed data but also the vehicle speed specifically over the pedestrian crossings. This was achieved by treating the timestamps corresponding to the pedestrian crossings as removed. The performance of these augmented imputation models is presented in Table 4.2

Table 4.2: MAPE values of forecasted speed using augmented imputations models

Imputation Models	**MAPE**
BayOTIDE	16.97
TCN-LSTM	12.27
CNN	15.78
LSTM	15.34
CNN-LSTM	13.26
BiLSTM	12.89
TCN	19.62
TCN-BiLSTM	12.77

The results clearly indicate that a dedicated forecasting model is necessary for predicting values at the pedestrian crossings. The speed at the pedestrian crossings should not be treated as removed data to be imputed using conventional imputation techniques, as this would not yield reliable outcomes.

Furthermore, the proposed hybrid TCN-LSTM-RFR model was evaluated against two attention-based forecasting models: a Sequence-to-Sequence model with an attention mechanism coupled with a Random Forest Regressor (Seq2Seq-Attention-RF), and a Temporal Fusion Transformer combined with a Random Forest Regressor (TFT-RFR).

Table 4.3: MAPE value of forecasted speed using hybrid TCN-LSTM-RFR and attention-based models after imputation of removed data

Forecasting Models	**MAPE**
TCN-LSTM-RFR	10.89
TFT-RFR	12.67
Seq2Seq-Attention-RFR	**10.10**

Table 4.3 presents the MAPE values for the forecasted vehicle speeds over the pedestrian crossings using the hybrid TCN-LSTM-RFR model and the attention-based Seq2Seq models, following imputation of the removed data. The results indicate that the imputation process enhanced the forecasting performance of the hybrid model. Specifically, the Mean Absolute Percentage Error (MAPE) of the TCN-LSTM-RFR model was reduced by 0.25%, demonstrating that the post-imputation model outperforms its pre-imputation counterpart.

Overall, the two-stage approach- imputation followed by forecasting -proves effective in reducing prediction error.

Moreover, the results in Table 4.3 show that the Sequence-to-Sequence model with an Attention mechanism, followed by a Random Forest Regressor (Seq2Seq-Attention-RFR), outperformed the hybrid TCN-LSTM-RFR model by 0.79%, achieving a lower MAPE of 10.10%. This improvement is likely attributed to the attention mechanism's ability to assign greater significance to relevant time steps, thereby enhancing the model's capacity to interpret and extract meaningful features from the time series data more effectively than the hybrid TCN-LSTM-RFR model.

4.3 Reinforment Learning

Reinforcement Learning (RL) is a branch of machine learning focused on decision-making through autonomous agents. Unlike traditional supervised learning, which relies on labeled datasets for training, RL agents learn through interaction with their environment. Rather than being trained on a fixed dataset, an RL agent gradually improves its performance by taking actions, observing the outcomes, and using this feedback to learn strategies that maximize a predefined goal or reward.

RL has been used in field of robotics, and natural language processing. It can also be extended and used for complex tasks and simulate human behaviour such as automated driving. In this study, RL agents were used to improve the forecasting models performance by tuning the hyperparameters of the hybrid TCN-LSTM-RFR model. Specifically, the dilation component of the Temporal Convolutional Network (TCN) model was configured using a RL agent to select the appropriate dilation rate in each layer of the TCN model.

In the following sections the various aspects of the reinforcement learning will be explained. Additionally, the Proximal Policy Optimizer algorithm that was used to train the RL agent will also be discussed in the following chapter.

4.3.1 Methods

State, Action, Reward

A reinforcement learning model is typically formalized as a Markov Decision Process (MDP), which is a mathematical framework used to describe decision-making in situations where outcomes are influenced both by randomness and by the decisions of an agent. The agent is the entity responsible for making these decisions, and the MDP formalizes the interaction between the agent and its surrounding environment. At any given time step t, the environment is represented by a state, which captures all relevant information about the environment at that moment. The state is a critical component in reinforcement learning, as it provides the agent with the necessary context to make informed decisions, rather than choosing actions at random.

The set of all possible decisions available to the agent in a given state is referred to as the set of actions. At each time step t, the agent observes the current state and evaluates which action to take based on this information. The chosen action affects the environment, resulting in a transition to a new state. This interaction, where the agent continually observes the state, selects actions, and experiences the consequences, forms the basis of the learning process in reinforcement learning.

The agent selects the most appropriate action by evaluating the potential outcomes of each action in terms of the reward it may produce. A reward is a form of feedback provided by the environment in response to the action chosen by the agent. The primary objective of a reinforcement learning agent is to maximize the cumulative reward over time. This process enables the agent to improve its decision-making strategy by favoring actions that yield higher rewards. Without a reward signal, the agent would lack guidance for selecting effective actions, making it impossible for the agent to learn an optimal policy or achieve its ultimate goal.

The MDP $\mathcal{M}$ uses these component to define the interaction between the agent and it environment. A MDP can be formal expressed as

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle \tag{4.32}$$

where $\mathcal{S}$ represents the set of all possible states s that the environment can occupy, while $\mathcal{A}$ denotes the set of all possible actions a that the agent can take. The state transition probability, $P(s'|s, a)$, defines the likelihood of the environment transitioning to state s' when the agent executes action a in state s. The reward function, $R(s, a)$, provides an immediate reward to the agent based on the action a taken in state s. The parameter $\gamma \in [0, 1]$ is the discount factor, which determines the extent to which the agent values future rewards relative to immediate rewards.

A lower value of γ, approaching 0, makes the agent more focused on immediate rewards, resulting in a greedy behavior where the agent selects actions that maximize short-term gains. This often discourages exploration, as the agent prioritizes transitioning to states that yield immediate high rewards. Conversely, a higher value of γ encourages the agent to value future rewards more heavily. This fosters exploratory behavior, allowing the agent to occasionally select actions that may yield lower immediate rewards but can lead to greater cumulative rewards in the long run. The underlying intuition is that an agent with a higher γ is better equipped to learn and navigate complex environments by balancing exploration and exploitation, ultimately achieving higher overall returns compared to a purely greedy agent.

Proximal Policy Optimizer

The Proximity Policy Optimization is a reinforcement learning algorithm that is part of the policy gradient family, and alternates between sampling data from the environment through interaction, and optimizing an objective function using stochastic gradient ascent [72]. PPO is designed to achieve stable and efficient policy updates by preventing excessively large changes to the policy, which can destabilize learning.

