CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Interpretable Methods for Adaptive Route Improvement Models Based on Behavioral Trajectory Prediction

Master's thesis in Computer science and engineering

Lucas Karlsson
Georges Kayembe

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Interpretable Methods for Adaptive Route Improvement Models Based on Behavioral Trajectory Prediction

Lucas Karlsson
Georges Kayembe

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Interpretable Methods for Adaptive Route Improvement Models Based on Behavioral Trajectory Prediction
LUCAS KARLSSON, GEORGES KAYEMBE

Supervisor: Hazem Torfah, Department of Computer Science and Engineering
Advisors: Jens Andersson and Jeffrey Spång, WirelessCar
Examiner: Nir Piterman, Department of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Interpretable Methods for Adaptive Route Improvement Models Based on Behavioral Trajectory Prediction

LUCAS KARLSSON
GEORGES KAYEMBE
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The emergence of Artifical Intelligence (AI) and Machine Learning (ML) have transformed various sectors by enabling machines to learn from data, recongnize patters, make decisions, and perform taskts requiring human intelligence. In the automotive domain, Smart EV Routing by WirelessCar optimizes electric vehicle routes using real-time and user data. One potential modification to such a system would be the integration of a model that autonomously determines the route based on historical data and environmental conditions. With the use of interpretability, one can ensure that such a system works and gives accurate predictions reflecting the user driven data. This thesis develops a framework to integrate such a system while focusing on interpretable trajectory prediction models and interpretability techniques. It also tackles the complexity of multivariate data and addresses issues related to data scarcity. While the focus has primarily been on models learned in simulation, the framework is designed with future applications in mind, aiming to support the development of methods with similar constraints and requirements.

The results show that the existing interpretability methods are inadequate for scenarios involving time series data where multiple variables (multivariate settings) affect the outcomes over time. Attempting to aggregate these methods in multivariate settings results in loss of information and increased complexity. This renders these methods impractical for dynamic domains such as car trajectory prediction, this was also evident in the result. To address these issues we developed Interpret Multivariate Timeseries (IMT), a method developed specifically for multivariate settings, as a true black box explainer. The results also show that our framework integrates multivariate forecasting models, interpretability methods, adaptability, and data generation, enabling interpretable and adaptive route improvements based on behavioral trajectory prediction.

Keywords: Machine Learning, Route Improvement, Trajectory Prediction, Interpretability, Transformers.

Acknowledgements

We would like to thank our supervisor, Hazem, for his help and guidance throughout the thesis. We would also like to thank our industry partner, WirelessCar, and our supervisors at the company, Jens and Jeffrey.

Lucas Karlsson & Georges Kayembe, Gothenburg, 2024-08-01

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

In this century, the rise of Artificial Intelligence (AI) and Machine Learning (ML) has been among the most impactful technological advancements. These technologies have transformed numerous aspects of our lives by enabling machines to learn from data, identify patterns, make decisions, and carry out tasks that traditionally require human intelligence. This transformation has been especially observable in the automotive domain. For instance, AI-based models have been used in predictive maintenance systems [1] where they analyze vehicle sensor data to anticipate and prevent potential breakdowns, thus improving vehicle reliability, reducing maintenance costs, and optimizing safety. Additionally, AI-based models have also been used in various driving support systems, such as problem-critical functions like lane-keeping assistance [2] and autonomous emergency braking [3]. The integration of such features is vital to meet the growing demand for safer vehicles. Despite the significant benefits, abstracting decisions from the users in favor of AI and ML introduces safety concerns. These concerns stem from the fact that the decision-making process is often obscured by the models, rendering the process opaque [4].

This opacity raises crucial questions: why should users blindly trust what a computer tells them, and what aspects of a users behavior led to a specific conclusion? How can one construct a system that performs predictions while maintaining transparency to end-users? These questions underscore the challenges and complexities associated with implementing machine learning models for adaptive route prediction. The issues of user trust, transparency, and interpretability are critical considerations that must be addressed to ensure the successful adoption and acceptance of such systems. This thesis developed a framework for adaptive route prediction and tackled the mentioned challenges, focusing on providing users with insights into the interpretation process of machine learning models by studying, testing and creating interpretability techniques.

Collaborating with WirelessCar [5], one of the leading innovators of connected vehicle services, and the creators of the service Smart EV Routing [6], we particularly investigated interpretability concerning a modification to Smart EV Routing. Smart EV Routing enables electric vehicle drivers to receive a route that takes charging stops into account after being given a start and a destination, ensuring that their journey is as convenient and safe as possible. The product uses real-time data from the vehicle and its environment to provide the most optimized route. The potential modification to the system is the integration of machine learning models that autonomously determine the route based on historical data and current environmental conditions. This approach would eliminate the need for direct user interaction, allowing for a seamless experience where users can simply enter the vehicle and trust that the AI-powered system will navigate them safely to their destination. Furthermore, anticipating driver routes in advance could facilitate the proactive identification of available charging stops along the journey. Additionally, knowledge of anticipated routes for many vehicles could improve urban planning and infrastructure development efforts, enabling authorities to optimize traffic flow and reduce congestion. This can be achieved by rerouting some vehicles away from congested areas, which will improve overall traffic efficiency and, in some cases, reduce travel times for commuters. These ideas show the potential of such a modelling and holds a great promise for further studying and optimizations.

The framework developed in this thesis effectively addresses the challenges presented, where the solution provided by the framework is structured into two sub-blocks: the learning block, and the interpretability block. In the learning block, data generation methods for simulating and generating efficient historical driver data are meticulously implemented to provide the framework with high-quality data for learning purposes. Moreover, a range of machine learning models for trajectory predictions are integrated to meet the need for models capable of capturing the relationships in the multivariate setting of the driver's historical data. Additionally, the learning block incorporates an adaptability paradigm that enables models to enhance their results based on previous predictions. While creating the learning block for adaptive route prediction, several challenges emerged. Such as the complex task of finding, tuning, and creating models that can capture relationships in multivariate data while maintaining interpretability. Additionally, there is the issue of the scarcity of real data. Since most car manufacturers own the rights to the data collected by their vehicles and are hesitant to share it with other companies, generating and simulating representative data for driving scenarios while ensuring its real-world applicability presents a significant challenge. The learning block is evaluated using driving scenarios designed as benchmarks to closely replicate real-world conditions. These scenarios serve as a basis for evaluating the framework's performance, including assessing how well the machine learning models predict car trajectories and refine their results over time. Simulation-based analyses are occasionally employed to enhance the accuracy of these predictions.

Finally, in the interpretability block, the framework introduces a novel method of interpretability called Interpretable Multivariate Timeseries (IMT). This method offers a suitable approach for explaining the predictions of complex models within the context of multivariate settings. The primary challenge with the interpretability block concerns the traditional methods of interpreting models that are often not suited for multivariate settings, where time series data involves multiple variables influencing outcomes over time. In order to use the traditional methods, modifications and aggregations are needed. Resulting in other issues, such as a loss of information and increased computational complexity. Rendering the methods impractical. The evaluation process of the interpretability block involves exploring how effectively the interpretability methods developed within the framework elucidate the decision-making processes and underlying factors influencing the model predictions. By conducting extensive evaluations and benchmarks across various driving scenarios, the framework's capabilities and performance can be accurately assessed.

The scope of this thesis is focused on laying the foundation for future research and development in the field of interpretable methods for trajectory prediction or similarly constrained prediction models. While the exploration of interpretability methods and trajectory prediction models has primarily focused on simulation-based approaches, there is ample opportunity for further refinement and expansion in real-world environments (this will be discussed further in the upcoming section). Additionally, no direct work was conducted with Smart EV Routing during the implementation of our suggested framework. The objective was to build multiple proofs-of-concept using our framework to demonstrate its functionality, as well as its possibilities and limitations.

We summarize our contributions as follows:

- **Learning framework**

  We present an approach for learning models suitable for adaptive route improvement in the form of a framework. The framework presents a robust methodology for constructing machine learning models capable of effectively handling the complexities inherent in multivariate data.

  A key feature of the framework is its adaptability mechanism. This feature enables machine learning models to continuously refine their predictions based on previous outcomes, thereby enhancing their predictive accuracy and responsiveness to evolving driving conditions.

  Additionally, the framework introduces an innovative approach to tackle data scarcity in trajectory prediction. Through the utilization of probabilistic methods to generate simulated high-quality data that closely mirrors real-world settings, effectively overcoming the limitations imposed by data availability.

- **Interpretability methods**

  The primary contribution of this thesis is in the realm of interpretability. It introduces a novel method called IMT for interpreting machine learning models within the context of multivariate time series. By providing insights into the decision-making processes of underlying models, IMT enhances the transparency and trustworthiness of trajectory prediction systems. This method highlights the importance of understanding how individual features influence predictions, thereby making the models more accessible and reliable for users.

- **Extensive evaluations**

  The framework provides valuable insights into the effectiveness and practical applicability of the methods developed in this thesis through rigorous testing and benchmarking across various driving scenarios.

2

Background

This chapter provides the necessary background and theoretical aspects of the project, which are later used in Chapter 3 and in Chapter 4. It begins by exploring the concept of interpretability in machine learning, followed by an examination of time series. Finally, it explores various models for trajectory prediction and some adaptive learning paradigms.

2.1 Interpretability

A machine learning model is interpretable if it allows someone to understand the reasons behind its decisions or predictions [7]. However, even if a machine learning model performs effectively, why do observers need to know the reasoning behind a specific decision? These machine learning models are being used in more and more important areas, such as healthcare and finance. While it is great to have automation, it needs to ensure that the systems created are strong and work properly, because they are used in sectors where even small errors can have significant consequences. Additionally, understanding how the model works and why it makes a certain decision, builds trust among users, enhancing their acceptance and adoption. In the following subsections, the taxonomies of interpretable machine learning are explored, followed by an exploration of their methodologies.

2.1.1 Taxonomies of Interpretability

Interpretability is not a one-dimensional attribute that can easily be classified, and is thus rarely a one-size-fits-all solution. Several papers have proposed ways of categorizing methods in interpretability [4][7][8][9][10]. In this section, a selection of these categorizations are explored.

2.1.1.1 Intrinsic and Post-hoc Interpretability

Interpretability models can be categorized based on the timing and location of explanation generation. Intrinsic interpretability involves incorporating interpretability directly into the model's structure, making the model self-explanatory [4][8]. For example, decision trees can be categorized as intrinsic interpretability models because they are naturally self-interpretable due to their structure, which allows one to easily understand the reasoning behind a specific result [4][8].

With intrinsic interpretability, explanations for predictions are generated during the decision-making process or while the model is being trained [4][8]. Post-hoc interpretability, on the other hand, involves using an external model to provide explanations for an existing model. In this approach, explanations for predictions are generated after the model being explained has been trained [4][8].

2.1.1.2 Global and Local Interpretability

Models of interpretability can be classified based on their scope as either *global* or *local*. Global interpretability refers to the understanding of a machine learning model's overall behavior. It aims to provide insights into the entire model, including how it makes decisions across all possible inputs. This can involve examining the importance of features and the interactions between them to understand the model's overall logic [10]. However, achieving global interpretability presents challenges due to the potential complexity arising from the number of parameters or features within a model. The greater the number of features, the more complex it becomes to establish a definitive global understanding of the model's workings [10].

Local interpretability, on the other hand, focuses on exploring the model's prediction for a specific instance, examining what the model predicts for this input, and why. This can involve examining the contribution of each feature to the prediction for a single instance. By analyzing a single prediction for each instance, this approach aims to finally decode the workings of the complex model [10].

2.1.1.3 Model-specific and Model-agnostic Explanations

Model-specific and model-agnostic explanations are two broad categories within the field of interpretable machine learning, each with its own approach to explaining the predictions made by machine learning models.

Model-specific explanations are customized to fit the inner workings of specific types of machine-learning models. As previously mentioned, decision trees are naturally interpretable because their structure, comprising branches and nodes representing decision rules, can be directly examined to understand how a decision is made. Model-specific explanations are most effective when the model's architecture facilitates direct interpretability [8].

Model-agnostic explanations, on the other hand, provide interpretations without any assumptions on the model internals. This approach does not rely on understanding the specific architecture of the model, but instead treats the model as a black box, focusing on its input-output relationship to generate explanations [8].

2.1.2 Methods of Interpretability

Numerous methods exist for inspecting machine learning models and elucidating their predictive behaviors [4][7][8][9][10]. In this section, a range of these techniques will be explored.

2.1.2.1 Feature Importance

The Feature Importance (FI) method offers a very simple but effective solution, acting as either a global or local interpretation. This method consists of providing an explanation through a ranked list of features according to their influence on the model's prediction [9]. Within the FI framework, permutation feature importance stands out as a key technique. It quantifies a feature's significance by evaluating the rise in prediction error of the model after the feature's values have been shuffled [10]. A feature is considered important when shuffling its values results in a higher model error, indicating the model's dependence on that feature for making predictions. On the other hand, a feature is considered unimportant if altering its values leaves the model error unchanged [10]. Feature Importance helps in facilitating the understanding of which features have the most impact on a model's decision. Identifying the less significant features can also guide the process of simplifying a model without significantly reducing its performance [10].

The permutation feature importance algorithm can be modeled as following [10]: Consider a trained model *f* with feature matrix *X* and target *y*. Let *L(y, f)* represent the function used to measure the original model error, such as mean squared error.

Algorithm 1 The permutation feature importance algorithm from [10]

Estimate the original model error $e_{\text{orig}} = L(y, f(X))$
for each feature j in $\{1, \ldots, p\}$ **do**
    Generate feature matrix X_{perm} by permuting feature j in the data X
    Estimate error $e_{\text{perm}} = L(Y, f(X_{\text{perm}}))$ based on the predictions of the permuted data
    Calculate permutation feature importance as quotient $FI_j = \frac{e_{\text{perm}}}{e_{\text{orig}}}$ or difference $FI_j = e_{\text{perm}} - e_{\text{orig}}$
end for
Sort features by descending FI

Looking at the pseudocode in Algorithm 1, the association between feature j and the true outcome y is broken, by permuting feature j in the data *X*. This enables the measurement of the importance of the feature j in the prediction process. The feature importance value provides a good interpretation of the model's behavior by showing the increase in model error when the information of the feature j is destroyed.

2.1.2.2 Local Interpretable Model-Agnostic Explanations

Local Interpretable Model-Agnostic Explanations (LIME) is a well-established technique that is used to explain the reasoning behind a machine learning model's predictions after the model has been trained. This post-hoc interpretability approach works by creating a simple, interpretable model that mimics the behavior of the complex model for a specific instance. The primary goal of the surrogate model is to replicate the complex model's predictions closely while providing clear explanations for each prediction [9][10].

Imagine having a complex model that provides predictions based on a specific data input. LIME starts by generating numerous variations of the data, creating a new dataset around the original data input. Next, LIME uses the complex model to make predictions on this new dataset of perturbed samples. This step allows LIME to observe how changes in the input data affect the model's predictions. LIME then uses these observations to train a simpler, interpretable model, such as a linear regression or decision tree, which serves as the surrogate model [9][10]. This surrogate model is specifically customized to approximate the black-box model's behavior for the original data point and its perturbed neighbors [9][10]. The surrogate model, being simpler and interpretable, can then be analyzed to understand which features influenced the prediction for the specific instance.

Formally, LIME is defined by the formula in 2.1. This formula is later used in 3.3.2.2 and 4.2.2.2, where the modifications made to the existed LIME are presented.

$$\xi(x) = \arg\min_{g \in G} \{\mathcal{L}(f, g, \pi_x) + \Omega(g)\} \tag{2.1}$$

where:

- g is the surrogate model.
- G is a class of potentially interpretable models, such as linear models.
- f is the model or black box being explained.
- $\pi_x(z)$ is the proximity measure between an instance z to x.
- $\mathcal{L}(f, g, \pi_x)$ is the loss function measuring how unfaithful g is in approximating f in the locality defined by π_x
- $\Omega(g)$ represents the measure of complexity of the model g.

To ensure interpretability, LIME aims to minimize $\mathcal{L}(f, g, \pi_x)$ while keeping $\Omega(g)$ low enough for interpretability. However, many interpretable models within the class G presented in [11], such as linear models and decision trees, may still pose challenges to capturing the properties of sequential data representing the car's position, along with the data describing its current environment and behavior.

2.1.2.3 SHapley Additive exPlanations

SHapley Additive exPlanations (SHAP) is a model-agnostic interpretability technique that draws inspiration from the Shapley values in game theory. The goal is to fairly allocate the payout among the players based on their contributions. In the context of machine learning, the payout represents the model's prediction, and the players represent the model's features [10][12]. The Shapley values represent the contribution that each feature has to the prediction of a particular instance by considering all possible combinations of features [12]. Formally, the Shapley value of a feature can be calculated using the following formula in 2.2. This formula is later used in sections 3.3.2.1 and 4.2.2.1, where the modifications made to the existed SHAP are presented.

$$\phi_i = \sum_{S \subseteq P \setminus \{i\}} \frac{|S|!(P - |S| - 1)!}{P!} (val(S \cup \{i\}) - val(S)) \tag{2.2}$$

where:

- ϕ_i is the Shapley value for feature i.
- P represents the total number of features excluding feature i.
- S is a subset of features excluding feature i.
- $val(S)$ is the model's prediction value when considering the features in subset S.
- $val(S \cup \{i\})$ is the model's prediction value when including the feature i in the subset S.

One simple way to understand this formula could be to consider the model's prediction $val()$ that predicts acceleration based on the features longitude (L), latitude (Lat), and velocity (V). The Shapley values for each feature can show how they individually contribute to the prediction. For example, the step-by-step computation of the Shapley value for longitude (L) involves the following steps:

- **Set of features**: $P = \{L, Lat, V\}$
- **Feature to evaluate**: $i = L$
- **Remaining features**: $P - \{i\} = \{Lat, V\}$
- **Subsets S of remaining features**: $S \subseteq P$
  1. $S = \emptyset (emptyset)$
  2. $S = \{Lat\}$
  3. $S = \{V\}$
  4. $S = \{Lat, V\}$
- **Marginal contribution**:

  Assuming $\frac{1}{3}$ and $\frac{1}{6}$ are the Shapley weights ($\frac{|S|!(P-|S|-1)}{P!}$) for single-feature and two-feature subset, respectively three-feature subset. For each subset S, the marginal contribution of longitude L can be calculated using the formula in 2.2 as the following:
  1. $\phi_{L1} = \frac{1}{3}[val(L) - val(\emptyset)]$
  2. $\phi_{L2} = \frac{1}{6}[val(L, LAT) - val(Lat)]$
  3. $\phi_{L3} = \frac{1}{6}[val(L, V) - val(V)]$
  4. $\phi_{L4} = \frac{1}{6}[val(L, Lat, V) - val(Lat, V)]$
- **Average marginal contribution**:

- $\phi_L = average(\phi_{L1} + \phi_{L2} + \phi_{L3} + \phi_{L4})$
- ϕ_L is the average marginal contribution and represents the Shapley value for the longitude feature (L), indicating its contribution to the prediction of acceleration.

However, calculating Shapley values for each feature, as demonstrated in the example above, can be computationally expensive due to the size of the feature set and the exponential number of possible feature combinations [10]. Additionally, while optimized versions such as KernelSHAP and TreeSHAP [10][12] have been developed to enhance computational efficiency, adapting SHAP to handle the complexity of time series data remains challenging.

2.2 Time Series

To properly evaluate interpretability methods in the context of trajectory prediction, several types of datasets can be used. However, trajectories consist of a sequence of data points that describe the movement of an object over time. Each data point typically includes information such as position, velocity, and possibly other attributes. For this reason, time series datasets naturally capture this sequential nature, making them ideal for modeling trajectory data. In this subsection, some fundamental concepts of time series will be examined, and the methodology for processing such data for this project will be explored.

2.2.1 Time Series Analysis

To understand the future, one must analyze the past. This is called Time Series Analysis [13] and is a crucial part of time series forecasting. In the automotive domain, a lot of temporal measurements are monitored and saved via sensors and records. All measurements are then bound by the time of recording, and the observations can be ordered and indexed by timestamps. These timestamps are then used to organize the data into a sequential order based on the time, called time series. One of the simplest and quickest ways to observe and analyze these time series is some sort of temporal visualization where the data is simply plotted in order by time and temporal relations can be found by observing the graph. For example, in the automotive industry, a temporal relation could be fuel consumption and distance traveled.

2.2.2 Spatio-Temporal Time Series Analysis

In contrast to classic Temporal Time Series Analysis, Spatio-temporal analysis is used not only to capture the temporal relations but also how space and time are related. In the automotive domain, a position can be added as a temporal measurement, the result is a spatiotemporal relation describing how a car physically moves on earth concerning time. The complexity of analyzing the time series increases significantly, but the ability to do temporal visualizations remains unchanged.

2.2.3 Time Series Modeling Methods

As mentioned above, time series analysis involves the study of data points collected and ordered over time. This subsection explores different methodologies employed to construct time series data.

2.2.3.1 Univariate and Multivariate Time Series

Univariate time series forecasting involves predicting the future values of a single variable based on its past observations. This approach involves using historical data of a target variable to make predictions [14]. For instance, consider the time series dataset consisting of the latitude coordinates of a car's position at different time intervals. Using a univariate approach would involve building a model that considers only one feature (latitude) to make predictions of positions. Univariate time series forecasting is typically simpler and easier to implement [14]. However, it may not be an accurate approach to describe reality. In the example mentioned above, the univariate approach focuses solely on one variable, without considering other potentially relevant variables such as longitude, velocity, or heading. As a result, it may overlook important relationships that could improve the accuracy of trajectory predictions [14].

Multivariate time series forecasting involves predicting future values of multiple variables based on their past observations. This approach is more complex and requires more data than univariate time series forecasting. However, multivariate time series forecasting can be more accurate, as it takes into account the relationships between different variables [14][15]. For example, in the context of predicting the car's trajectory, only looking at how fast the car is moving, important things might be missed, such as which directions the car is heading or how far it has traveled along the latitude axis over time.

2.2.3.2 Single-step and Multiple-step Forecasting

In single-step forecasting, also known as one-step forecasting, the goal is to predict only the next value of the series. In the context of trajectory predictions, single-step forecasting could involve predicting the trajectory of the car for the immediate future, typically the next time step, which could be a second or minute into the future. For example, using the car's position and velocity at time t, and predict its position at $t+1$ (one time step ahead) [16].

In contrast, multiple-step forecasting involves predicting the trajectory of the car for multiple future time steps. Instead of predicting just the next time step, the aim is to forecast the car's position over a longer horizon, such as the next several seconds or minutes. For example, using the car's position and velocity at time t, and predict its position at $t+1$, $t+2$, $t+3$, and so on, for multiple time steps ahead [16].

2.3 Trajectory Prediction

Various methodologies have been explored by researchers to forecast vehicle trajectories, employing a broad spectrum of analytical and computational techniques. This pursuit has been driven by the critical need to improve the driving experience, enhance safety protocols, and advance the development of autonomous vehicles [17][18]. Trajectory prediction, characterized as a time series event, involves analyzing sequential data to anticipate future positions. This process is naturally complex due to the dynamic nature of vehicles' movements, influenced by numerous factors including driver behavior, traffic conditions, and environmental variables [19][17][18].

Given the sequential and temporal aspects of trajectory prediction, it aligns closely with time series analysis, requiring models capable of capturing temporal dependencies and patterns over time [19]. For the sake of conciseness, a selection of machine learning models that are particularly well-suited to the context of trajectory prediction and time series events will be explored, and some metrics to assess the accuracy of these models will be introduced. These models are adept at processing and predicting based on sequential data, offering insights into the likely paths vehicles will take based on historical and real-time data inputs.

2.3.1 Long Short-Term Memory

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) architecture [20], designed to address the issue with long sequences in machine learning. Specifically, LSTM can apprehend long-term dependencies in sequential data. This is done by incorporating a mechanism called gates that regulate the flow of information inside the architecture [21][22]. These gates determine what information should be kept, discarded, or passed on to the next time step, allowing the network to selectively remember or forget information over long periods. This selective memory capability makes LSTMs particularly effective for tasks where understanding the context from far back in the sequence is crucial for making accurate predictions or decisions.

Even with the development of numerous LSTM variants designed to overcome traditional challenges, the problem of vanishing and exploding gradients persists in certain scenarios [23]. Vanishing gradients mean that a models' optimization stagnates and shows no significant improvements, exploding gradients are the opposite, where a model's learning becomes unstable and won't learn any information from the training. For instance, in the realm of language modeling, LSTMs demonstrate an effective context size averaging around 200 tokens. However, they distinctly identify only about 50 tokens in proximity, suggesting that LSTMs, despite their advancements, may still face difficulties in capturing long-term dependencies under specific circumstances [23]. This limitation points to the need for further enhancements or alternative approaches to address these enduring challenges effectively.

2.3.2 Transformer

The Transformer model has been proposed as a new architecture, based on the Attention mechanism to analyze sequential data [24]. The Attention mechanism allows the Transformer to access any part of the sequence regardless of the distance between each item, making it potentially more suitable for grasping the recurring patterns with long-term dependencies [23]. In contrast to recurrent models such as RNNs and LSTMs, which process the input and output elements sequentially, Transformers process all input simultaneously, enabling efficient parallel computation. They excel in handling real-world forecasting applications that involve both long- and short-term repeating patterns. For example, the trajectory of a vehicle within urban traffic networks shows patterns influenced by daily and hourly cycles. In such cases, accurately modeling long-term dependencies, like the impact of weekday rush hours and weekend traffic flows, is crucial to effectively predict the vehicle's trajectory. Identifying these patterns and understanding their influence on the vehicle's trajectory over time is essential for developing reliable prediction models [24][23].

2.3.3 Evaluation Metrics for Trajectory Prediction

There are several metrics for assessing machine learning performance in the context of trajectory prediction, but this subsection will explore a few key metrics for conciseness:

- **Accuracy:**

  This metrics evaluates how well and accurate the predictions generated by the model align with the true outcomes [25]. In trajectory prediction, accuracy could provide a general sense of the model's performance by considering both true positives and true negatives relative to all predictions. The formula for calculating the accuracy is:

  $$\frac{TP+TN}{TP+TN+FN+FP}$$

  Where TP/TN is true positives and negatives, and FP/FN is false positives and negatives.

- **Precision and Recall:**

  Precision[25] measures the ratio of true positives to the total number of positives predicted by the model, given by:

  $$\frac{TP}{TP+FP}$$

  Recall[25], on the other hand, measures the ratio of true positives to all actual positives, and it is calculated as:

  $$\frac{TP}{TP+FN}$$

  In trajectory prediction, precision could indicate how reliable the model's positive predictions are, while recall could show how well the model captures all relevant positives cases.

- **F1-score:**

  The F1-score[25] is the harmonic mean of precision and recall, which is calculated by:

  $$2 \cdot \frac{precision * recall}{precision + recall}$$

  It provides a single metric that balances the trade-off between precision and recall, which is important in trajectory prediction tasks where both false positives and false negative can have significant implications.

2.4 Adaptive Learning

As discussed in the previous section, Deep Neural Networks, such as RNNs and Transformers, excel at sequential prediction tasks. However, ensuring these models adhere to temporal properties in sequence prediction, especially when dealing with multiple variables and temporal features, remains a significant challenge. To address this issue, enhancing the models' ability to learn and refine their performance over time is crucial. One effective strategy is implementing a framework that optimizes model execution based on past results, a process known as adaptive learning. Adaptive learning refers to a method where the model adjusts its strategies or updates its parameters in response to changes in the underlying data patterns [26]. Unlike traditional machine learning, adaptive learning compels the underlying models to learn dynamically, introducing a level of responsiveness not typically seen in standard approaches [26].

2.4.1 Temporal Logic Model Checking

Temporal logic is a formal system used to reason about propositions and events over time. The most popular used temporal logic in computer science is the Linear Temporal Logic (LTL), which is very useful for specifying correctness properties in a system [27]. LTL specifies the correctness properties of infinite computations in reactive systems, such as safety, liveness, and fairness [27]. To achieve this, LTL follows the syntax and semantics of temporal logic by constructing LTL formulas from:

- A finite set of atomic propositions, for example, a, b, p, t.
- Basic logical operators, such as $\neg$ (negation) and $\wedge$ (conjunction).
- Basic temporal operators, such as $\Diamond$ (eventually) and $\Box$ (always).

Formalization using LTL can be employed to specify the inherent properties of the models utilized for predicting trajectories. Typically, the constraints specified by LTL are already known by the system or defined by users before prediction, such as constraints imposed by the physical world. For instance, an example specification for a trajectory prediction model might ensure that the model never predicts a position outside the road, as shown by the following syntax: $\Box$ *(vehiclePosition < gridSize)*.

In other situations, a fairness approach may need to be adopted by relaxing strict constraints in the model prediction. Consider the scenario where there is a destination, and eventually, that destination has to be reached. When this is going to happen is not determined, but some time in the future it has to. In such cases, the following syntax can be used $\Diamond$ *(vehiclePosition = destinationPosition)* which indicates that the predicted position will eventually be the goal.

3

Framework

This chapter describes the process of developing the framework to achieve the results presented in Chapter 5. The chapter starts by introducing the framework interface, including the environment, system, and safety specification. Following, the learning, which includes data generation, model training, and adaptability processes. Finally, the interpretability, where different methods of multivariate interpretability developed in this thesis are presented.

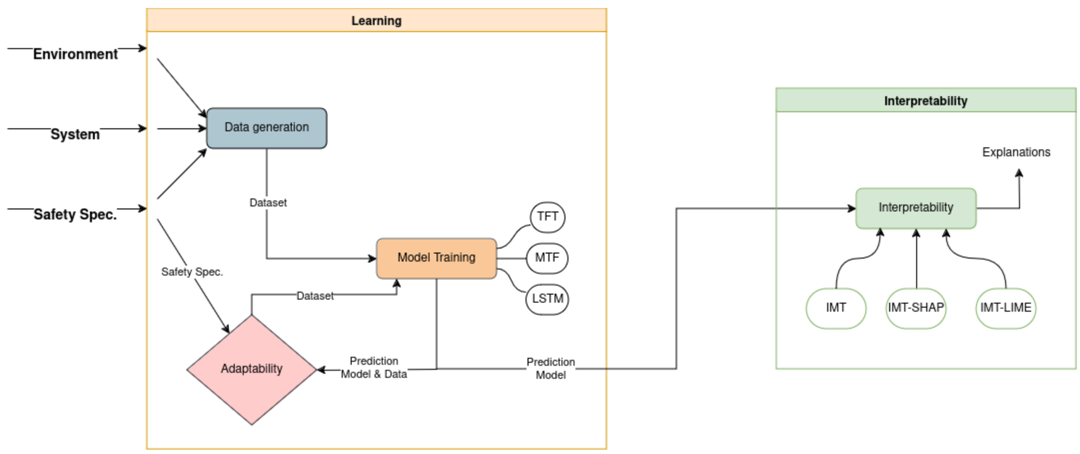

Figure 3.1: The workflow of the framework and its architecture, composed of two distinct blocks: the learning block, which performs data generation, model training, and model adaptability processes, and the interpretability block, where the results of the learning block can be interpreted.

Figure 3.1 displays the workflow of the framework and its architecture. The image shows two distinct blocks: the learning block and the interpretability block. Attached to the learning block are the framework interface inputs, which consist of the environment, the system, and the safety specification. In the following section, the features and properties of the framework are detailed, introduced, and specified.

3.1 Framework Interface Inputs

This section outlines the framework interface inputs, including the environment, system, and safety specification.

3.1.1 Environment

The environment is the map input, featuring details such as debris, walkers, or junctions. This is used as a backbone when learning the behaviors of the driver. Currently, the environment is represented by trip data. The trip data is a set of random drives around the map, exploring all possible roads. This is done by utilizing the probabilistic programming language Scenic, and the simulator Carla, which are described in Section 3.2.1.3.

3.1.2 System

The system is the driving history of a car. In the same format as the trip snippet from Figure 3.7. Because this thesis focuses on simulation-based approaches, the driving journal is also generated using the probabilistic programming language Scenic and the simulator Carla, similarly to what is done for the environment, this process is described further in 3.2.1.4. In cases where it is not possible to get the same frequency of data the learning process can interpolate to fill in the blanks, this can be done using tools listed on the OpenStreetMap (OSM) website [28]. The theory behind the interpolation is that some data between the original time steps will be lost. However, having the goal of a quite large prediction horizon, the original time steps will still be many enough for the model to make accurate predictions. The focus is rather on the behavioral patterns in the drives than the insignificant variations between the tenth of a second.

3.1.3 Safety Specification

The safety specification states the hard constraints that guide the learning process. The model will always learn and adapt the different behaviors, but never at the cost of violating the specification. Trips that violate the safety specification could be, for example, trips where the car crashed, drove against the road direction, stopped and parked illegally, etc. Most importantly, these kinds of trips are not passed to the model for training. Thus, a very rigid process of data cleaning is required.

The safety specification contains either simple invariants such as $distToCarFront \geqslant 10$ and the validation process, as presented in Section 3.2.3, would make sure that in all time steps the invariant holds. The safety specification can also contain temporal logic formulas, as introduced in 2.4.1. For example, we can write the invariant introduced here as the LTL formula, $\Diamond(distToCarFront \geqslant 10)$.

3.2 Learning

The learning is the part of the framework where given the environment, the system, and the safety specification, data gets generated and important features of route predictions are learned. In the following sections, data generation and model training are presented.