In reinforcement learning, one common approach to solving sequential decision-making problems is the use of policy-based methods. These methods involve directly optimizing a parameterized policy $\pi(a|s;\theta)$, which defines a probability distribution over actions $a \in \mathcal{A}$ given a state $s \in \mathcal{S}$. The parameters θ are typically optimized to maximize the expected cumulative reward over time.

In complex environment, stochastic polices $\pi(a|s;\theta)$ are modeled using deep neural networks. By using these neural networks, the policy is able to approximate high-dimensional, non-linear mapping from states to action distributions, enabling the agent to learn sophisticated behaviors in diverse and dynamic settings. Policy gradient methods, including PPO, aim to optimize the parameters θ of these stochastic polices, by computing the gradient of the expected rewards using the policies, and perform gradient ascent. This allows the reinforcement learning agent to improve its performance by iteratively refining its policy based on the responses it receives from the environment. The objective function that is used to maximize the expected reward can be expressed as:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] \tag{4.33}$$

where, $\tau = (s_0, a_0, s_1, a_1, \dots)$ is a trajectory, and $[R(\tau)]$ is the total reward of the trajectory.

Furthermore, the policy gradient theorem defined as the gradient of the expected return can be formally defined as

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) \cdot \hat{A}(s,a)] \tag{4.34}$$

where, $\hat{A}(s,a)$ is the advantage function that estimates how much better an action a is compared to the average. The Policy Gradient Theorem illustrates how much the policy parameters need to be changed in order to increase the probability of a good action.

The core purpose of PPO is to improve the traditional Policy gradient methods such as Trust Region Policy Optimization (TRPO). TRPO improves the stability and reliability of the RL agent training constraining the size of each policy update. It does so by maximizing a surrogate objective while enforcing a constrain on how much the new policy deviates from the old one. PPO on the other hand, improves TRPO by avoiding large updates to the policy that can potentially destabilize the learning of the RL agent. By updating the policy θ, PPO uses gradient ascent along with a clipping mechanism to limit how much the policy updates and changes at each step. The objective function of a PPO can be defined as

$$L^{\text{CLIP}}(\theta) = \mathbb{E}[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{4.35}$$

where, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies, $\hat{A}_t$ is the advantage estimate at time t, ϵ is the small hyperparameter (e.g 0.1 or 0.2) that sets the trust regions, and clip$(\cdot)$ is the function that prevents the large updates to the policy. The clipping in the PPO objective function ensures that if the new policy is very different from the old policy, then the update to the policy is suppressed. It utilizes an actor-critic architecture, where the actor or the policy learns the parameterized policy to select the appropriate actions, and a critic that estimates the value function to provide feedback on the quality of the actions.

Proposed Method for Dynamic Dilation

The proposed method includes training a reinforcement learning agent that uses a PPO to learn to select the optimal dilation rate in each of the five layers of the Temporal Convolutional Network (TCN) model. For this, the agent utilizes the TCN model in its environment and uses the index of the current layer of the TCN model, in addition to the previous layers dilation rate, and the cumulative receptive filed as part of its state.

Furthermore, the reinforcement learning (RL) agent is allowed to select a dilation rate of either $[0, 1, 2, 3, 4]$, corresponding to effective dilation rate of $[1, 2, 4, 8, 16]$ (exponentially scaled by a factor of 2) for each layer of a TCN model. These dilation rates are part of the agent's action space within its environment. After the TCN model is configured based on the agent's choices, it is trained and validated using the specified training and validation datasets. The environment provides a reward to the RL agent, defined as the negative value of the validation loss, which is calculated as the difference between the predicted and actual speed values. This reward structure provides the agent with an incentive to minimize the validation loss, and encourage it to learn action selections that results in lower prediction errors.

Figure 4.7 illustrates the architecture of the reinforcement learning (RL) architecture employed in this study. The region highlighted in blue represents components of the RL agent's environment. This includes the TCN-LSTM-RFR model, which is configured with various dilation rates. The RL agent learns to configure the TCN with optimal dilation rates, which are later integrated into a hybrid TCN-LSTM-RFR model which is used for forecasting.

The TCN model, once trained and validated on the designated datasets, produces a reward, defined as the negative Mean Absolute Percentage Error (MAPE) validation loss, which is returned to the multi-layer perceptron (MLP) network used by the RL agent. The yellow-highlighted section depicts the RL agent itself, including its inputs, outputs, and the internal MLP network. The green-highlighted section shows the state representation of the environment, comprising the set of features provided to the RL agent.

Based on the current state, the MLP produces two outputs: a probability distribution over possible actions (i.e., dilation rates for each TCN layer) and an estimate of the expected reward (negative validation loss, i.e. negative MAPE value). Using these outputs, the RL agent selects dilation rates to configure the TCN model, which is then evaluated to generate a new reward. This process is repeated over 10,000 episodes to enable the RL agent to learn the optimal dilation rate configuration for each TCN layer. Once training is complete, the optimized TCN is combined with an LSTM and a Random Forest Regressor (RFR) to form the final hybrid TCN-LSTM-RFR model.

4.3.2 Results using Reinforce-learned TCN-LSTM-RFR

Along with the proposed reinforced-learned dynamic dilation method, experiments were conducted in which the RL agent's state representation was extended to include a gradient decent metric, alongside the current layer index, the previous layer's dilation rate, and the

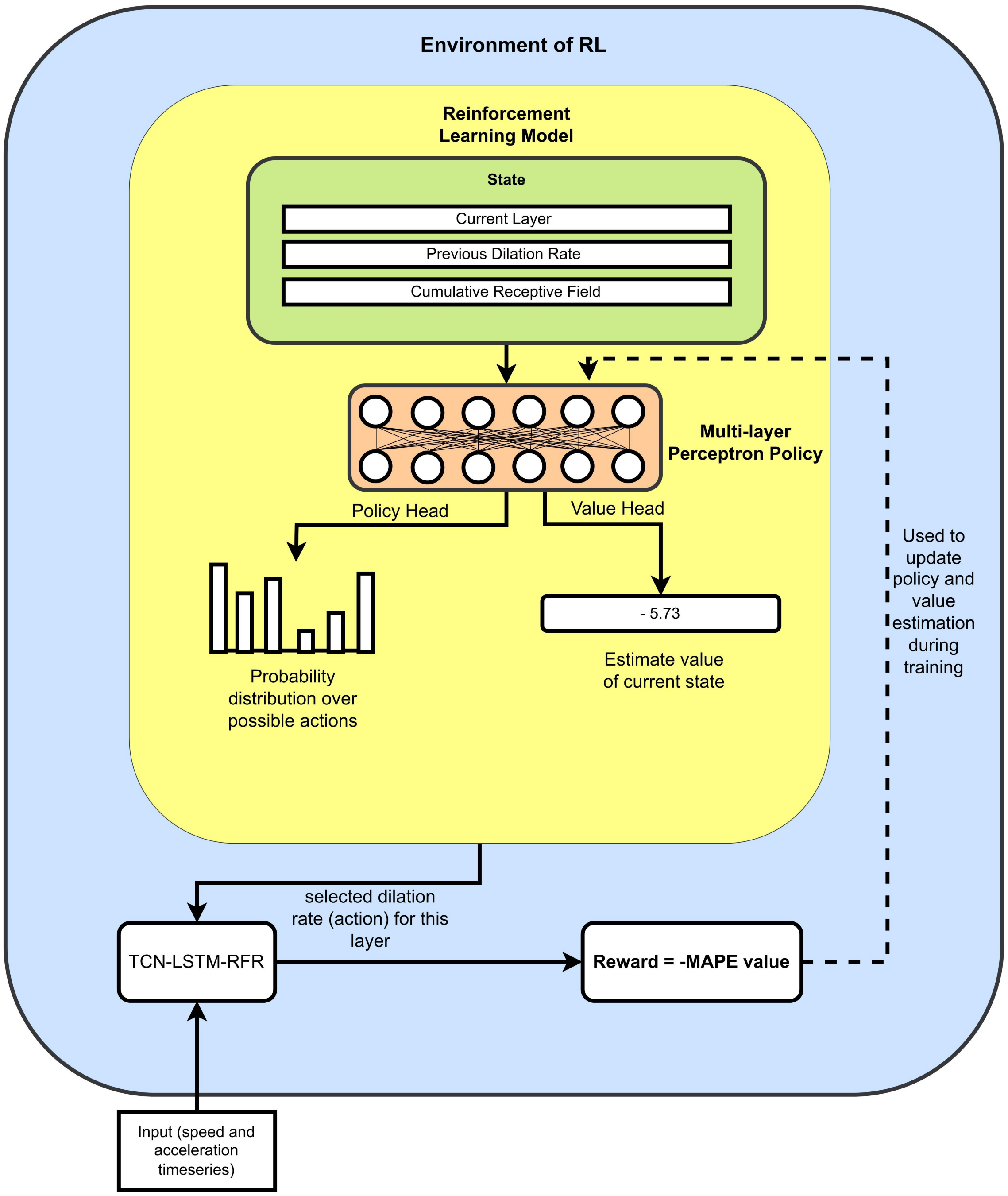
Environment of RL
Reinforcement Learning Model
State
Current Layer
Previous Dilation Rate
Cumulative Receptive Field
Multi-layer Perceptron Policy
Policy Head
Value Head
Probability distribution over possible actions
- 5.73
Estimate value of current state
Used to update policy and value estimation during training
selected dilation rate (action) for this layer
TCN-LSTM-RFR
Reward = -MAPE value
Input (speed and acceleration timeseries)

Figure 4.7: Reinforement Learning architecture for selecting optimal dilation rate for each layer of the TCN model

cumulative receptive field. However, this extended state configuration resulted in suboptimal performance in terms of the final speed predicted over the pedestrian crossings by the hybrid TCN-LSTM-RFR model. In contrast, the model trained using the original, simpler state representation yielded better results.

Furthermore, to evaluate the effectiveness of the proposed RL model architecture, it was compared against another RL model where, instead of allowing the RL model to select a predefined dilation rate for each layer whose dilation rate increased exponentially by a factor of 2, the RL agent was allowed to select a continuous value between 1 and 16 for the dilation rate in each layer of the TCN model. Moreover, another experiment was also conducted where RL model was trained to not only dynamically select the dilation rate in each layer of the TCN, but also find an optimal number of TCN layer (between 1 and 5). The experiments concluded that the fixed 5-layer with pre-defined dilation rate for each layer of TCN model continued to outperform the models with continuous dilation rate values, and varying number of TCN layers.

Table 4.4: Result for Forecasting Using Dynamic Dilation Rate in TCN

TCN Models	**MAPE**
Static Dilation	11.14
Dynamic Dilation – using RL (CNN Policy)	11.02
Dynamic Dilation – using Gradient Decent as Part of State	11.45
Dynamic Dilation – using RL (MLP Policy)	**10.95**
Dynamic Dilation – using RL (MLP Policy): Continuous Values	11.01
Dynamic Dilation – using RL (MLP Ploicy): Soft-Actor Critic	12.76

The results of the experiments conducted using the data without the imputed data are listed in Table 4.4. It can be concluded from the experiments that forecasting accuracy of the hybrid TCN-LSTM-RFR model to forecast the speed of the vehicles over the pedestrian crossings outperform the static dilation model in most cases. The TCN model with the 5 fixed TCN layer with a MLP policy, and PPO outperformed the TCN model created using static dilation, where the dilation increased exponentially by a factor of 2, by 0.19%. In addition to using the PPO, the Soft-Actor Critic (SAC) optimizer was also used to evaluate with the performance of PPO approach. However, the RL model with the PPO continued to outperform the RL model with a SAC optimizer as it was only able to produce a forecast value with a 12.76% MAPE value. Similarly, the RL agent with a CNN policy as the neural network instead of the MLP neural network, was also outperformed by the proposed RL model architecture by 0.07%.