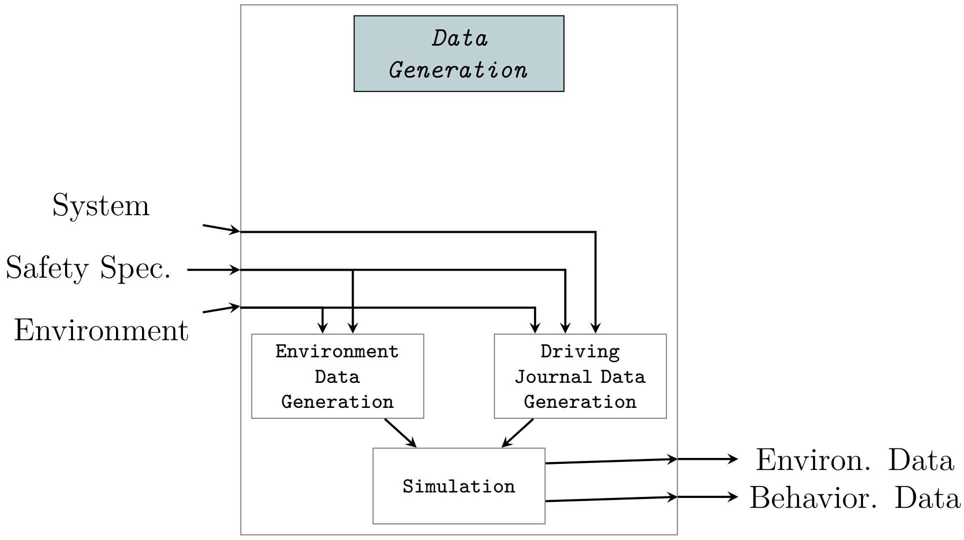

Figure 3.2: Internal structure of the Data Generation, with the inputs and outputs of the workflow

3.2.1 Data Generation

In this section, the process of generating data for the training is presented. The input to the data generation is the same as the input to the framework, and the output of the generation is the data that can be used for training, as can be seen depicted in Figure 3.2. Firstly, the tools used to generate data are explained. Secondly, the concrete details of how to generate environment data and simulate drives. Lastly, the details concerning generating and simulating drives that resemble a driving journal.

3.2.1.1 Carla

Carla [29] is an open-source simulator used for autonomous driving research. The simulation platform enables users to specify sensors, environmental conditions, map features, car features, and a lot more.

Carla has a very powerful, exposed, and flexible API, which means that end users can utilize it to configure and control all the properties of a simulation. Most of the focus for this thesis was the actors and the API that controlled them. Actors are explained by the Carla developers as entities that can interact with a simulation, for example, vehicles, pedestrians, and traffic signals. They have certain properties that are connected to them, which can then also be changed by the API. One can also wrap an actor in an agent. An agent is a script that controls the behavior of an actor, this way one can automate the driving. A destination can then be set by an agent and a behavior to follow when driving to that destination, for example, having an aggressive behavior when driving to a predefined destination home.

To generate the data needed, Carla was used as the simulator, running in the background. Scenic was used as the interface for communicating with the Carla API, the following section goes into detail about the programming language Scenic.

```
# badlyParked.scenic

ego = new Car
spot = new OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * Range(10, 20) deg
badParkedCar = new Car left of spot by 0.5,
    facing badAngle relative to roadDirection
```

Figure 3.3: badplyParked, a simple Scenic program that spawns a badly parked car on any visible curb from the observer

3.2.1.2 Scenic

Scenic [30][31][32] is a domain-specific probabilistic programming language. The core idea behind scenic is that you create a scenic program (a `.scenic` file) that is a distribution over scenes. Every time you execute this `.scenic` program, the compiler will generate a scene from those distributions. This makes it easy to create a large set of possible outcomes with a few lines, as shown in Figure 3.3.

What happens in Figure 3.3, is that firstly an ego car is spawned (in a lane selected uniformly from the map). Ego is the alias used for the vehicle that is observed. After that, a spot is found on any visible curb from the ego, and a new point is created. The angle is calculated uniformly in a certain range of degrees. Then a badly parked car is placed at the observed point, facing the angle relative to the direction of the road. This scenic program has an extreme amount of possible scenes that can be created from it. Figure 3.4 represents the first 4 examples of running the program and shows how powerful Scenic can be in the data generation domain. From a program containing a couple of lines, data that represents a car's trip and how it behaves whilst driving can be generated. The generation process can then be iterated over, and the results can be combined into something resembling a driving journal.

3.2.1.3 Generating and Simulating Environment Data

The environment, as specified in Section 3.1.1, is represented as a set of drives around the map. Figure 3.5 shows how these drives are generated using Carla and Scenic with a focus on capturing the spatial dependencies in a map, such as how segments are connected.

This program picks a point uniformly on the map and drives a random route, but terminates once it reaches 20 minutes of driving. The environment input (to the data generation), would be the map Town05 which is a Carla predefined map with roads and obstacles. To generate multiple drives, one can simply iterate over this program, and here follows the process of how to do that. Firstly, the process of running multiple scenic programs, gathering the data from the simulator, and storing the data appropriately needs to be automated. A Python script can be created, as displayed in Figure 3.6, which starts the simulator, loads a scenic program, and

Figure 3.4: Carla's output when executing the badlyParked scenic program

```
# noise.scenic
param map = localPath('Town05.xodr')
param lgsvl_map = "Town05"
model scenic.simulators.carla.model
behavior Drive():
  do AutopilotBehavior()

start = new OrientedPoint on Uniform(*network.lanes)
ego = new Car at start, with rolename 'hero',
                        with behavior Drive()

terminate when simulation().currentTime >= 12000
```

Figure 3.5: Scenic program that spawns a car that takes a random drive around the map

creates the scene.

```
# run.py
simulator = CarlaSimulator("Town05", "./Town05.xodr")
scenario = scenic.scenarioFromFile("./noise.scenic")
scenes, _ = scenario.generate()
simulation = simulator.simulate(scenes, maxSteps=1000)
```

Figure 3.6: Python script to run a Scenic program

Now the simulation object can be used to access recorded values in the simulation and the position of the ego vehicle in every time step. This is the core of the script. Then, a list of objects can be created, where every object represents a scenic program and how it should be executed. Iterate over this list and for every iteration parse the data coming from the simulation and store it in JSON files for later use. It is also during these iterations the simulation results get checked against the safety specification, and if the data generated ever violates the specification it does not get saved because the model should not learn bad behavior. An example of such bad behavior would be tailgating, where the specification would contain an invariant or LTL formula that states, for all time steps, the distance to the car in front should never be lower than a certain threshold, as can be seen in the example from 3.1.3. An example of how the generated data can look is seen in Figure 3.7. Other than the information that can be retrieved from the simulation, some have to be created as well, more specifically drives have to be bound by time. Thus, a Python library can be utilized to create a date for the drive, and also a time of when it started. Using the temporal information, the weekday of the drive can also be found, something that can be very helpful when making predictions later, because a lot of people's driving behaviors are related to time.

The environment-related data from Figure 3.7 is *x, y, z, heading, segment, segType, distToInter, distToCarFront, distToCarBehind*, and from this data, a model can learn multiple features of a map. For example, the combination of *heading* and *segment* results in *roadDirection* or something more simple like being able to derive *segment, laneId* from *x, y, z*. Having enough of this data yields a model with somewhat spatial awareness. Once there is enough generated data, it is passed through to the learning as the environment data.

3.2.1.4 Generating and Simulating The Driving Journal

To generate data that resembles a driving journal, the same process as described in the previous section about generating environment data can be used, but this time it is based on how real people behave and their driving habits. For simplicity, a driving persona was created to be used as a reference when constructing the scenarios. Consider the persona *driver*0, someone who drives very aggressively, and goes to work in the morning and home in the afternoon. Likes the restaurant in the city but does not like traffic. Figure 3.8 displays a scenic program that is specifically

```
...,
{
    "driverId": 0,
    "time": 2,
    "weekday": 2,
    "weather": 0,
    "x": 27.14455223083496,
    "y": 118.3385009765625,
    "z": -0.04205554723739624,
    "heading": 3.072747620416466,
    "velocity": 0.014725978638346732,
    "acc": 17.030845591088653,
    "segment": 14,
    "segType": "road",
    "laneId": 0,
    "distToInter": 13.756722932334695,
    "distToCarFront": 4.614102227156014,
    "distToCarBehind": -99
},
...
```

Figure 3.7: Example of a time step from the Scenic+Carla generated data

constructed for *driver*0, also included in the program the safety specification example from 3.1.3. Because of the safety specification, and even though the aggressive characteristics of the driver would influence their maneuvering, nothing of that will be learned by the model if the driver gets too close to the car in front. The scenic program is similar to the one for generating environment data, but this time focuses on driving from point A and finishing when arriving at point B, instead of finishing the drive after a couple of minutes. The process of running multiple scenic programs and saving them as JSON files can be automated as described in Section 3.2.1.3.

3.2.2 Model Training

The input to the model training is the result of the data generation, and Figure 3.9 depicts the entire flow of the model training. The figure illustrates the high-level description of its process, but more details can be found in Section 4.1.2.

```
# knownToRandom.scenic
home = (11032.56, 9251.50, 100.0)
work = (4921.83, -1257.49, 49.09)
restaurant = (-1738.01, 10474.20, 280.21)

locations = [home, work, restaurant]
times     = [600, 1200, 1800, 3600]
start_pos = Uniform(*locations)
t         = Uniform(*times)
end       = new OrientedPoint on Uniform(*network.lanes)
start     = new OrientedPoint on start_pos

behavior Drive():
    take SetBehaviorAutopilot(end, "aggressive")

ego = new Car at start,
    with behavior Drive(),
    with rolename 'hero'

record ((distance to distToCarFront) > 10) as safety
terminate when (distance from ego.position to end) < 10
    or simulation().currentTime >= t*10
record (ego.heading) as heading
record (roadDirection) as roadDirection
record (ego.velocity) as velocity
record (ego._road) as road
record (ego._lane) as lane
record (ego.distanceToClosestIntersection()) as distToInter
```

Figure 3.8: An example of a Scenic program that generates data related to specific driving habits

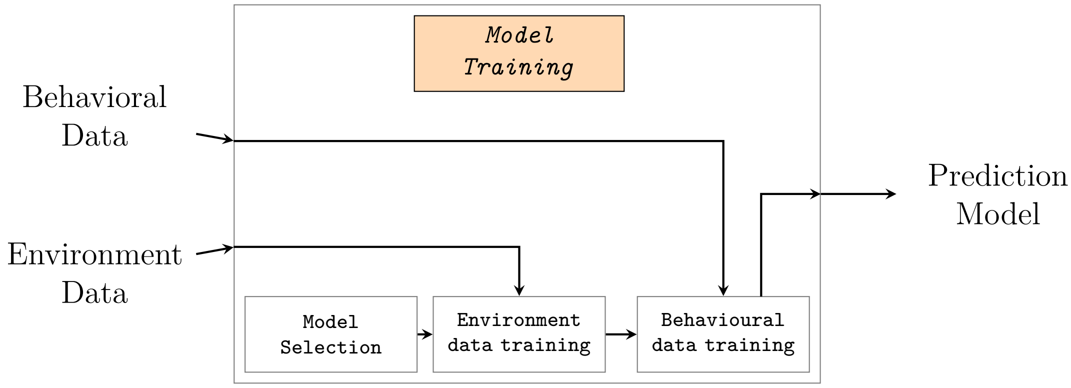

Figure 3.9: Internal structure of the Model Training, with the inputs and outputs of the workflow.

The **model selection** is the process of selecting a model that the training can be performed on. The currently supported models are the multivariate transformer, LSTM, and the temporal fusion transformer, with the possibility of being extended with models that inherit similar constraints and properties.

The **environment data training** is model training on the data from 3.2.1. The goal here is to give the model spatial awareness and force it to learn how roads and lanes are connected. This gives the model a robust base to learn behavioral patterns.

The **behavioral data training** takes the model after the initial environment training. It then lowers the rate of learning and tries to capture the relations and behavioral patterns of the input system, an example of such behavioral patterns can be seen in 3.2.1.4.

3.2.3 Adaptability

The flow of the adaptability is depicted in Figure 3.10. Again, consider $persona0$ first mentioned in 3.2.1.4. The input is the prediction model from the initial training or previous adaptability iterations, the safety specification, and the behavioral data that the model should adapt to. For $persona0$ that would be the initially trained model, the safety specification of not tailgating, and a new behavior, for example, *on weekdays, later than you leave for work, you drop off the kids at daycare.* Furthermore, the adaptability starts at the simulation by predicting and driving routes from and to the daycare. Then in the data validation, if any drives are found to fail the safety specification or predict badly, these are saved. The drives that did fail the safety specification will be used by the neighborhood analysis to generate new data for training, and the drives with bad predictions will be added to the newly generated data. More details about the implementation of the adaptability can be seen in Section 4.1.3, but following this paragraph is a high-level description of the internal parts of the adaptability.

The **simulation-based analysis** is the process of testing the model on the data it should adapt to. This is done by letting the model, firstly, predict routes and then try to drive those routes, secondly, predict routes and compare them to the ground

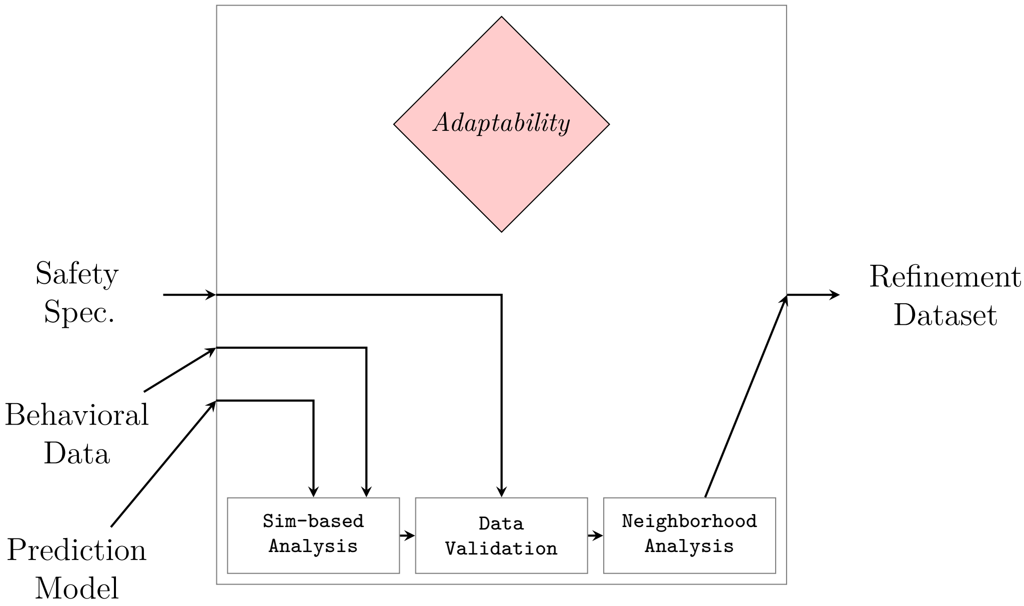

Figure 3.10: Internal structure of the Adaptability, with the inputs and outputs of the workflow

truth route. The results are analyzed and saved before being passed forward to the validation.

The **data validation** is the data cleaning process, where the results from the previous step are iterated through and the trips that violate the safety specification are identified. The routes that failed the validation or had bad predictions are sent forward.

The **neighborhood analysis** is the last process of adaptability. Such analysis, in our context, is merely looking at *neighboring* segments to the identified problematic ones. The segments found from the analysis can be used to generate a part of our refinement dataset using Carla and Scenic. The other part comes from the failed predictions, by combining the two parts, our refinement dataset is obtained.

3.3 Interpretability

When considering interpretation methods for multivariate input and output, not much research has been conducted. Even many black box explainers cannot capture the importance of such data without significant modifications. In this section, Interpret Multivariate Timeseries (IMT) is introduced, to address the challenges faced by established interpretability methods. Additionally, IMT-SHAP and IMT-LIME will also be introduced, two interpretability methods created by adapting existing interpretability frameworks. Figure 3.11 depicts the workflow of the interpretability. The process begins by selecting an interpretability method, subsequently, the relevant model information is passed to the interpretability method, such as the loss and predict function. Finally, explanations are obtained by passing unpredicted data to the interpretability method. Again, consider the example of $persona0$ from Section

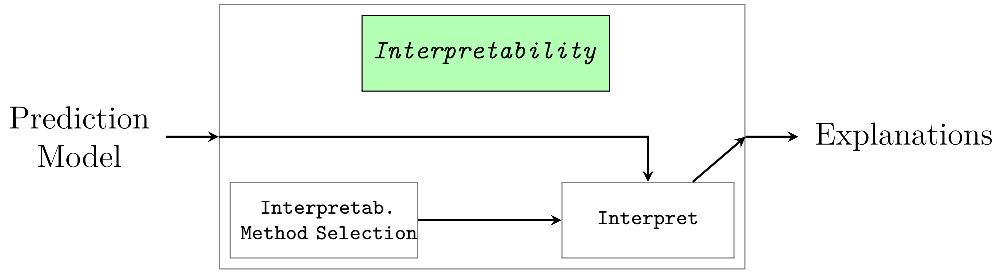

Figure 3.11: Internal structure of the Interpretability with the input and output of the workflow.