Table 4.5: Results for Static and Dynamic Dilation TCN Models and Other Forecasting Approaches on Datasets With and Without Imputation

Models	**With OR Without Last 3 seconds of Imputed Data Values**	**MAPE**
TCN-LSTM-RFR (STATIC DILATION)	WITHOUT	11.14
TCN-LSTM-RFR (DYNAMIC DILATION)	WITHOUT	10.95
Seq2Seq Attention with RFR	WITH	10.10
TCN-LSTM-RFR (DYNAMIC DILATION)	**WITH**	**9.85**

The reinforcement learning model was also evaluated using the original dataset combined with imputed values generated by the CNN imputation model. The results of this experiment are presented in Table 4.5. These findings demonstrate that the hybrid TCN-LSTM-RFR model incorporating a TCN with dynamically selected dilation rates via a reinforcement learning agent and utilizing the imputed data from the CNN imputation model outperformed all other forecasting models examined in this thesis for predicting vehicle speeds at pedestrian crossings. Specifically, it achieved a MAPE of 9.85%, representing a notable reduction of 1.29% compared to the hybrid TCN-LSTM-RFR model without dynamic dilation and imputation. Furthermore, it surpassed the TCN-LSTM-RFR model with dynamic dilation but with the inclusion of the imputed removed values by 1.10%, and outperformed the Seq2Seq-Attention model with RFR (with imputed data) by 0.25%. Prior to this experiment, the sequence-to-sequence attention with Random Forest Regressor (Seq2Seq-Attention-RFR) model with imputed data had been outperforming all the other forecasting models that were employed in this thesis (as mentioned in the section 4.2.2). Nevertheless, the introduction of the reinforced-learned TCN model with imputed data was able to show the superiority of dynamic dilation by outperforming the Seq2Seq-Attention-RFR model.

Moreover, to further validate the results, the models listed in Table 4.5 are evaluated using five random train-test splits, with 80% of the data used for training and 20% for evaluation in each split. The mean MAPE values across all five runs, along with their corresponding 95% confidence intervals, are presented in Table 4.6

Table 4.6: Mean MAPE Results with 95% Confidence Intervals for Static and Dynamic Dilation TCN Models and Other Forecasting Approaches on Datasets With and Without Imputation (5 train-test splits)

Models	**With OR Without Last 3 seconds of Imputed Data Values**	**MAPE**	**Confidence Interval**
TCN-LSTM-RFR (STATIC DILATION)	WITHOUT	11.22	[9.19, 13.24]
TCN-LSTM-RFR (DYNAMIC DILATION)	WITHOUT	10.85	[9.64, 12.07]
Seq2Seq Attention with RFR	WITH	10.23	[9.28, 11.17]
TCN-LSTM-RFR (DYNAMIC DILATION)	**WITH**	**9.82**	**[9.23, 10.41]**

4.4 Summary

This chapter presented a novel two-stage framework to predict the speed of left-turning vehicles at an intersection over a pedestrian crossing. The framework was created to first impute the removed data from the time series, followed by forecasting the speed of the vehicles. In the first stage of the framework, to impute the last three seconds of the removed speed and acceleration data, various model for imputation were utilized. These included using deep learning models, state-of-the-art (SOTA) as well as some attention-based imputation models. Using these various model for imputation, several experiments were conducted which revealed that a classical CNN imputation model was able to impute the speed and acceleration data with the lowest error with a MAPE of 10.89%. The CNN model used for imputation had a multi-headed input, meaning that the lagged speed and acceleration data were sent as input as two separate branches. With the help of the input data, the

CNN model is was able to interpret the temporal patterns within the data, and impute the removed values. The CNN model simultaneously predicted the imputed values for both, the removed speed and acceleration data.

The second stage in the framework used the imputed data from the first stage of the framework to forecast the speed of the vehicles using the hybrid TCN-LSTM-RFR model described in Chapter 3. The combination of the imputed data along with the lagged speed and acceleration data were sent as multi-headed inputs to the hybrid TCN-LSTM-RFR model to forecast the speed of the vehicles. The result suggested that imputing the removed values assisted the TCN-LSTM-RFR to improve its forecasting accuracy by 0.25%, with a MAPE value of 10.89%. To further evaluate the results of forecasting, the combination of the imputed data, along with the lagged speed and acceleration values were used with other attention-based forecasting methods, namely Temporal Fusion Transformer (TFT), and Sequence-to-Sequence Attention model with a Random Forest Regressor (Seq2Seq-Attention-RFR). Based on the experiments conducted, the Seq2Seq-Attention-RFR model was able to outperform the TCN-LSTM-RFR model by 0.79%, with a MAPE value of 10.10%.

This chapter also introduced a novel reinforcement learning (RL) approach for dynamically selecting the dilation rate hyperparameter for each layer of the TCN component within the hybrid TCN-LSTM-RFR model. The RL agent interacted with the TCN-LSTM-RFR model as its environment and employed the Proximal Policy Optimization (PPO) algorithm to learn optimal dilation rates for the TCN layers. The environment's state representation consisted of the current layer index, the dilation rates of preceding layers, and the cumulative receptive field. The agent received a reward based on the negative validation loss (i.e., negative MAPE), which guided its learning process in choosing appropriate actions. The available action space for the agent included dilation rates of [1,2,4,8,16], and it used a Multi-Layer Perceptron (MLP) policy to learn and predict suitable dilation rates for each layer. Once the optimal dilation rates were selected, the updated TCN model was integrated with the LSTM and RFR components to form a hybrid model with dynamically selected dilation rates. Upon evaluation, this dynamic hybrid model achieved a MAPE of 10.95%, outperforming the static-dilation version by 0.19%. Moreover, when the dynamically dilated TCN-LSTM-RFR model was trained using the combined original and imputed data, it achieved the best forecasting performance across all models in this thesis, with a MAPE of 9.85%.

5 Conclusion

This thesis introduces a two-stage hybrid deep learning framework to forecast the speed of vehicles over the pedestrian crossings using speed and acceleration values. In the first stage, speed and acceleration values of the vehicles present in the dataset are imputed, which are then passed onto the second stage. In the second stage, the combined-original as well as imputed-data are provided as input to the hybrid deep learning framework in order to forecast the speed of the vehicles over the pedestrian crossings. Additionally, a novel reinforcement learning agent is developed to dynamically select the optimal dilation rate in the Temporal Convolutional Network (TCN) that is part of the forecasting model, used in the forecasting stage of the two-stage framework.

5.1 Contribution

The primary contribution of this thesis is developing a novel hybrid deep-learning two-stage framework for forecasting advisory speeds for vehicles making left turns at intersections before reaching the pedestrian crossings, with a particular emphasis on optimizing deep learning architectures using reinforcement learning (RL).

To assess the effectiveness of the proposed framework, a real-world vehicle trajectory dataset collected from Lankershim Boulevard, which includes speed and acceleration data, is used. To replicate real-world scenarios, where the advisory speed needs to provided to drivers some time before the vehicle reaches the pedestrian crossing, the last 3 seconds of speed and acceleration data are removed from the dataset.