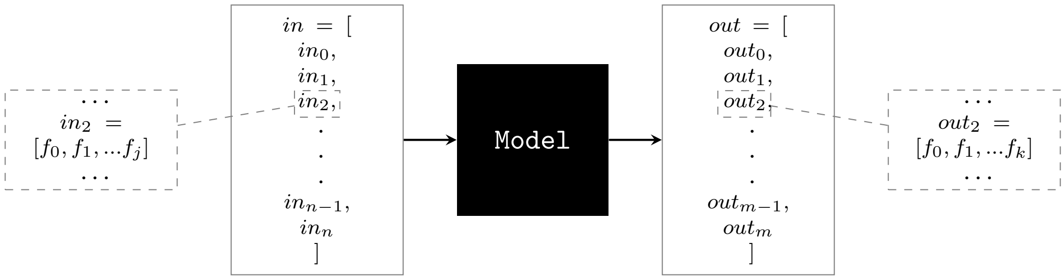

Figure 3.12: Illustration of multivariate input and output on a black box model

3.2.1.4 after the adaptability process is done. An explanation of a prediction that suggests a route to the daycare could indicate an important feature that represents how many people are in the car, the weekday, or the time of day.

3.3.1 Interpret Multivariate Timeseries

In this section, IMT is introduced. A method for interpreting multivariate models.

3.3.1.1 Masking & Permutations

Our chosen method of interpretation is usually referred to as permutation feature importance [33]. Predicting a baseline, then permuting the input and performing another prediction, allows the two predictions to represent the importance (relative loss) of that permutation with the help of a loss function. However, the focus will not only be on using permutations but also masking different combinations. The theory behind it is loosely based on the Permutation Explainer created by the developers of SHAP [12] but adapted to suit our needs of being able to capture both temporal and spatial relations. Figures 3.12, 3.13 and 3.14 are examples of how an explanation can be calculated.

After gathering the two predictions out and out_p, from 3.12 and 3.13 the loss can be calculated with $loss(out, out_p)$ to see what impact the masking had on the model being able to achieve the baseline prediction again. If it is high, then time step in_2 is important. However, because it is hard to set a threshold of how significant a loss is, the focus is on the relation between losses of different masks. That is, if the same masking process is done for every $in_0, in_1, ...in_n$ and the result is plotted, the significant time steps can be determined by their relative loss.

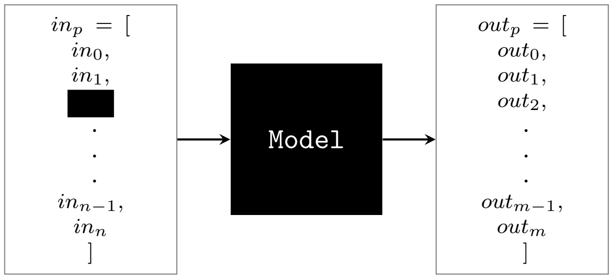

Figure 3.13: Illustration of masking an inputted time step to a black box model

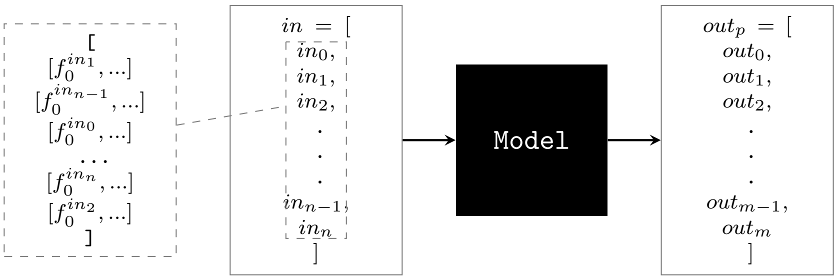

Figure 3.14: Illustration of permuting an inputted feature to a black box model

Now to permutations, looking at Figure 3.14 the order of f_0 has changed since 3.12. Calculating $loss(out, out_p)$ again, gives the impact the permutation had on being able to achieve the baseline prediction. If it is high, then the order of f_0 matters and f_0 is most likely an important feature. If it is low, it is most likely a less important feature. But, just as with the masking, the relative importance is what matters and the same process has to be done for all the features.

3.3.1.2 Modularity and Extensibility

Modularity and extensibility are two very essential features of a framework. Because the aim is to work with the very vague description of "multivariate time series" it is not possible to rely on all models being the same. That was also the motivation behind making the framework post-hoc and completely model-independent. As can be seen by looking at Figure 3.12, 3.13 and 3.14, the model in the middle is completely interchangeable.

The only thing that the framework demands of the user is consistency in the size of input and outputs. It has to be in the form of $input.shape = (n, j)$ and $output.shape = (m, k)$, in more general terms:

$$input.shape = timesteps\_in \times features\_in$$
$$output.shape = forecasted\_timesteps \times target\_features$$

This is a hard constraint, but because you pass along the prediction function as an argument the user can write the function themselves. In that function, they can do the necessary reshaping before and after they call their model's internal prediction function.

3.3.1.3 Explanations

In this section, the power of IMT is illustrated by displaying the implemented interpretability plots, with every plot comes a short explanation and description.

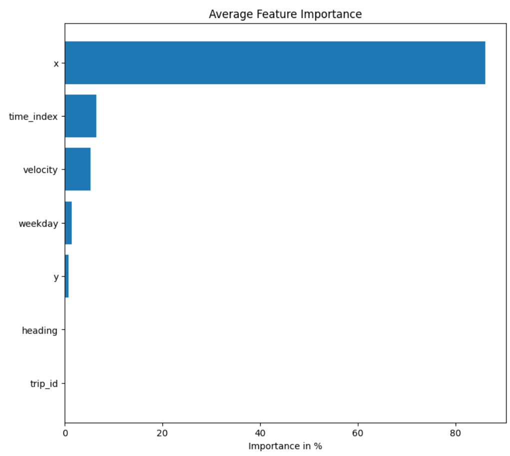

Figure 3.15: Averaged feature importance across all input time steps

The averaged feature importance in Figure 3.15 is calculated by masking out every feature for all input time steps. The importance is then calculated using the loss function. Thus, the result is all features impact on the output. This information gives insight into what features were important for a certain prediction.

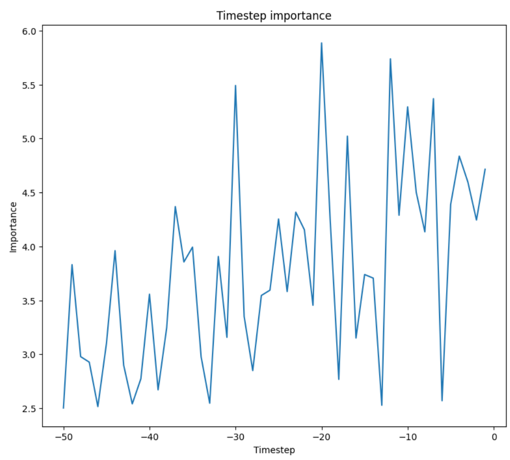

Figure 3.16: Input time step importance

The time step importance in Figure 3.16 is calculated by masking the entire time step, every masking or permutation gets evaluated against the baseline prediction. The result is how important a specific time step is to the output. This information can be used to see when in the past was most important for a certain prediction.

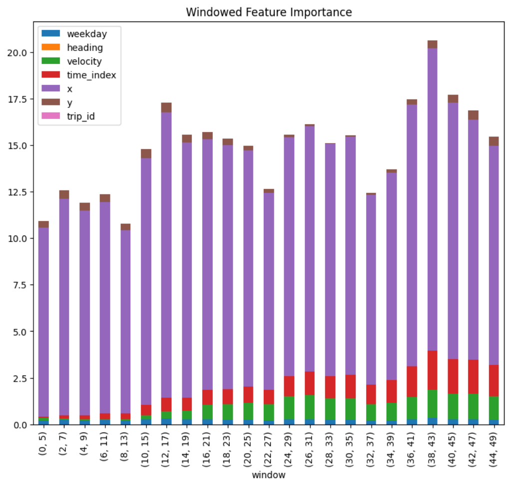

Figure 3.17: Sliding window segmented feature importance

The sliding window feature importance in Figure 3.17 is calculated by taking the user-provided window size and stride. The input gets segmented into windows in those windows the features get masked, their importance calculated, and evaluated. After the evaluation, the stride is used to find the next segment of interest. Performing this until there are no possible windows left. This information can be used to find what segment in the past was most important for a certain prediction.

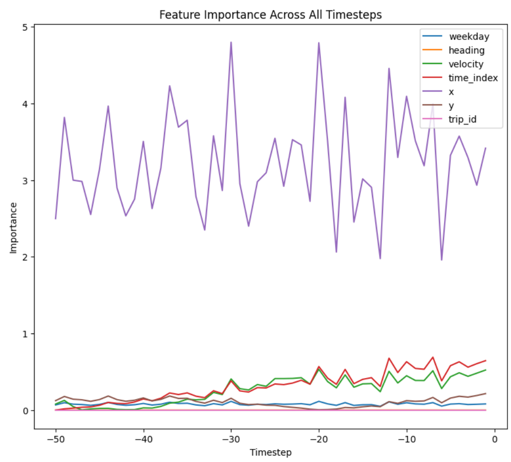

Figure 3.18: Importance of specific features across all time steps

The importance of specific features across all time steps in Figure 3.18 is calculated by creating a mask for every combination of feature and time step and evaluating against the baseline prediction. This is a very computationally heavy task, and creating the plot takes some time. Performing sliding window feature importance with $window\_size = 1, stride = 1$ is equivalent to this plot.

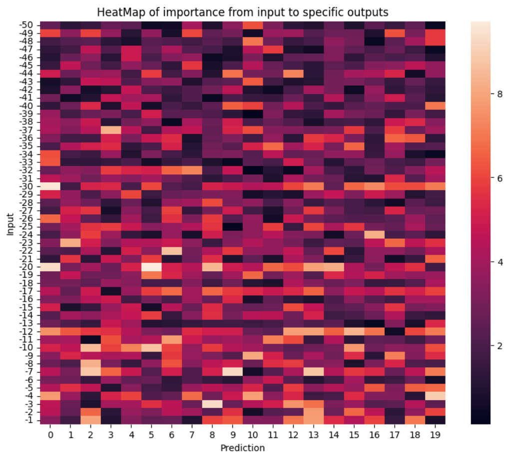

Figure 3.19: Heat map of time steps importance to specific predicted time step

The heat map in Figure 3.19 is created by iterating through all the input time steps, masking them, and evaluating them. Whilst the evaluation is happening, the impact the masking has on every outputted time step, one by one is saved. Then the matrix is created with size $input\_size \times output\_size$ where every box is the impact that the input time step had on the output time step. This information can be used to find how things in the past affected certain parts of the future, such as a right turn 5 minutes ago was the reason for the model predicting another right turn in 2 minutes.

3.3.2 Established Interpretability Techniques

In this section, IMT-SHAP and IMT-LIME are introduced, both interpretability methods derived and created from their respective original framework. These methods required significant modifications to accommodate higher dimensionalities and multi-horizons. These modifications are introduced below and then presented in Chapter 4. Our approach was to gain a deep understanding of LIME and SHAP and subsequently adapt them to suit our specific task requirements.

3.3.2.1 IMT-SHAP

To derive a SHAP approach that aligns with the concept behind IMT, which involves distributing the prediction score of a model for a specific input to its base features, we began by evaluating Equation 2.2 as the basis for designing our the algorithm for IMT-SHAP as described in 4.2.2.1.

IMT-SHAP, as illustrated in Figure 3.20, offers valuable insights into how individual features influence the model's prediction. However, the additional insight provided is not only identifying the features that influence the model's prediction but also understanding how they influence it. A negative Shapley value suggests that a feature has a high negative influence on the prediction, as observed for the feature *pos.x* in Figure 3.20. In contrast, *pos.z* makes a highly positive contribution to the prediction.

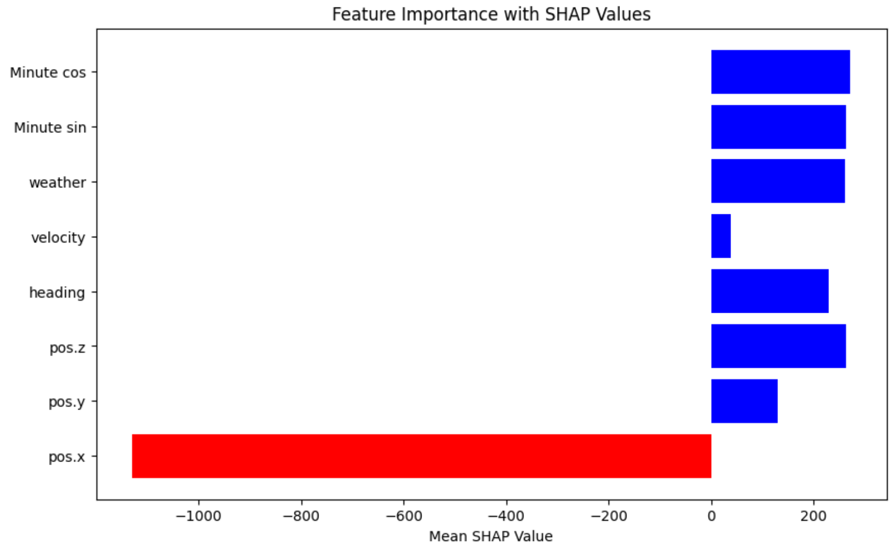

Figure 3.20: IMT-SHAP is used to explain feature attribution for a given prediction. The Shapley values for each feature are computed, with positive values (e.g., pos.y) in blue indicating a positive influence on the prediction, and negative values (e.g., pos.x) in red indicating a negative influence.

Formally, both having a high positive Shapley value or a high negative Shapley value means that the feature is important for the prediction and that changes in the value of this feature impact the model's predictions. If the aim is to only determine the importance of features, it could be sufficient to consider the absolute value of the Shapley value for all features. However, understanding how a feature impacts the prediction could be important in some scenarios. For example, in a model predicting house prices, if the feature "number of bedrooms" has a positive Shapley value for a specific house, it could mean that having a certain number of bedrooms increases the predicted price of that house. Conversely, if the same feature has a negative Shapley value, it could mean that having a certain number of bedrooms decreases the predicted price of that house.

3.3.2.2 IMT-LIME

To derive a LIME approach that aligns with the concept behind IMT, we began by exploring the components of Equations 2.1 and the LIME algorithm, as outlined in [11]. We then analysed how to implement each part to fit multivariate time series forecasting. This process led to the algorithm depicted in 4.2.2.2.

As illustrated in Figure 3.21, similar to IMT-SHAP, IMT-LIME provides the contribution effects of each feature to the prediction, where a positive coefficient indicates a positive effect of the feature on the prediction, while a negative coefficient suggests an inverse relationship.

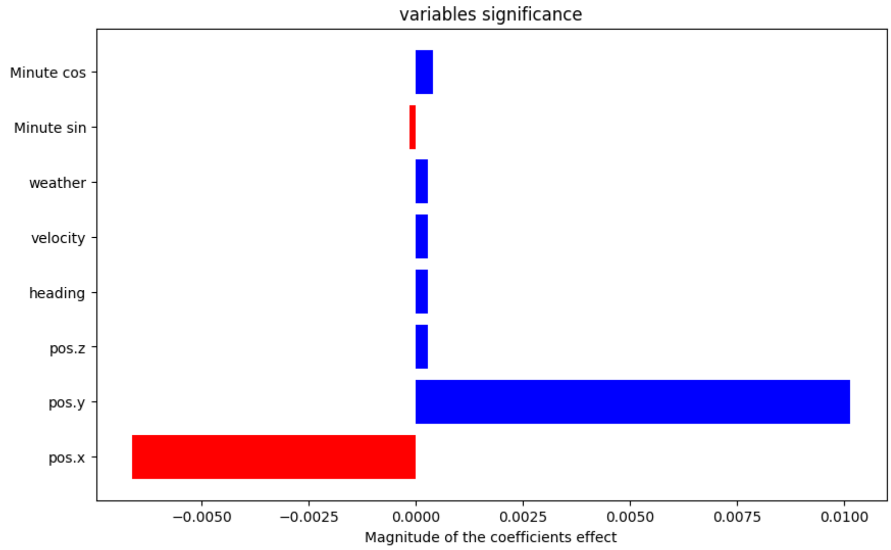

Figure 3.21: IMT-LIME is used to explain feature attribution for a given prediction. The effect values for each feature are computed, with a positive coefficient (e.g., pos.y) indicating a positive effect of the feature to the prediction and a negative coefficient (e.g., pos.x) suggesting an inverse relationship

4

Implementation

In this chapter, the implementations done to achieve the results in the following chapter will be presented. These implementations could range from slight modifications to already existing tools, to completely new tools created specifically for this purpose.