In the first stage of the proposed framework the removed data is imputed using a Convolutional Neural Network (CNN), which is then used for forecasting vehicle speed in the second stage of the two-stage framework. In the second stage, a hybrid deep learning model created using a Temporal Convolutional Network (TCN), a Long Short-Term Memory (LSTM) and

Random Forest Regressor (RFR) are combined to forecast the speed of the vehicles over the pedestrian crossings.

Furthermore, a reinforcement learning (RL) agent is developed to dynamically select the optimal dilation rates for the TCN layers in the forecasting model. Operating in a custom environment, the agent makes sequential decisions based on the current layer index, the previously chosen dilation rate, and the cumulative receptive field. It receives feedback in the form of negative validation loss, incentivizing configurations that minimize forecasting errors. The optimized TCN structure, discovered through this adaptive search, is then incorporated into the hybrid deep learning model.

Extensive comparative analysis revealed that using the RL-learned hybrid architecture was able to outperform all other alternative models used in the experiment to forecast speed by achieving a Mean Absolute Percentage Error (MAPE) value of 9.85%. While hybrid and multi-stage deep learning models are effective on their own, the inclusion of RL opens new avenues for adaptive, data-driven model refinement, potentially leading to safer and more efficient traffic systems.

The RL-enhanced model achieved a MAPE value lower than any alternative configurations, as it validated the feasibility and potential of using reinforcement learning for automated deep learning architecture optimization in time-series forecasting tasks. This highlights a promising direction for future work, where RL can serve as a meta-learning tool to fine-tune complex models for domain-specific applications such as urban traffic management.

5.2 Limitations

5.2.1 Limited Performance Gain

The models evaluated in this study, including Random Forest Regressor (RFR), Temporal Convolutional Network (TCN), Long Short-Term Memory (LSTM), and hybrid or attention-based models such as TCN-LSTM-RFR and Sequence-to-Sequence with Attention and RFR, were compared using the Mean Absolute Percentage Error (MAPE) metric. While the differences in performance were relatively small, typically ranging from 1% to 3%, these improvements still demonstrate the viability of the proposed framework. Specifically, the framework shows potential for forecasting the speed of left-turning vehicles over pedestrian crossings and providing advisory speed recommendations to drivers. With access to larger datasets and richer feature sets, these gains are expected to become more pronounced. The key contribution of this work lies in demonstrating that reinforcement-learned TCNs can adapt to dynamic traffic data, paving the way for more practical and scalable forecasting models.

5.2.2 Inference Time

The proposed framework in this thesis aims to provide advisory speed recommendations to drivers of Connected and Autonomous Vehicles (CAVs). To simulate a real-world scenario, the last 3 seconds of speed and acceleration data are intentionally removed. The framework then imputes these missing values and forecasts the vehicle's speed as it approaches pedestrian crossings.

However, simply removing 3 seconds of data is not sufficient to evaluate the framework's practical viability. In real-world applications, the framework must not only accurately impute and forecast speed values but also do so within a time frame suitable for real-time decision-making.

While model training is computationally intensive, it is performed offline. In a deployed CAV system, only the inference phase, which would comprise of data preprocessing (i.e., extracting speed and acceleration values), imputation, and forecasting. These task would be executed in real time, and since vehicle sensors can quickly extract speed and acceleration data [73], the primary concern is the time taken for inference.

To evaluate this, the framework was tested on an ASUS TUF GAMING FX705GE machine with an Intel Core i7-8750H CPU @ 2.20GHz, 16GB RAM, running Windows 11. The inference times for the proposed framework and baseline models are shown in Figure 5.1. The results indicate that the framework achieves inference in under one second, demonstrating its potential for real-time integration into autonomous vehicles.

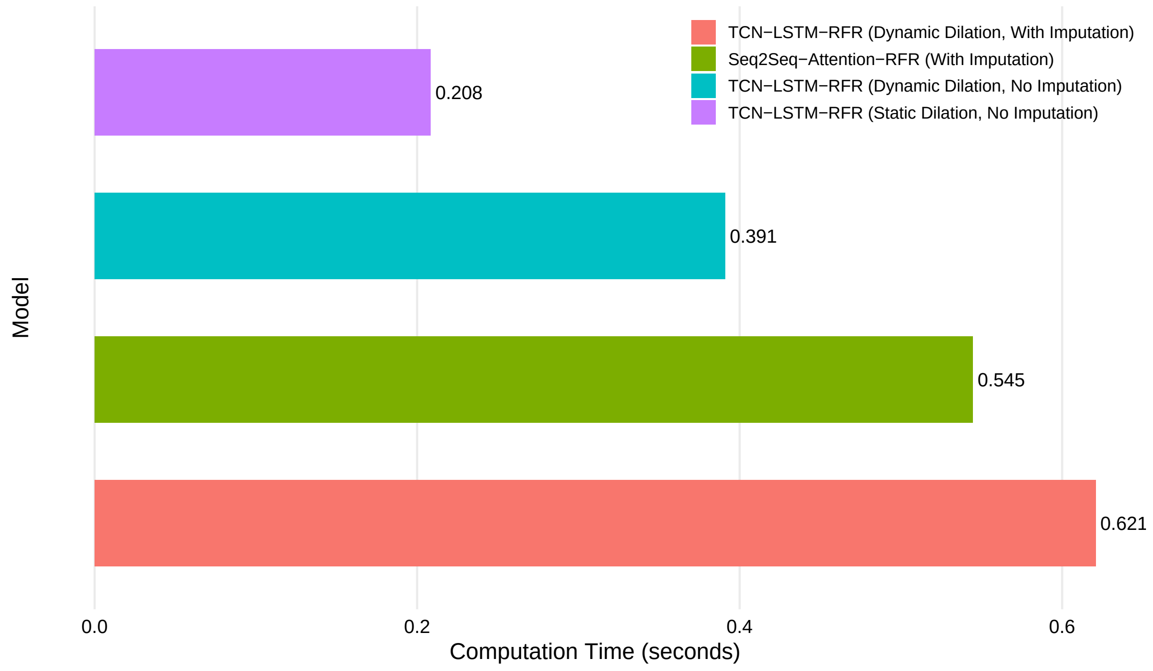

Figure 5.1: Inference time for models to forecast the speed of vehicles over the pedestrian crossing

Nonetheless, this evaluation was conducted on relatively high-end hardware. For broader applicability, especially in vehicles with less sophisticated onboard systems, the framework should be tested on embedded systems or edge devices to ensure consistent performance under constrained computational resources.

5.2.3 Generalizability

While the proposed framework demonstrates promising results in imputing and forecasting vehicle speed over the pedestrian crossings, its generalizability remains a key limitation. Two primary concerns must be addressed to ensure broader applicability:

1. The framework has been tested using only one dataset. Although this provides initial insights into its performance, it limits the ability to generalize the results across

different driving environments, vehicle types, and sensor configurations. To validate the robustness and adaptability of the framework, it must be evaluated on diverse datasets collected from various sources and scenarios.

2. The dataset used for training and evaluation is relatively small. This scarcity of data can restrict the model's ability to learn complex patterns and may affect its performance in real-world conditions. For the framework to be reliably deployed in autonomous vehicles, it should be trained on a larger and more comprehensive dataset. This would not only improve its accuracy but also enhance its ability to generalize across different operational contexts.

Addressing these limitations through broader data collection and cross-dataset evaluation will be essential for transitioning the framework from a research prototype to a deployable solution in real-world autonomous vehicle systems.

5.3 Future Works

Future enhancements to the RL-based TCN-LSTM-RFR model's performance could involve integrating additional contextual variables, such as spatial and temporal gaps between the subject vehicle and surrounding traffic, to further improve forecasting accuracy. Refining data imputation techniques, particularly for reconstructing the final three seconds of the removed or incomplete speed and acceleration data, also offers potential for improving prediction robustness.

Furthermore, the forecasting capability of the RL-based TCN-LSTM-RFR model could be advanced through the development of more sophisticated reinforcement learning strategies. Specifically, optimizing the policy design for selecting dilation rates in the TCN module can lead to more effective temporal feature extraction. By allowing the model to dynamically adapt its dilation configurations based on evolving traffic scenarios, it may better capture long-term dependencies and sudden behavioral changes in vehicle trajectories.

Additionally, integrating external environmental and behavioral cues, such as road geometry, weather conditions, and driver intent, could further enhance the model's generalization ability across diverse urban driving contexts. Exploring multi-agent reinforcement learning approaches to account for interactions among multiple vehicles presents another promising direction for future research.

Finally, the deployment of target or advisory speed predictions derived from this model into real-world traffic systems offers significant potential for improving urban mobility and safety. This requires close collaboration with urban planners and traffic management authorities to ensure seamless integration with existing intelligent transportation infrastructure. Such efforts could facilitate the adoption of the proposed framework in real-time applications, ultimately contributing to safer, more efficient, and more adaptive urban traffic ecosystems

Bibliography

[1] IIHS, "Fatality facts 2022: pedestrians," *Fatality Facts 2022 Pedestrians*, Jun. 2024. https://www.iihs.org/research-areas/fatality-statistics/detail/pedestrians#:~:text=Pedestrian%20fatalities%20account%20for%2018,their%20lowest%20point%20in%202009.&text=Twenty%2Dfive%20percent%20of%20pedestrian,hit%2Dand%2Drun%20crashes.

[2] S. C. Government of Canada, "Circumstances surrounding pedestrian fatalities, 2018 to 2020," *The Daily* -, Oct. 2023. https://www150.statcan.gc.ca/n1/daily-quotidien/231030/dq231030a-eng.htm

[3] C. Oh, Y. Kang, B. Kim, and W. Kim, "Analysis of pedestrian-vehicle crashes in korea: Focused on developing probabilistic pedestrian fatality model," in *Proceedings: International technical conference on the enhanced safety of vehicles*, National Highway Traffic Safety Administration, 2005, pp. 8p–8p.

[4] P. E. Gårder, "The impact of speed and other variables on pedestrian safety in maine," *Accident Analysis & Prevention*, vol. 36, no. 4, pp. 533–542, 2004.

[5] K. Haleem, P. Alluri, and A. Gan, "Analyzing pedestrian crash injury severity at signalized and non-signalized locations," *Accident Analysis & Prevention*, vol. 81, pp. 14–23, 2015.

[6] A. Ankunda, Y. Ali, and M. Mohanty, "Pedestrian crash risk analysis using extreme value models: New insights and evidence," *Accident Analysis & Prevention*, vol. 203, p. 107633, 2024.

[7] N. Mahmoud, M. Abdel-Aty, and A. Abdelraouf, "The impact of target speed on pedestrian, bike, and speeding crash frequencies," *Accident Analysis & Prevention*, vol. 192, p. 107263, 2023.

[8] J. Jin *et al.*, "Variable speed limit modelling to improve traffic safety and efficiency of mixed traffic flow by a two-stage framework," *Transportmetrica A: Transport Science*, vol. 21, no. 2, p. 2253476, 2025.

[9] Q. Zhang, H. Kong, X. Zhang, and T. Liu, "Considering cut-in behavior in mixed traffic: A longitudinal potential field-based car-following model for autonomous vehicles," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 239, no. 1, pp. 206–219, 2025.

[10] M. H. Tawfeek and K. El-Basyouny, "Location-based analysis of car-following behavior during braking using naturalistic driving data," *Canadian Journal of Civil Engineering*, vol. 47, no. 5, pp. 498–505, 2020.

[11] M. H. Tawfeek, "Human-like speed modeling for autonomous vehicles during car-following at intersections," *Canadian Journal of Civil Engineering*, vol. 49, no. 2, pp. 255–264, 2022.

[12] K. Nobukawa, T. Gordon, and D. LeBlanc, "Anticipatory speed control model applied to intersection left turns," *Vehicle system dynamics*, vol. 50, no. 11, pp. 1653–1672, 2012.

[13] C. Dias, M. Iryo-Asano, M. Abdullah, T. Oguchi, and W. Alhajyaseen, "Modeling trajectories and trajectory variation of turning vehicles at signalized intersections," *IEEE Access*, vol. 8, pp. 109821–109834, 2020, doi: 10.1109/ACCESS.2020.3002020.

[14] S. Long, F. Hui, L. Cheng, W. Wang, Y. Wang, and X. Jin, "Intelligent vehicle left-turn trajectory planning at intersections," in *2023 7th international conference on transportation information and safety (ICTIS)*, IEEE, 2023, pp. 2059–2067.