4.1 Learning

In this section, trajectory prediction and the tools and tool modifications used to perform it will be presented.

4.1.1 Scenic and Carla

Scenic and Carla are two fundamental building blocks for this thesis, but they are not perfect for our specific use case. Carla is massive, it has countless features, parameters, and sensors, and everything can be tweaked, monitored, and extended. Scenic, however, is a minimalistic programming language that enables communication with simulators, for example, Carla. Unfortunately, Scenic lacks full coverage. This is not inherently bad because both Scenic and Carla, as stated before, are open source. There were features of Carla we needed that scenic had yet to implement. Table 4.1 is the full table of features needed that were not implemented.

Feature	Carla	Scenic
Distance to cars around the ego	Yes	No
Set destination for Carla's autopilot	Yes	No
Behavioral Autopilot	Yes	No
Carla Agents	Yes	No
Distance to inanimate objects around the ego	Yes	Partially
Car data such as acceleration, throttle, braking, etc.	Yes	Partially
Headless Simulations	Yes	Partially
Open Drive generated maps from OpenStreetMap exports	Partially	Partially

Table 4.1: Table displaying features needed for data generation and if they are currently implemented in Scenic and Carla

Firstly, when analyzing the source code of both Scenic and Carla, some features were tightly coupled together, for instance, Carla's autopilot, the behavioral autopilot, and agents all rely on each other and need to be implemented together. The other features are mostly exposing a getter and can be as simple as a couple of lines of code. There are of course more features from Scenic and Carla that were used, but those were already implemented, and no modifications were needed. In the following paragraphs, the feature modifications done are presented, but also the importance and use-case of the features.

For the features *distance to cars around the ego*, *distance to inanimate objects around the ego*, and *car data such as acceleration, throttle, braking, etc.* only getters to scenic had to be exposed. Because the features were already implemented in Carla, the request with the relevant information is simply forwarded. This information is then recorded in every simulation time step, see the example of a time step in Figure 3.7.

For the features *Set destination for Carla's autopilot*, *Behavior Autopilot*, and *Carla Agents* there were some more modifications needed. All vehicles spawned with scenic are created as Carla Actors, and for the autopilot to work, the actors need to be wrapped in agents. Thus, the code is modified such that once the autopilot is activated for a car, it gets wrapped in an agent. This was the main modification needed, but some extra functionality from the agents was exposed through Carla's API. For instance, the set destination that autopilots to a destination.

For the feature *Headless Simulation*, a line of code had to be added in the Scenic main program loop such that Carla is forced to run in a no-rendering mode. This is most likely not yet implemented in Scenic because it came in one of the newer updates of Carla. Running in a no-rendering mode means that the simulations run quicker and use fewer resources.

The last feature needed was *Open Drive generated maps from OpenStreetMap exports*. Scenic and Carla can utilize the road description framework OpenDRIVE [34], which contains a high-level description of how roads are connected and road features, such as speed limit and road geometry. From these OpenDRIVE files, Carla and Scenic can generate simulations and graphical representations. For example, we want to export a part of Gothenburg using OpenStreetMap [35] and then generate an OpenDRIVE file from this export and run it with Scenic and Carla. Carla has a built-in converter called `osm2odr.py`, however, not all generated OpenDRIVE files can be run with Scenic. We resorted to third-party converters, which do not work great for larger exports, but smaller parts of Gothenburg worked fine.

To automate the process of generating data, Python scripts were created. These Python scripts manage everything needed with Carla and Scenic, starting simulations, gathering the data, and saving it in JSON files. In Algorithm 2, is the pseudocode representing the functionality of the script. The script is extremely valuable for data generation because a user can specify what scenarios to generate and let Scenic work with the probabilities and distributions over scenes. Enabling the creation of a lot of data with minimal interactions. The general flow of the automation is scenarios are loaded, iterated through and ran with Scenic and Carla and after every execution the data is parsed and saved.

Algorithm 2 Automation script for Carla and Scenic

```
Load scenarios specification from a separate file
Start the Carla simulator and connect to it
for each scenario s in scenarios do
    Parse scenario information and max amount of trips from s
    Create scenic scenario from s
    current_trip = 0
    while current_trip is less than max_trip do
        Create time and weather data
        Construct scenario s with Scenic and run the simulation
        if Simulation successfully ran s then
            Get the recorded data from the simulation
            Parse the data accordingly
            Save the data in a JSON file
        else                                ▷ Failed the constraints from the spec.
            Do not save data, report the error, and continue
        end if
    end while
end for
```

4.1.2 Model

In this section, the concrete implementations of the models are described, as what modifications have been done to existing architectures, and how these elements come together to enable multivariate trajectory prediction.

4.1.2.1 Temporal Fusion Transformer

We used the implementation of Temporal Fusion Transformer (TFT) from PyTorch Forecasting[36]. In this implementation, the team at PyTorch Forecasting used the information from the original paper [19] and created a PyTorch-based version of the model. This enables the users of the model to utilize other PyTorch functionality. However, there were some issues with the TFT model PyTorch implemented, as the loss function was hard-coded in the architecture, but this was a simple fix.

Using TFT for forecasting is not the simplest task. There is a lot of boilerplate to do for the model to be trainable. For instance, every feature has to be classified as either known or unknown, this means that if predictions are made for a specific horizon, are the values for the features known in every step of that horizon? An easy example of this is time, if the time delta between every time step is known then the time of day is known in every future time step, thus time is known.

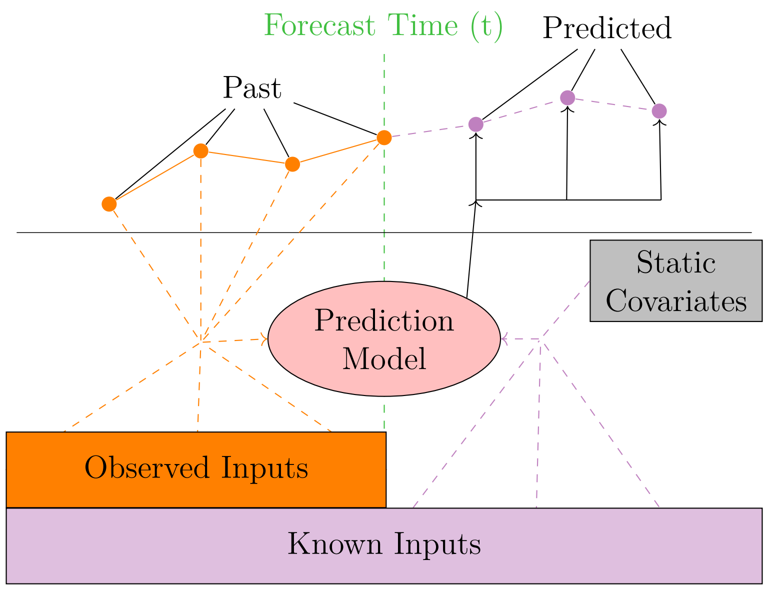

Figure 4.1: Structure of Temporal Fusion Transformer

From Figure 4.1, one can see that TFT is heavily reliant on the separation of features. And that separation is very important to make good predictions. It is important to note that the forecasting example from the Figure could be multivariate but from the graph, we are only plotting a single target output.

Observed inputs are the unknown features that will always contain data, but their values remain unknown until the corresponding time step has occurred. This could be features such as acceleration, velocity, or road segment.

Known inputs are the known features. They are known ahead of time, but they are not necessarily static. These features could be, for example, time, weather, or weekday.

Static covariates are also known features but do not change over time. Such features could be driverId or other static car-related information.

Using this information, the TFT model can be constructed, trained, and hyperparameters optimized using various tools available from PyTorch-forecasting.

TFT is not a regular forecasting transformer, it also has some interpretability embedded in the architecture. The first part of TFTs interpretability comes from variable selection blocks that are specifically designed to learn the global importance weights of the input features. This means that the feature importance of a prediction can easily be retrieved. TFT also has shared weights for all attention heads, which means that relevant time steps can easily be traced and the temporal importance for a prediction can easily be plotted, such as the time steps 20 to 30 seconds in the past were the most important to the prediction. In Figure 4.2, are two examples of interpretability outputs from TFT.

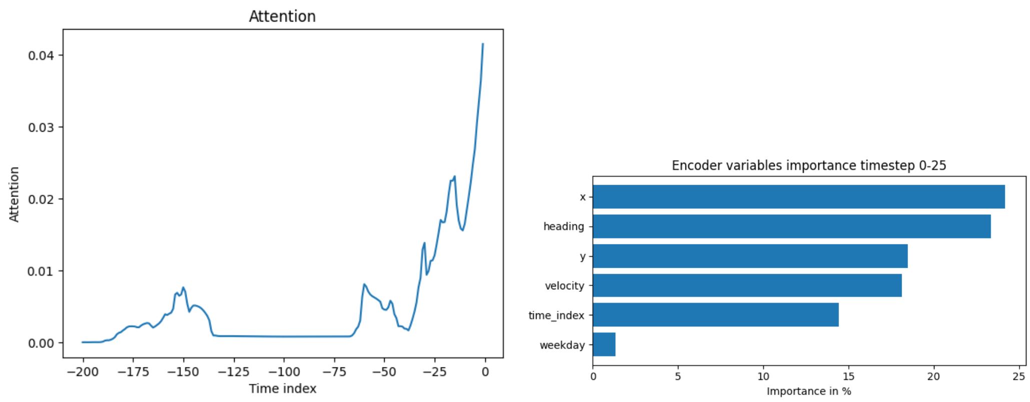

Figure 4.2: TFT Intrinsic Interpretability Examples

4.1.2.2 Multivariate Transformer

The Multivariate Transformer (MTF) is a special branch of transformers that allows for multivariate input and output. Combining this feature with time series, the result is something similar to TFT without the ability to classify different features, such as static covariates. The implementation used with the framework utilizes Keras [37], a deep learning API that runs above PyTorch or TensorFlow, and is a modification of the classic transformer, as seen in [24]. Figure 4.3 is the architecture of the regular transformer.

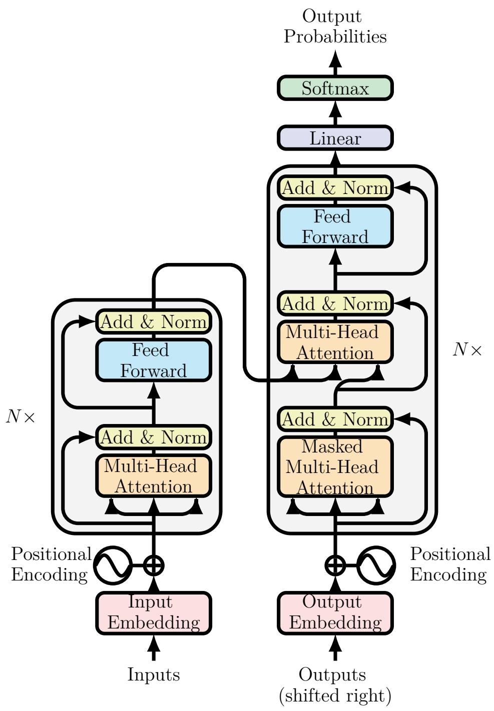

Figure 4.3: The architecture of the original Transformer presented in Attention Is All You Need [24]

The architecture of the transformer is very standardized and could be used as a core when creating the multivariate transformer. However, some modifications had to be done for the architecture to support the very specific needs of this thesis. For the transformer to capture multivariate input, the encoder, and decoder have to be modified such that they encode and decode multivariate time series.

4.1.2.3 LSTM

As mentioned in Section 2.3.1, LSTM, is a great machine learning model to use when forecasting and is often referred to as the predecessor to the Transformer. To implement an LSTM model, we relied on the information from the original paper [20] but also the implementation of an LSTM cell done by Keras[37]. This implementation contains the overhead for creating an LSTM cell with a specific size and how to chain and stack the cells to get a specific result. The architecture of a cell can be seen in Figure 4.4, this is an adapted version of the one in the original paper.

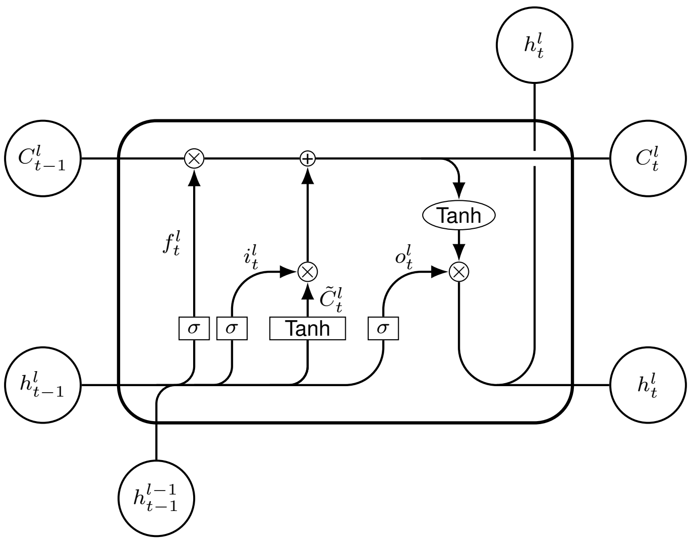

Figure 4.4: Architecture of a modified LSTM cell

Other than using the LSTM cell implementation of Keras, we created the rest of the model ourselves to get the very specific features needed for this thesis, i.e., multivariate forecasting. The implementation stacks multiple LSTM layers and is configured to be many-to-many.

4.1.3 Adaptability

The adaptiveness of the models, as described in Section 2.4, is introduced by what is called adaptability. The term is used to describe the process of refining, adapting, and further learning of a model after finishing training. This loop can be seen in Figure 3.1, and in more detail in Figure 3.10.

The concrete implementation of adaptability is a bit more complex. To achieve the adaptiveness, there is a need for continuous driving input. But, because the focus is simulation applications, that is used now as well. In Algorithm 3 is the pseudocode for the adaptability process. From the pseudocode, the algorithms start by importing and loading the already-trained model. After that, the data to perform adaptiveness is loaded. Then the algorithm iterates through all the drives and segments them into smaller windows that will later be used to drive and predict trajectory on. After all the predictions are obtained, the simulation drives them. If the driving fails, either by failing the safety specification or by being wrong, the roads that were predicted, and the correct roads, are used to generate more data in and around that area. This is done to force the model to completely capture the complexity of the roads, and the correct routing. When all the specified drives have been iterated through, the training of the model starts, now with a lower learning rate to not unlearn anything that is deemed good from the initial training. This introduces adaptability by forcing the model to learn behaviors.

Algorithm 3 Adaptability Process

```
Load the model and saved weights

for each drive s in drives do
    Parse s information
    Create driveable windows w_s of s
    Drive the w_s
    Parse data from the drive
    Predict the trajectory using the data from the drive
    Try to drive the prediction
    if Drive the prediction fails or is not correct then
        Save the roads along the prediction
        Generate driving data for the failed roads
    end if
end for
Retrain model with low learning rate on those failed predictions/drives
```

4.2 Interpretability

In this section, the modifications made to state-of-the-art interpretability techniques and the creation of a new multivariate interpretability framework are explained.

4.2.1 Interpret Multivariate Timeseries

As described in Section 3.3 where IMT first was introduced. IMT is a post-hoc approach to interpretability for multivariate machine learning models. The underlying math behind the interpretability is similar to other more established frameworks such as SHAP or LIME, but with the focus being directly working with multidimensional input and output.

The general idea of IMT is that when looking at a model as a black box where one can utilize the model's prediction function. An observer can use and modify inputs and see how that affects the output and based on the observations and changes in the output then score the importance of the change to the input, such as masking or permuting a feature over the entire input. In Algorithm 4 is the pseudocode for IMT.

Algorithm 4 Pseudocode for IMT

Require: Black-box model f. No. of features n,
Require: Time steps in $time_{in}$. Time steps out $time_{out}$
Require: Input to score x. No. of permutations p
Require: maskInput mi True/False, permInput pi True/false
  $importance\_scores \leftarrow [0, 0, \ldots, 0]$ ▷ shape = (n)
  $masks \leftarrow []$
  $mask \leftarrow [1, 1, \ldots, 1]$ ▷ shape = (n)
  $target \leftarrow predict(x)$
  for i in $\{0, 1, 2, \ldots, n-1\}$ **do** ▷ Gather all masks
    $mask\_copy \leftarrow$ copy of mask
    $mask\_copy\_feature \leftarrow mask\_feature(i, mask\_copy)$
    Append masks with $mask\_copy\_feature$
  end for
  for i in $\{0, 1, 2, \ldots, len(masks) - 1\}$ **do** ▷ Calculate mask and permutation importance
    if mi **then**
      $m \leftarrow masks[i]$
      $x_1, x_2 \leftarrow$ two copies of x
      $masked\_x \leftarrow m * x_1$
      $masked\_y \leftarrow predict(masked\_x)$
      $loss[i] += loss(target, masked_y)$
    end if
    if pi **then**
      for j in $\{0, 1, 2, \ldots, p-1\}$ **do**
        $xT \leftarrow Transposex_1$
        Shuffle the row $xT[i]$
        $perm\_x \leftarrow TransposexT$
        $perm\_y \leftarrow predict(perm\_x)$
        $loss[i] += loss(target, perm_y)$
      end for
    end if
  end for
  return $loss$ ▷ Feature Importance

From the pseudocode in Algorithm 4, it is evident that IMT consists of masking and permutations. This is also described in 3.3. The pseudocode is for the implementation of feature importance, thus, not covering the other plots of IMT in 3.3. However, the implementations are closely related and are more about changing how to iterate through the input. More precisely, the pseudocode relies on input from a system that requests an explanation, the system then sends a prediction model and some necessary information to calculate the importance.

For the masking part, the algorithm computes all the feature masks, that is, the masks that cover a feature. Then the feature is scored based on how close to the initial target the model gets after the masking, which is then the importance of that feature.

For the permutations, iterations are performed over all the features and over a set constant, which determines the number of permutations to be done. A column (with values of a feature) is then shuffled, and the result is predicted and evaluated against the target. Again, the features are scored based on how close the model can get to the target with the permuted input. The more permutations done, the more accurate the result will be. Because the chance of a single good permutation skewing the result decreases.

Depending on the input data, masking, and permutation can either be combined or run separately. This is important because, as stated in sections 4.2.2.2 and 4.2.2.1 not all features can be treated equally, if, for instance, there exists a classification feature, then masking the classes would be inappropriate because that masking could resemble a new class which would give an unfair importance score.

4.2.2 Established Interpretability Techniques

In this section, the modifications made to existing interpretability techniques, specifically SHAP and LIME, to adapt them for use in multivariate settings are explained.

4.2.2.1 IMT-SHAP

Algorithm 5 for IMT-SHAP takes a function f which represents the prediction model, that takes two-dimensional input and returns two-dimensional output. Additionally, the algorithm requires a list of features j and an input matrix X (representing the time series being explained). For each feature $1, ..., p$ in j, the algorithm starts by creating a coalitions' matrix M as illustrated in Figure 4.5.

This matrix M has a size of $2^{|j|-1}$ and enables the computation of the average contribution of a given feature j_i for all coalitions in M.

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Figure 4.5: A binary coalition matrix with three features, where 1 indicates the feature is in the coalition and 0 indicates the feature is not in the coalition.

Consider the input matrix X with a shape of [timesteps, features], where *timesteps=5* and *features=3*. Assuming, the IMT-SHAP algorithm starts by computing the aver-

Algorithm 5 IMT-SHAP based on classic Shapley values of game theory

Require: Black-box model f. Input instance X . Original prediction y
Require: List of features j. Loss function $L(w, f(X))$ (Mean square error - MSE)
Require: Weight function $w(j)$
Ensure: Average Contribution
  $average\_contribution \leftarrow []$
  for each feature j in $\{1, ..., p\}$ **do**
    $marginal\_contribution \leftarrow []$
    $v \leftarrow$ Generate matrix of feature combinations of size $2^{|N|-1}$
    for each combination i in v **do**
      $X_{perm\_without\_j} \leftarrow$ Permute values of features not in combination i from X
      $X_{perm\_with\_j} \leftarrow X_{perm\_without\_j} \cup \{j\}$
      $e_{perm\_without\_j} \leftarrow L(y, f(X_{perm\_without\_j}))$
      $e_{perm\_with\_j} \leftarrow L(y, f(X_{perm\_with\_j}))$
      $w_i \leftarrow w(j)$
      $mc_i \leftarrow w_i \cdot (e_{perm\_with\_j} - e_{perm\_without\_j})$
      $marginal\_contribution \leftarrow marginal\_contribution \cup \{mc_i\}$
    end for
    $avc_j \leftarrow \text{mean}(marginal\_contribution)$
    $average\_contribution \leftarrow average\_contribution \cup \{avc_j\}$
  end for
  return $average\_contribution$

age contribution of the first feature, *Feature1*. The first row in matrix M indicates that permutation will be applied for *Feature1* and *Feature2*. For the initial coalition (the first row in M), Two distinct matrices (combinations) will be created, X^1 and X^2, deviated from matrix X, as the Figure 4.9 Shown below:

$$\begin{pmatrix} v1_0 & v2_0 & v3_0 \\ v1_1 & v2_1 & v3_1 \\ v1_2 & v2_2 & v3_2 \\ v1_3 & v2_3 & v3_3 \\ v1_4 & v2_4 & v3_4 \end{pmatrix}$$

Figure 4.6: Matrix X

$$\begin{pmatrix} p & p & v3_0 \\ p & p & v3_1 \\ p & p & v3_2 \\ p & p & v3_3 \\ p & p & v3_4 \end{pmatrix}$$

Figure 4.7: Matrix X^1

$$\begin{pmatrix} v1_0 & p & v3_0 \\ v1_1 & p & v3_1 \\ v1_2 & p & v3_2 \\ v1_3 & p & v3_3 \\ v1_4 & p & v3_4 \end{pmatrix}$$

Figure 4.8: Matrix X^2

Figure 4.9: Matrix X represents the original input matrix. Matrix X^1 represent the combination excluding *Feature1*, while X^2 represents the combination included *Feature1*. p indicates the permutation value for features that are not in the combination.

After obtaining these distinct matrices, and since the prediction of the matrix X is known, the algorithm computes the predictions for the two remaining matrices, X^1 and X^2. Subsequently, the algorithm uses the mean square error function to determine the average error between the original prediction and the prediction made without *Feature1*, denoted as *mse[f(X), f(X^1)]*. Similarly, it calculates the average error between the original prediction and the prediction made with the inclusion of *Feature1*, denoted as *mse[f(X), f(X^2)]*. In this step, with an understanding of how these combinations influence the predictions, the Equation 2.2 is used to compute the marginal contribution of *Feature1*:

$$W(\text{Feature1}) \times (mse[f(X), f(X^2)] - mse[f(X), f(X^1)])$$

The function *W()* determines the weight of *Feature1* based on the following corresponding part of Equation 2.2:

$$\frac{|S|!(P-|S|-1)}{P!}$$

This process is repeated for all coalitions in *M*, and the mean of these values is saved as the Shapley value of *Feature1*. Similarly, the process will be applied to the rest of the features.

The permutation method, which is used to mask the values of features not in the coalition, involves randomly shuffling the values of a single feature. For instance, shuffling the value of *Feature1* for the first data point $[t, ..., t+4]$ and the second data point $[t+5, ..., t+9]$. However, this assumes access to more data than just the prediction being explained. But what if we only have access to the data for the prediction itself? Can the time steps within the first data points simply be shuffled, like $[t+4, t+2, t, t+3t+1]$?

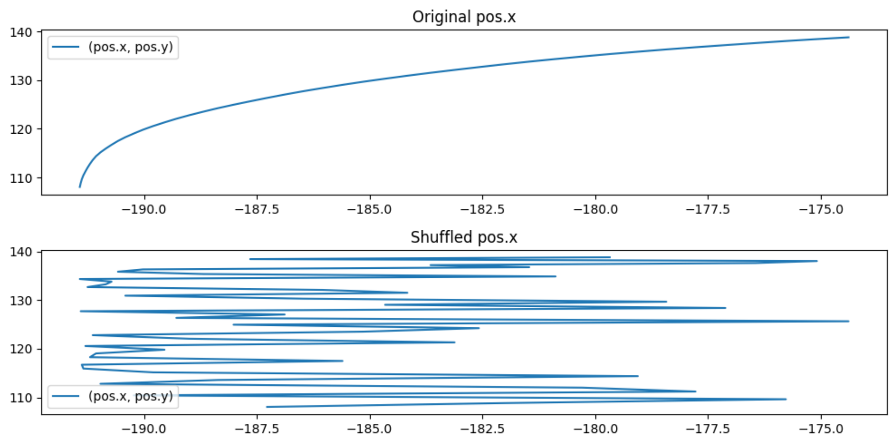

Figure 4.10: Randomly shuffling the values of a feature (e.g., pos.x) within the same sequence of timesteps. The first plot shows the original data of the given feature before permutation. The second plot shows the result of the random permutation, resulting in unrealistic data.

As shown in Figure 4.10, shuffling the time steps within the data points can lead to unrealistic data. In this case, the longitude and latitude coordinates of the car show a normal trajectory in the original dataset, but when one of these features is shuffled, the result displays a highly unrealistic movement of the car.
This issue can be explained as the following, imagine a model that predicts a patient's BMI based on their height, weight, and age. If we want to assess the importance of age in the prediction and consider a patient with *age=20*, *weight=80*, and

height=1.80, shuffling the age value with that of a patient who is 5 years old would result in an unrealistic data (*age=5*, *weight=80*, and *height=1.80*). This could potentially lead to a lower reported importance value for the age feature, even though it might be significant in predicting BMI.

The strategy used involved exploring correlations within the set of features and then designing an algorithm that performs permutations without generating unrealistic data. Figure 4.11 displays the varying correlations among different feature combinations. Notably, velocity shows a valuable correlation with vertical position (pos.z), which is logical given scenarios where changes in altitude affect vehicle speed. However, while correlations exist between other features, they appear to be less pronounced. It is worth noting that these observations could vary depending on the collection of data points being analyzed.

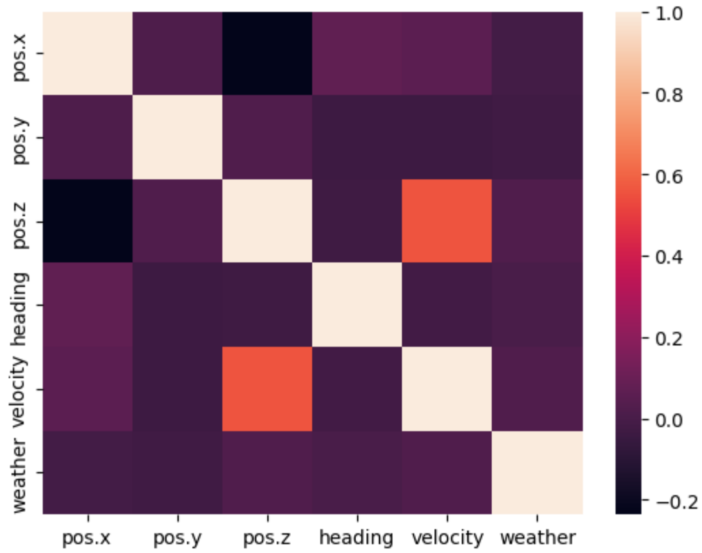

Figure 4.11: Heatmap showing correlated features from a dataset of car driving history, including features such as position, velocity, weather, and more. Velocity shows a significant correlation with pos.z, while pos.z doesn't seem to be correlated with pos.x.

To address this issue, we have proposed Algorithm 6, which performs permutation by analyzing correlations within features and generates perturbation values based on these correlations. The algorithm starts by computing the correlation matrix of the input matrix using the Pearson correlation coefficient. Then, it iterates over each feature in the matrix and identifies highly correlated features (correlation coefficient between 0.7 and 1.0). For each highly correlated feature, it computes a perturbation value based on the correlation grade between the features. However, for features that are not highly correlated with any other feature, noise perturbation is added to introduce variation in their values.

Algorithm 6 Permutation Algorithm for Generating Perturbation Values

Require: Matrix X
Ensure: Perturbations Matrix
  function GeneratePerturbations(X)
    $corr\_matrix \leftarrow$ CalculateCorrelationMatrix(X)
    $pertubations\_matrix \leftarrow []$
    for each feature i **in** X **do**
      Find indices of features with high correlation with feature i
      Compute perturbation_values based on correlation grade
      $pertubations\_matrix$.append($pertubation\_values$)
      if maximum absolute correlation value for feature i is ≤ 0.7 **then**
        Multiply values of feature i with its mean to add noise perturbation
      end if
    end for
    return $pertubations\_matrix$
  end function

Overall, Algorithm 6 ensures that perturbation values are generated in a way that reflects the underlying relationships between features in the time series data. As displayed in Figure 4.12, a modified trajectory is observed resulting from perturbed data. Unlike Figure 4.10, the trajectory now appears more realistic, enabling the model to accurately compute the contribution of each specific feature. However, in cases where features are a classification, such as the ID of the driver or the weather, this algorithm will come up with values that are not in any class and thus if the model does not allow unknown classes, will fail the predictions. To combat this issue, an argument was added to the IMT-SHAP initialization that lets the user specify the features that are allowed to be modified, thus also usable by IMT-SHAP.

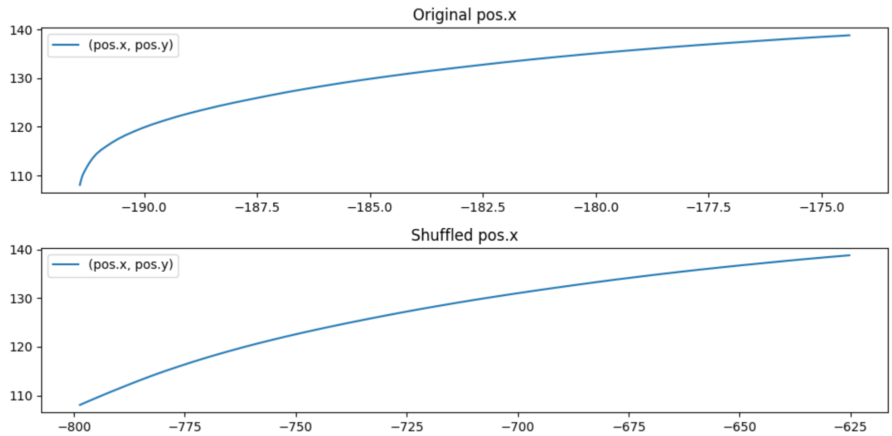

Figure 4.12: Randomly shuffling the values of a feature (e.g., pos.y) by using the permutation algorithm, Algorithm 6. The first plot shows the original data of the given feature before permutation. The second plot shows the result of the permutation, illustrating realistic data and demonstrating the efficacy of Algorithm 6.

Lastly, to avoid creating a coalition matrix M of size $2^{|j|-1}$ for all feature $j_i, (i = 1, ..., n)$, a sampling approach is employed to approximate the Shapley values. This method significantly reduces the computational complexity by randomly generating a fixed number of subsets for each feature instead of evaluating all possible subsets. This approach captures a wide range of feature interactions and provides a robust approximation of the feature importance while maintaining computational efficiency. This sampling method not only ensures scalability to high-dimensional data but also converges to the true Shapley values as the number of samples increases [12].

4.2.2.2 IMT-LIME

Algorithm 7 for IMT-LIME, begins by generating synthetic data samples around the original data, similar to the approach outlined in [11]. Next, it calculates similarity scores by computing the Euclidean distance between the original data and the permuted data, with a lower value indicating closer proximity to the original data and a higher value indicating greater distance. The algorithm then evaluates predictions for the new data using the black-box model.

Algorithm 7 IMT-LIME based on classic LIME in [11]

Require: Black-box model f. Number of samples N
Require: Instance x. Similarity kernel π_x (Mean square error - MSE)
Require: Surrogate interpretable model V (Vector Auto Regression - VAR)
Ensure: Vector Effect Matrix

```
similarity_vector ← []
new_samples ← []
ten_best_samples ← []
vector_effect_matrix ← []
for i in {1, 2, 3, ..., N} do
    z_i ← sample_around(x)
    S_i ← π_x(z_i)
    new_samples ← z_i
    similarity_vector ← S_i
end for
ten_best_samples ← select_ten(similarity_vector,new_samples)
for each sample j in ten_best_samples do
    MSE(f(x), f(j))            ▷ Evaluating the black-box with perturbed data
    exp ← V(j)                 ▷ Geting effect matrix for sample j
    MSE(f(j), V(j))            ▷ Evaluating the surrogate approximation
    exp ← vector_effect_matrix
end for
return vector_effect_matrix
```

Subsequently, the algorithm identifies the ten samples with the closest similarity scores and high prediction accuracy from the black-box model. These samples are selected for further analysis. The algorithm proceeds by fitting a Vector Auto Regression (VAR) model to the ten best-permuted data samples. Finally, the vector effect matrix computed by the VAR model, which quantifies how much a unit change in one variable at a previous time step (or steps) affects another variable at the current time step, is employed to provide explanations for the local behavior of the complex model.