[15] J. Zhao, V. L. Knoop, J. Sun, Z. Ma, and M. Wang, "Unprotected left-turn behavior model capturing path variations at intersections," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 9, pp. 9016–9030, 2023.

[16] R. S. Tsay, "Time series and forecasting: Brief history and future research," *Journal of the American Statistical Association*, vol. 95, no. 450, pp. 638–643, 2000.

[17] Z. Hussain, S. S. Mohammed, C. Dias, Q. Hussain, and W. K. Alhajyaseen, "Empirical analysis of car-following behavior: Impacts of driver demographics, leading vehicle types, and speed limits on driver behavior and safety," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 108, pp. 188–205, 2025.

[18] Z. Zhou, Z. Yang, Y. Zhang, Y. Huang, H. Chen, and Z. Yu, “A comprehensive study of speed prediction in transportation system: From vehicle to traffic,” *Iscience*, vol. 25, no. 3, 2022.

[19] A. Elfar, A. Talebpour, and H. S. Mahmassani, “Predictive speed harmonization using machine learning in traffic flow with connected and automated vehicles,” *Transportation research record*, vol. 2678, no. 4, pp. 398–414, 2024.

[20] S. Chand, “Modeling predictability of traffic counts at signalised intersections using hurst exponent,” *Entropy*, vol. 23, no. 2, p. 188, 2021.

[21] P. Chen, H. Liu, H. Qi, and F. Wang, “Analysis of delay variability at isolated signalized intersections,” *Journal of Zhejiang University SCIENCE A*, vol. 14, no. 10, pp. 691–704, 2013.

[22] Y. Akagi and P. Raksincharoensak, “Stochastic driver speed control behavior modeling in urban intersections using risk potential-based motion planning framework,” in *2015 IEEE intelligent vehicles symposium (IV)*, IEEE, 2015, pp. 368–373.

[23] A. Wolfermann, W. Alhajyaseen, and H. Nakamura, “Modeling speed profiles of turning vehicles at signalized intersections,” in *3rd international conference on road safety and simulation RSS2011, transportation research board TRB, indianapolis*, 2011, pp. 1–17.

[24] K. Tang, S. Chen, Y. Cao, D. Zang, and J. Sun, “Lane-level short-term travel speed prediction for urban expressways: An attentive spatio-temporal deep learning approach,” *IET Intelligent Transport Systems*, vol. 18, no. 4, pp. 709–722, 2024.

[25] P. Polverino, E. A. Adinolfi, and C. Pianese, “Target speed computation through predictive cruise control for vehicles energy consumption reduction,” *Energy Conversion and Management*, vol. 298, p. 117757, 2023.

[26] M. Nesa and Y. Yoon, “Speed prediction and nearby road impact analysis using machine learning and ensemble of explainable AI techniques,” *Scientific Reports*, vol. 14, no. 1, p. 25208, 2024.

[27] U. Sander, "Opportunities and limitations for intersection collision intervention—a study of real world 'left turn across path'accidents," *Accident Analysis & Prevention*, vol. 99, pp. 342–355, 2017.

[28] C. Xue-mei, W. Zi-jia, L. Meng-xi, *et al.*, "A multiple-vehicle decision-making model for left-turn behavior of AVs at urban intersections based on conflict resolution," in *2020 7th international conference on information science and control engineering (ICISCE)*, IEEE, 2020, pp. 1565–1571.

[29] B. Tang, S. Khokhar, and R. Gupta, "Turn prediction at generalized intersections," in *2015 IEEE intelligent vehicles symposium (IV)*, IEEE, 2015, pp. 1399–1404.

[30] M. S. Shirazi and B. T. Morris, "Vision-based turning movement monitoring: Count, speed & waiting time estimation," *IEEE Intelligent Transportation Systems Magazine*, vol. 8, no. 1, pp. 23–34, 2016.

[31] A. Abdelhalim, M. Abbas, B. B. Kotha, and A. Wicks, "A framework for real-time traffic trajectory tracking, speed estimation, and driver behavior calibration at urban intersections using virtual traffic lanes," in *2021 IEEE international intelligent transportation systems conference (ITSC)*, IEEE, 2021, pp. 2863–2868.

[32] Z. Liu, Z. Zhu, J. Gao, and C. Xu, "Forecast methods for time series data: A survey," *Ieee Access*, vol. 9, pp. 91896–91912, 2021.

[33] M. Valipour, "Critical areas of iran for agriculture water management according to the annual rainfall," *Eur. J. Sci. Res*, vol. 84, no. 4, pp. 600–608, 2012.

[34] Z. Malki *et al.*, "ARIMA models for predicting the end of COVID-19 pandemic and the risk of second rebound," *Neural Computing and Applications*, vol. 33, pp. 2929–2948, 2021.

[35] F. Pokou, J. Sadefo Kamdem, and F. Benhmad, "Hybridization of ARIMA with learning models for forecasting of stock market time series," *Computational Economics*, vol. 63, no. 4, pp. 1349–1399, 2024.

[36] Q. Wen *et al.*, "Transformers in time series: A survey," *arXiv preprint arXiv:2202.07125*, 2022.

[37] B. Mossop, R. Bismil, and Q. A. Rahman, "Forecasting hospital mental-health admissions with a novel hybrid deep learning architecture," in *2023 IEEE international conference on bioinformatics and biomedicine (BIBM)*, IEEE, 2023, pp. 4093–4100.

[38] B. Mossop and Q. A. Rahman, "Infectious disease forecasting using multivariate incomplete time-series: A hybrid architecture with stacked dilated causal convolutions," in *2023 IEEE international conference on bioinformatics and biomedicine (BIBM)*, IEEE, 2023, pp. 4220–4227.

[39] Z. Chu, W. Ma, M. Li, and H. Chen, "Adaptive decision spatio-temporal neural ODE for traffic flow forecasting with multi-kernel temporal dynamic dilation convolution," *Neural Networks*, vol. 179, p. 106549, 2024.

[40] J. Yao, D. Wang, H. Hu, W. Xing, and L. Wang, "ADCNN: Towards learning adaptive dilation for convolutional neural networks," *Pattern Recognition*, vol. 123, p. 108369, 2022.

[41] H. Abed and B. Gyires-Tóth, "Adaptive temporal convolutional network for language modeling," Jan. 2024, pp. 85–90. doi: 10.3311/WINS2024-015.

[42] W. Zhan *et al.*, "INTERACTION dataset: An international, adversarial and cooperative motion dataset in interactive driving scenarios with semantic maps," *arXiv:1910.03088 [cs, eess]*, 2019.

[43] J. Bock, R. Krajewski, T. Moers, S. Runde, L. Vater, and L. Eckstein, "The inD dataset: A drone dataset of naturalistic road user trajectories at german intersections," in *2020 IEEE intelligent vehicles symposium (IV)*, 2020, pp. 1929–1934. doi: 10.1109/IV47402.2020.9304839.

[44] E. Barmpounakis and N. Geroliminis, "pNEUMA dataset." Zenodo, 2024. doi: 10.5281/zenodo.10491409.

[45] P. Sun *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2020.

[46] S. Ettinger *et al.*, “Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset,” in *Proceedings of the IEEE/CVF international conference on computer vision (ICCV)*, pp. 9710–9719.

[47] K. Chen *et al.*, “WOMD-LiDAR: Raw sensor dataset benchmark for motion forecasting,” in *Proceedings of the IEEE international conference on robotics and automation (ICRA)*, 2024.

[48] B. Wilson *et al.*, “Argoverse 2: Next generation datasets for self-driving perception and forecasting,” in *Proceedings of the neural information processing systems track on datasets and benchmarks (NeurIPS datasets and benchmarks 2021)*, 2021.

[49] H. Caesar *et al.*, “nuScenes: A multimodal dataset for autonomous driving,” *arXiv preprint arXiv:1903.11027*, 2019.

[50] A. Rasouli, I. Kotseruba, and J. K. Tsotsos, “Are they going to cross? A benchmark dataset and baseline for pedestrian crosswalk behavior,” in *Proceedings of the IEEE international conference on computer vision workshops*, 2017, pp. 206–213.

[51] A. Rasouli, I. Kotseruba, T. Kunic, and J. K. Tsotsos, “PIE: A large-scale dataset and models for pedestrian intention estimation and trajectory prediction,” in *International conference on computer vision (ICCV)*, 2019.

[52] U. S. D. of Transportation Federal Highway Administration, “Next generation simulation (NGSIM) program lankershim boulevard videos.” http://doi.org/10.21949/1504477, 2016.

[53] G. Bontempi, S. Ben Taieb, and Y.-A. Le Borgne, “Machine learning strategies for time series forecasting,” in *Business intelligence: Second european summer school, eBISS 2012, brussels, belgium, july 15-21, 2012, tutorial lectures*, M.-A. Aufaure and E. Zimányi, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 62–77. doi: 10.1007/978-3-642-36318-4_3.

[54] N. K. Ahmed, A. F. Atiya, N. E. Gayar, and H. E.-S. and, “An empirical comparison of machine learning models for time series forecasting,” *Econometric Reviews*, vol. 29, no. 5–6, pp. 594–621, 2010, doi: 10.1080/07474938.2010.481556.

[55] K.-R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik, “Predicting time series with support vector machines,” in *International conference on artificial neural networks*, Springer, 1997, pp. 999–1004.

[56] S. Samantaray and A. Sahoo, “Prediction of flow discharge in mahanadi river basin, india, based on novel hybrid SVM approaches,” *Environment, Development and Sustainability*, vol. 26, no. 7, pp. 18699–18723, 2024.

[57] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.

[58] G. Avinash, H. Pachori, A. Sharma, and S. Mishra, “Time series forecasting of bed occupancy in mental health facilities in india using machine learning,” *Scientific Reports*, vol. 15, no. 1, p. 2686, 2025.

[59] T. Q. Pham, T. Matsui, and J. Chikazoe, “Evaluation of the hierarchical correspondence between the human brain and artificial neural networks: A review,” *Biology*, vol. 12, no. 10, p. 1330, 2023.

[60] H. Hewamalage, C. Bergmeir, and K. Bandara, “Recurrent neural networks for time series forecasting: Current status and future directions,” *International Journal of Forecasting*, vol. 37, no. 1, pp. 388–427, 2021.

[61] I. E. Livieris, E. Pintelas, and P. Pintelas, “A CNN–LSTM model for gold price time-series forecasting,” *Neural computing and applications*, vol. 32, pp. 17351–17360, 2020.

[62] S. Siami-Namini, N. Tavakoli, and A. S. Namin, “The performance of LSTM and BiLSTM in forecasting time series,” in *2019 IEEE international conference on big data (big data)*, IEEE, 2019, pp. 3285–3292.

[63] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, “Temporal convolutional networks for action segmentation and detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 156–165.

[64] A. B. P. Utama, A. P. Wibawa, A. N. Handayani, W. S. G. Irianto, A. Nyoto, *et al.*, “Improving time-series forecasting performance using imputation techniques in deep learning,” in *2024 international conference on smart computing, IoT and machine learning (SIML)*, IEEE, 2024, pp. 232–238.

[65] Z. Zhang, “Missing data imputation: Focusing on single imputation,” *Annals of translational medicine*, vol. 4, no. 1, p. 9, 2016.

[66] A. Vaswani *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[67] Y. Zhang and P. J. Thorburn, “A dual-head attention model for time series data imputation,” *Computers and Electronics in Agriculture*, vol. 189, p. 106377, 2021.

[68] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister, “Temporal fusion transformers for interpretable multi-horizon time series forecasting,” *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, 2021.

[69] W. Cao, D. Wang, J. Li, H. Zhou, L. Li, and Y. Li, “Brits: Bidirectional recurrent imputation for time series,” *Advances in neural information processing systems*, vol. 31, 2018.

[70] W. Du, D. Côté, and Y. Liu, “Saits: Self-attention-based imputation for time series,” *Expert Systems with Applications*, vol. 219, p. 119619, 2023.

[71] S. Fang, Q. Wen, Y. Luo, S. Zhe, and L. Sun, “BayOTIDE: Bayesian online multivariate time series imputation with functional decomposition,” *arXiv preprint arXiv:2308.14906*, 2023.

[72] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[73] X. Chen, Z. Wang, Q. Hua, W.-L. Shang, Q. Luo, and K. Yu, “AI-empowered speed extraction via port-like videos for vehicular trajectory analysis,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 4, pp. 4541–4552, 2022.