As outlined in Algorithm 7, two important aspects need to be addressed to efficiently implement IMT-LIME such as determining the best approach for sampling data around the instance to be explained and selecting a suitable surrogate interpretable model for multivariate time series tasks. Starting with the first step, which involves creating samples around the given data, we could start by considering Algorithm 6 explained in the previous section for SHAP. However, for SHAP, the need to mask one feature at a time was essential for generating perturbed data for that feature while preserving correlations with other features and avoiding unrealistic data. Here, the aim is to avoid unrealistic data, but there is a need to create new samples around all features simultaneously. The trivial approach we adopt, as shown in Figure 4.13, involves multiplying the values of all features by the same random perturbation factor. For instance, given an input data *y={x, y, z, ..., heading, velocity}*, the new data around this could be obtained by *2*y*. However, again, this leads to the same issue as is present in IMT-SHAP. Multiplying values by some random perturbation factor assumes that the values are continuous, which they are not for classifications. This is fixed by the same argument as in IMT-SHAP where there is a set of modifiable features, unfortunately, this means that not all features can have an importance using this technique.

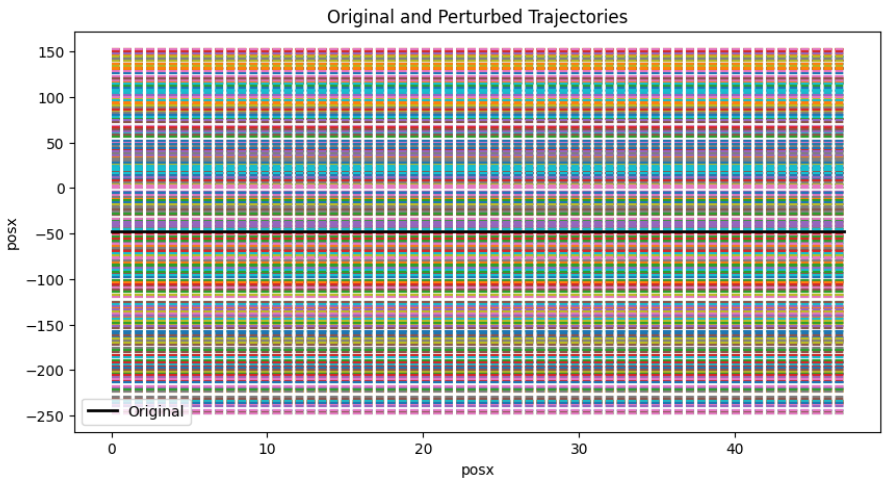

Figure 4.13: Sampling new data randomly around the values of a feature (e.g., pos.x). This process helps IMT-LIME create a range of new values around the value of the original feature that can be analyzed later to assess how small changes in the input data impact the prediction.

Equation 2.1 suggests that the surrogate interpretable model g needs to maintain a sufficiently low complexity for interpretability ($\Omega(g)$). One method deemed suitable for addressing the challenges of multivariate time series is the Vector Auto Regression model (VAR) [38]. The model's general equation can be expressed as follows:

$$Y_t = c + A_1Y_{t-1} + A_2Y_{t-2} + ... + A_pY_{t-p} + e_t \tag{4.1}$$

where:

- Y_t is a vector that contains the present values of each of the variables.
- p is the order of the VAR model. It determines the number of time steps back that are used for predicting the future.
- A is a vector of coefficients.
- e is the error value.

The model simplifies the complexity of the black-box by capturing the linear interdependencies among the time series data. Consider explaining the following input data, which consists of three features *(x1, x2, x3)* and extends over two time steps, intending to forecast one additional time step:

$$\mathrm{X} = \begin{pmatrix} x1_{t0} & x2_{t0} & x3_{t0} \\ x1_{t1} & x2_{t1} & x3_{t1} \end{pmatrix}$$

Figure 4.14: An example of input data with three features *(x1, x2, x3)*, and extends over two time steps.

In the example above, and according to Equation 4.1, the VAR model is assumed to have order $p = 2$. Using the values from $t0$, and $t1$, the forecast for $t2$ is determined by the following specific equations:

$$\begin{aligned}
x1_{t2} &= c_1 + A^1_{11}x1_{t0} + A^1_{12}x2_{t0} + A^1_{13}x3_{t0} + A^2_{11}x1_{t1} + A^2_{12}x2_{t1} + A^2_{13}x3_{t1} + u_{t1},\\
x2_{t2} &= c_2 + +A^1_{21}x1_{t0} + A^1_{22}x2_{t0} + A^1_{23}x3_{t0} + A^2_{21}x1_{t1} + A^2_{22}x2_{t1} + A^2_{23}x3_{t1} + u_{t2},\\
x3_{t2} &= c_3 + +A^1_{31}x1_{t0} + A^1_{32}x2_{t0} + A^1_{33}x3_{t0} + A^2_{31}x1_{t1} + A^2_{32}x2_{t1} + A^2_{33}x3_{t1} + u_{t3},
\end{aligned}$$

As the equations above demonstrate, the VAR model describes a relationship between the past and future values of each variable. Each variable is predicted based on past values of itself and past values of all other variables in the system. Then, the model uses the lag structure (the order p), to specify the number of past observations for each variable to predict future values. From these equations, the VAR model extracts the effect values $(A^1_{11}), ..., (A^2_{33})$, as shown in Figure 4.15, into coefficient matrices. These matrices describe the effect of variables at a specific lag (order).

$$A_1 = \begin{bmatrix} A^1_{11} & A^1_{12} & A^1_{13} \\ A^1_{21} & A^1_{22} & A^1_{23} \\ A^1_{31} & A^1_{32} & A^1_{33} \end{bmatrix} \qquad A_2 = \begin{bmatrix} A^2_{11} & A^2_{12} & A^2_{13} \\ A^2_{21} & A^2_{22} & A^2_{23} \\ A^2_{31} & A^2_{32} & A^2_{33} \end{bmatrix}$$

Figure 4.15: Coefficient matrices describing the effect of variables at lag 1 (A_1), and lag 2 (A_2)

Explicitly, A_{12}^1, as seen in Figure 4.15, represents the coefficient for the effect of variable x_2 at lag 1 on variable x_1 at the current time ($t0$). Basically, A_{12}^1 indicates how much a unit change in a variable x_2 affects the prediction of the variable x_1. Finally, if the variable x_1 is considered as the target variable being predicted by the black-box model, our approach is to use the coefficient matrices, and then compute the average effect value for each variable. For example, for the variable x_2, the average, $avg(A_{12}^1, A_{12}^2)$, indicates the average effect that variable x_2 has on predicting the target x_1 across each lag and implicitly each time step.

4.2.2.3 Overview of the Modifications

Comparing the LIME algorithm outlined in [11] with Algorithm 7 designed for IMT-LIME, we can observe similarities in their structure. However, to derive IMT-LIME from the original algorithm, We made the following specific modifications:

- **Data perturbation:**

  LIME generates synthetic data samples around the instance being explained. This functionality is tailored in the original LIME framework for a conventional ML context, where you have features with specific values, such as *numberOfRooms*. In time series, a feature consists of a sequence of values over time. For instance, *numberOfRooms* could be a sequence of values each second over an hour: 1, 3, 5, 5, and so on. To address this issue, we proposed the permutation algorithm for multivariate time series, that generates perturbations values, as outline in Algorithm 6.

- **Similarity score:**

  LIME computes the distance between the permuted data and the original instance, saving these similarity scores as weights. LIME employs a Euclidean distance function to measure the distance between points from the original instance and the points in the perturbed data. It then applies an exponential kernel function to these distances, transforming them into similarity scores. The same issue as described in the previous step arises, where the functions are tailored for a conventional ML context.

  In our adaptation, we implemented a function that computes the Euclidean distance between each pair of features, considering the sequential nature of values in time series. The similarity score for the perturbed data is then computed by taking the square root of the average of these distances.

- **Black-box evaluation and Lasso regression:**

  The conventional LIME evaluates the predictions on the new synthetic data and the original instance, examining how small changes in the data influence model predictions. Next, it conducts a Lasso regression analysis to reduce the number of variables, extracting a subset of variables that still contribute significantly to accuracy using the permuted data.

  In IMT-LIME, we select a sample of ten to twenty of the best-perturbed data based on their similarity scores. We then use the black-box model to get predictions for this selected sample, evaluating the impact of small changes in the data input. We skip the step of feature reduction, due to the complexity it creates in the context of multivariate time series.

- **Surrogate model:**

  Subsequently, LIME fits a simple model linear model (typically a weighted linear regression model) to the permuted data using the new set of variables, incorporating the similarity scores as weights.

  However, the linear nature of this model posses a barrier applied to multivariate time series. To address this issue, in IMT-LIME, we fit a Vector Auto Regression model (VAR), specially designed to capture the linearity in multivariate settings as explained in Equation 4.1.

- **Feature importance:**

  Finally, LIME uses the feature weights derived from the simple model to provide explanations for the local behavior of the complex model.

  In, IMT-LIME as displayed in Figure 4.15, the coefficient effect matrix from the VAR model is used to explain the contribution of each feature.

Comparing the KernelSHAP algorithm (the version of SHAP designed to work with any type of models) and the classic SHAP (the version based exclusively on Equation 2.2) as outlined in [12], with Algorithm 5 designed for IMT-SHAP, we can observe the following specific differences:

- **Weight coalitions and weighted linear model:**

  When calculating the Shapley value for feature i as outlined in Equation 2.2, KernelSHAP assigns weights to each coalition of the remaining features based on their distance from the instance being explained. This is typically achieved using the Shapley kernel [12]. Subsequently, KernelSHAP employs a weighted linear regression to fit a model that approximates the original model. The coefficients of this linear model represent the Shapley values for each feature. The authors in [12] refer to this approach as Linear LIME combined with the classic Shapley value estimation.

  IMT-SHAP, on the other hand, does not incorporate a linear model due to the unsuitability of classic linear ML models for multivariate time series data. Instead, IMT-SHAP relies solely on classic Shapley value estimation, as defined in Equation 2.2. However, to address the computational complexity arising from the exponential number of feature coalitions, IMT-SHAP employs a sampling approximation. When computing the Shapley value for feature i as described in Equation 2.2, IMT-SHAP generates a fixed number of random subsets of feature coalitions. This randomness ensures a diverse and representative sampling of the feature space.

- **Classic Shapley value issue:**

  The issue with classic SHAP lies in the dimensionality of the data being explained. Like LIME, traditional SHAP was implemented for standard regression and classification tasks. Consider this part of Equation 2.2:

$$val(S \cup \{i\}) - val(S)$$

In a traditional machine learning context, the prediction values obtained from $val(S \cup \{i\}$ and $val(S)$ could be a probability predicting a specific target class or another value, which does not create inconsistency in the subtraction operation between these two terms. However, in multivariate settings, especially in multi-step forecasting as described in 2.2, the prediction results are a sequence of values, sometimes with high dimensionality such as 3D. This creates inconsistency in the subtraction operation in this formula and generally explains why the current classic SHAP does not work well in multivariate settings.

To address this issue, IMT-SHAP computes the loss between the original instance prediction and the subset of features including feature i being explained, $loss(val(original\_input), val(S \cup \{i\})$, and then calculates the loss between the original instance prediction and the subset of features excluding feature i, $loss(val(original\_input), val(S)$. Lastly, the loss values are used in the subtraction operation. In other words, Equation 2.2 is redefined for IMT-SHAP as follows:

$$\phi_i = \sum_{S \subseteq P \setminus \{i\}} \frac{|S|!(P - |S| - 1)!}{P!} \left(loss(val(o), val(S \cup \{i\})) - loss(val(o), val(S))\right) \tag{4.2}$$

where:

- ϕ_i is the Shapley value for feature i.
- P represents the total number of features excluding feature i.
- S is a subset of features excluding feature i.
- $val(o)$ is the model's prediction value for the original instance.
- $val(S)$ is the model's prediction value when considering the features in subset S.
- $val(S \cup \{i\})$ is the model's prediction value when including the feature i in the subset S.

5

Evaluation

In this chapter, the evaluation of the framework is presented, focusing primarily on assessing the developed interpretability methods: IMT, IMT-LIME, and IMT-SHAP. The chapter begins by introducing the experimental setup used for conducting the evaluations. it then explores various benchmarks employed to assess the performance of the framework. Lastly, the chapter presents the results of these evaluations. The results section first evaluates the performance of different prediction models. Subsequently, it assesses the interpretability methods mentioned earlier in relation to the results of these prediction models.

5.1 Experimental Setup

To facilitate the reproducibility of the results, an environment was set up on an Amazon Web Services EC2 machine [39]. These machines are specifically designed for cloud computing and are thus great for training machine learning models. When creating an EC2 instance, you get to pick from a list of pre-specified configurations, an instance with a GPU was chosen to utilize CUDA when training our machine-learning models. The name of that instance is `g4dn.xlarge`. Table 5.1 shows the specifications of the launched instance:

Component	Value
OS	Ubuntu
vCPUs	4
Memory (GiB)	16
Physical Processor	Intel Xeon
Clock Speed (GHz)	2.5
CPU Architecture	x86_64
GPUs	1
GPU Architecture	NVIDIA t4
Video Memory (GiB)	16

Table 5.1: Specifications of the AWS `g4dn.xlarge` instance

This machine was used to execute and save data from simulations, the training of the machine learning models, and lastly the evaluation of the different models and

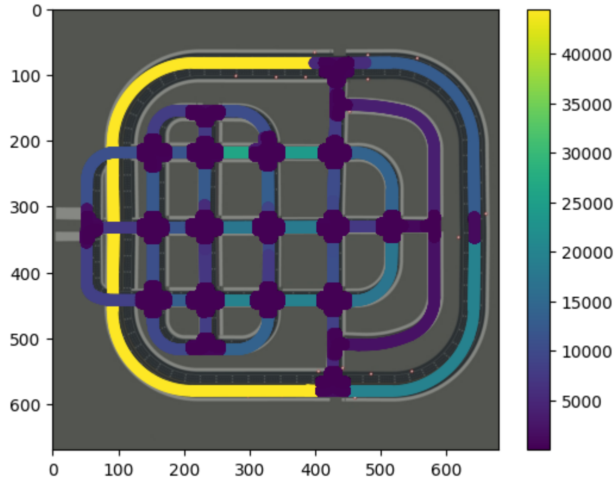

Figure 5.1: Road segment occurrences in the generated environment data

the interpretability results. The machine learning models used for the benchmarks are the ones listed in Section 4.1.2, for all the benchmarks the models get significant training with a similar model complexity. This is ensured by checking the number of trainable parameters for the models. The interpretability methods used are the ones described in Section 4.2 but also the internal interpretability of the TFT as described in Section 4.1.2.1.

5.2 Benchmarks

Here, the various benchmarks used to assess the performance of the framework are presented, along with their evaluation criteria and the environments in which these benchmarks are established.

5.2.1 Benchmark 1 — Town05/Work

For the first benchmark, the framework is being evaluated on a routing problem from Carla's map Town 05. The special thing with this environment is that depending on the values of the features, the model will suggest two different routes. The features the model is supposed to capture are spatial, such as information about the cars around the ego, or even information about the ego vehicle.

For this benchmark, all the data used to train the model is from Scenic and Carla. Thus, the initial map training data is simulated, as well as the behavioral data. In Figure 5.1 you find a plot of all the map data where the closer to the top of the color bar the segment is, the more occurrences of that segment were present in the data.

The goal of the benchmark is to accurately predict the trajectory of the car when going to work, either the normal route or the alternative route when there is congestion in the form of traffic. But also being able to catch the reason behind the route prediction. Such as a varying acceleration and a lot of cars around the ego mean congestion and the model should suggest a different route. In figure5.2 are the two

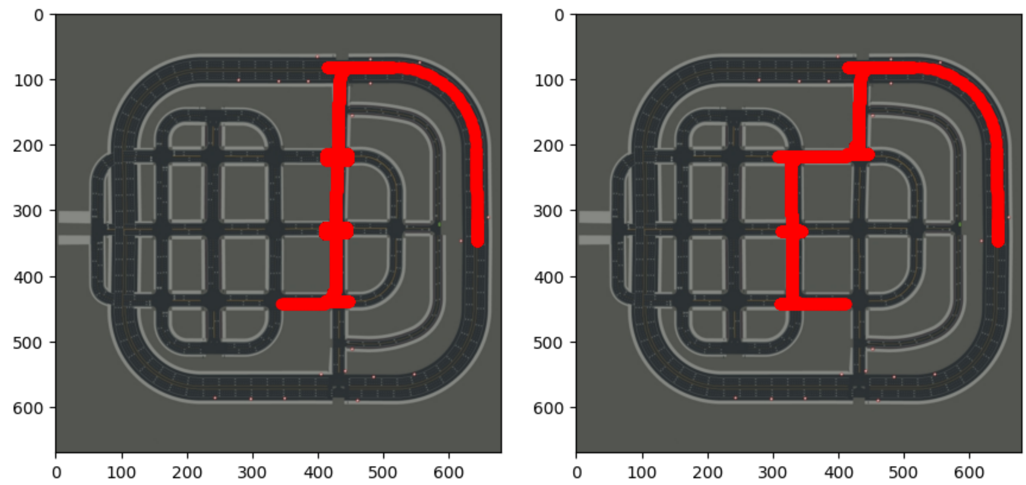

Figure 5.2: Example of the regular and alternative route to work.

different routes displayed, the left image represents the regular drive to work and the right image represents the alternative route in case of congestion.

The dataset used for the benchmark is the behavioral data for work trips, and the evaluation is done against unseen work trips.

5.2.2 Benchmark 2 — Town05/Restaurant

The second benchmark is somewhat similar to the first one. It is also a routing problem on Carla's map Town5. Here the model should capture temporal dependencies since the goal is to suggest a route to a restaurant because the drive is later in the day. The map training is the same as in benchmark 1 and is displayed in Figure 5.1. The behavioral data is gathered from Scenic and Carla in the same way as Benchmark 5.2.1. Figure 5.3 is a graphical representation of the route that should be captured. The goal here is the same as for Benchmark 1, to accurately predict the trajectory whilst maintaining interpretability. The feature that should be captured is time, being later in the day should result in a prediction that leads us to the restaurant.

The dataset used for the benchmark is the behavioral data for restaurant trips, and the evaluation is done against unseen restaurant trips.

5.2.3 Benchmark 3 — Adaptability

The third benchmark is about the different models' adaptability. The goal of the benchmark is to see how much data and how much training is needed for a model to capture a new behavior. Consider the first benchmark. If one were to use the adaptability introduced in Section 4.1.3 how much adaptability is needed for the model to capture the new behavior? Also, how interpretable is the result from the point of starting the adaptability until the new behavior is captured?

We are going to do this by starting at benchmark 1 and then trying to introduce

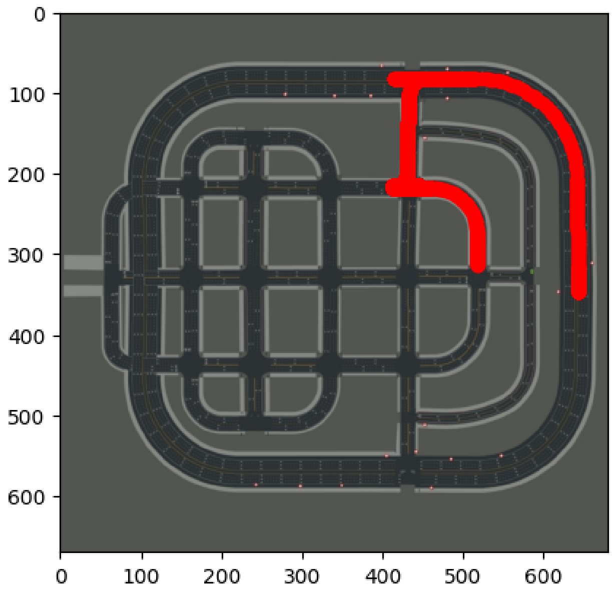

Figure 5.3: An example of a drive to the restaurant

the behavior from benchmark 2. The goal is to compare the different models and interpretability methods and see how the adaptability affects them both and how well they perform over time of adaption. Thus, the dataset and evaluation set from benchmarks 1 and 2 are going to be used.

5.2.4 Benchmark 4 — Model Quality

For the last benchmark, the quality of models is to be examined. To compare the models and the interpretability done on them, the performance of the models and how that performance is correlated with the interpretability need to be evaluated.

For Benchmark 4, benchmarks 1 and 2 will be used again, but this time the focus is on limiting the training of the models. The models should be from the scale of performing very badly to performing as well as expected. To do this, the hyperparameters are tuned and the data the model gets to train is limited.

The goal is to find relations between the quality of the model and the quality of interpretability. Is there a correlation and if so how strong is it? The interpretability will also be examined, comparing it to the intrinsic interpretability of the TFT, to determine how well the post-hoc approach performs.

The datasets from benchmarks 1 and 2 will be joined and used together for this benchmark.

5.3 Results

This section presents the results of the framework evaluation against the benchmarks outlined in Section 5.2. The results are structured into two parts: First, it evaluates the performance of the framework and its models in trajectory prediction tasks. Second, it assesses the interpretability of the framework using the predictions of these models, focusing on the interpretability methods developed: IMT, IMT-LIME, and IMT-SHAP.

5.3.1 Trajectory Prediction

Here, the performance of the prediction models is evaluated against established benchmarks.

5.3.1.1 Influence of the Model on Benchmark Performance

The metrics of the models are displayed in Table 5.2. They are obtained by using the framework as intended and training against benchmarks 1 and 2 and then evaluating the models on a testing dataset suited for both benchmarks. The performance of the models presented in the table can be seen in Figure 5.4, the order, row for row, is TFT, MTF, and LSTM.

Model	Accuracy	F1 score	Precision	Recall
TFT	61.31%	0.1694	0.1720	0.1868
MTF	59.39%	0.2417	0.2755	0.2616
LSTM	38.07%	0.1308	0.1464	0.1435

Table 5.2: Model performance with respect to the metrics: accuracy, F1 score, precision, and recall

From Table 5.2 and Figure 5.4, it is evident that TFT is the best performer of the underlying models. The regular multivariate transformer is a close second but cannot include known future inputs, covariates, and static invariants. Furthermore, observing the predictions made one can see that the quality of the model has a great impact, but a high accuracy does not necessarily correlate to accurate predictions. It is also notable that LSTM performs significantly worse than both TFT and MTF, something that was expected because it is the least complex and promising model from the roster. Lastly, because TFT and MTF show the most promising results and are more complex and rigid models, the trajectory prediction results from now on will exclude LSTM.

5.3.1.2 Impact of Restrained Training on Benchmark Performance

Given the two models, TFT and MTF, the framework was evaluated towards benchmark 4 by limiting the data the models are being trained on using a percentage of the dataset. The points of interest are 25%, 50%, and 75% data, where 100% can be seen in Table 5.2. The results of the limited training can be seen in Table 5.3 and the performance in Figure 5.5.

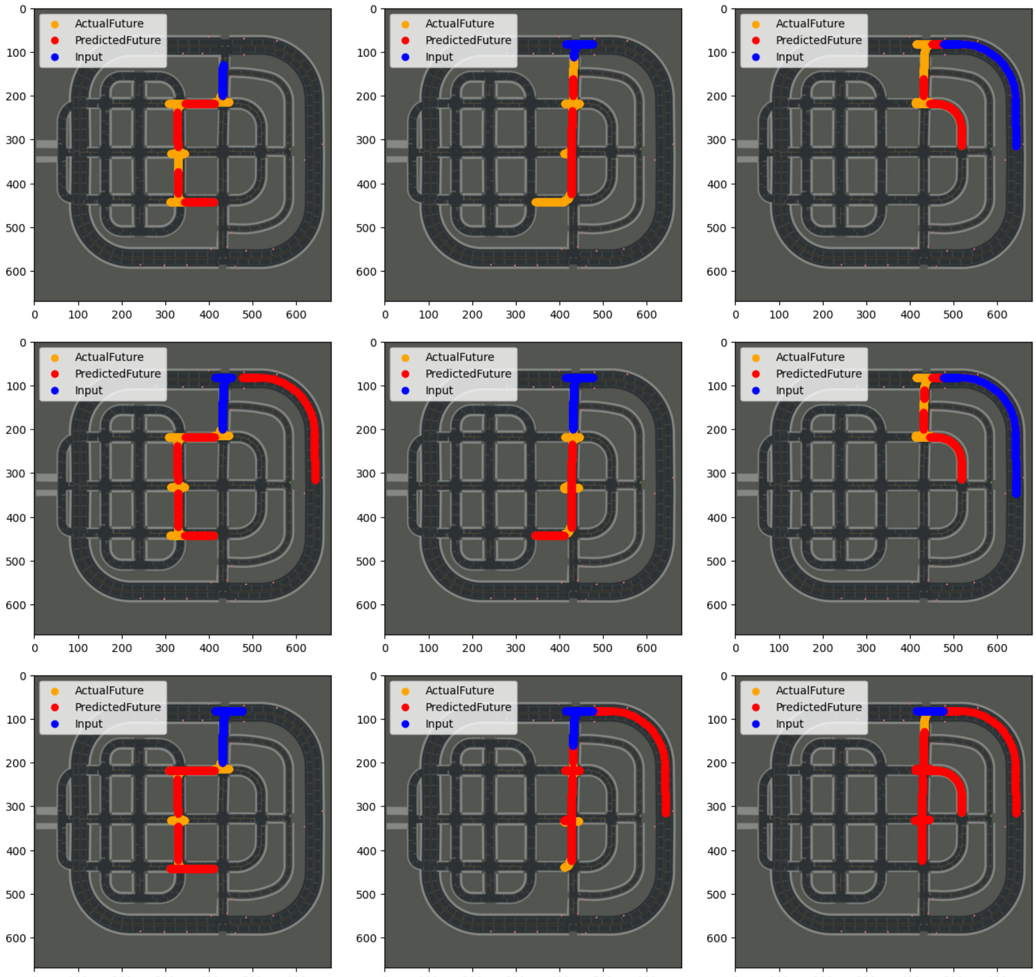

Figure 5.4: Results from evaluating TFT, MTF, and LSTM on benchmark 1 and 2

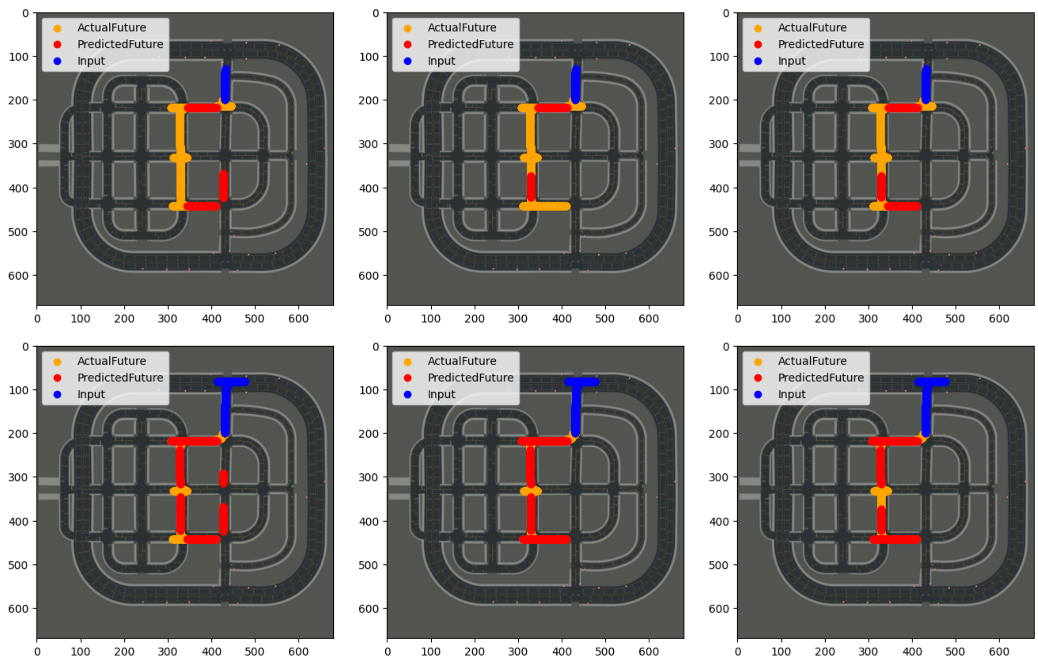

Figure 5.5: Results from evaluating a limited TFT (upper) and MTF (lower) on benchmark 1 and 2, from left to right: 25, 50, and 75 percent

Model	Training	Accuracy	F1 score	Precision	Recall
TFT	25%	46.43%	0.0943	0.0856	0.1180
	50%	56.55%	0.1370	0.1444	0.1624
	75%	57.98%	0.1491	0.1536	0.1626
MTF	25%	40.60%	0.1136	0.1112	0.1313
	50%	56.55%	0.3053	0.3425	0.3235
	75%	58.33%	0.2858	0.3160	0.3008

Table 5.3: Evaluation metrics on limited models according to Benchmark 4

From Table 5.3, it is evident that both models become better with more data, the increments go down the closer to 100 percent they get, signaling that we have reached the peak of accuracy, and after that point, there will be some diminishing returns. Most likely, more complex data could still be beneficial and result in an even higher accuracy, that is, if one could extend the training and reach the peak later in the process.

Looking at the images in Figure 5.5 it is evident that MTF is quicker at learning the routing and has good results already at 50 percent. However, TFT is still not performing as expected when reaching 75 percent of the data and still needs the last 25 to completely learn the rerouting. This indicates that MTF could be the best performer, specifically being able to adapt and learn new data faster. However, that could also indicate that MTF easily unlearns old behaviors, which is bad given the context.

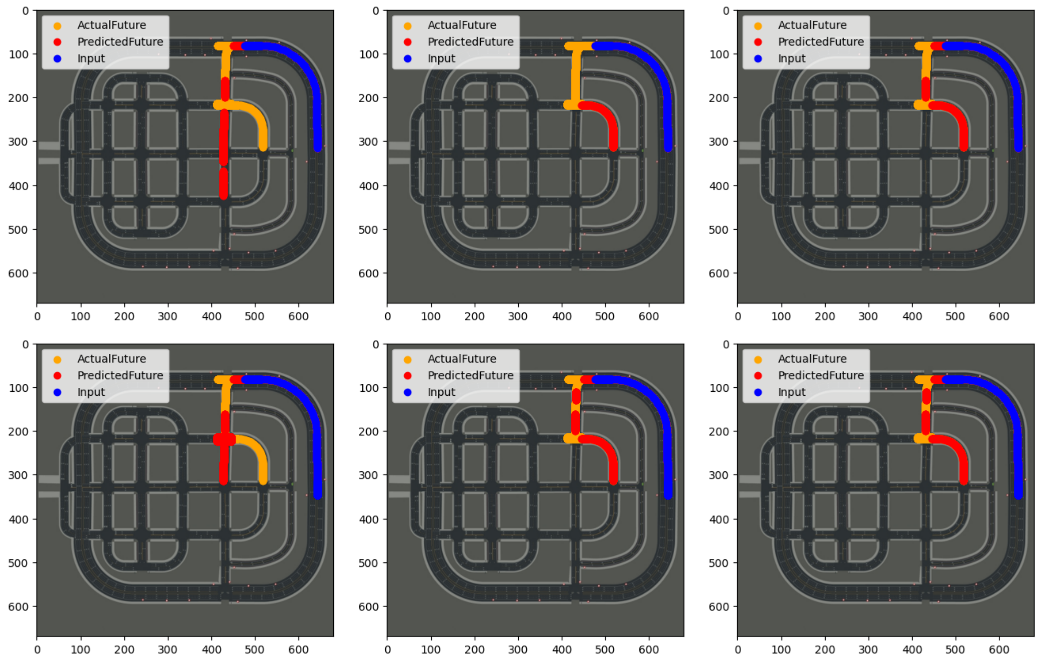

Figure 5.6: Results from evaluating TFT (upper) and MTF (lower) on benchmark 3

5.3.1.3 Effect of Adaptability on Benchmark Performance

Given the framework and the underlying models TFT and MTF, the framework will be evaluated against benchmark 3. This is done as described in 5.2. The results are presented in Figure 5.6, from left to right: 0%, 50%, and 100% of the original restaurant training data, consisting of 50 drives.

We can see that both models can capture the new behavior. Again, MTF seems to do this faster than TFT, as can be seen in the preceding section about restrained training, another result that is in favor of the multivariate transformer. However, when training with 100% of the restaurant data, TFT reached an accuracy of 80% when being evaluated on unseen restaurant data, whilst MTF only reached 72.9%. This does not in itself say a lot because the interpretability of the models is still unknown. However, the adaptability process does, without fail, capture the new behavior for both of the models.

5.3.2 Interpretability

Here, the interpretability methods are evaluated against the benchmarks.

5.3.2.1 Influence of the Model on Interpretability

In this section, Figures 5.7, 5.8 and 5.9 present the results from evaluating the interpretability methods for the respective trajectory prediction in Section 5.3.1.1. The order is row for row, IMT, IMT-SHAP, and IMT-LIME. The order, column for column, is the same as in Section 5.3.1.1.

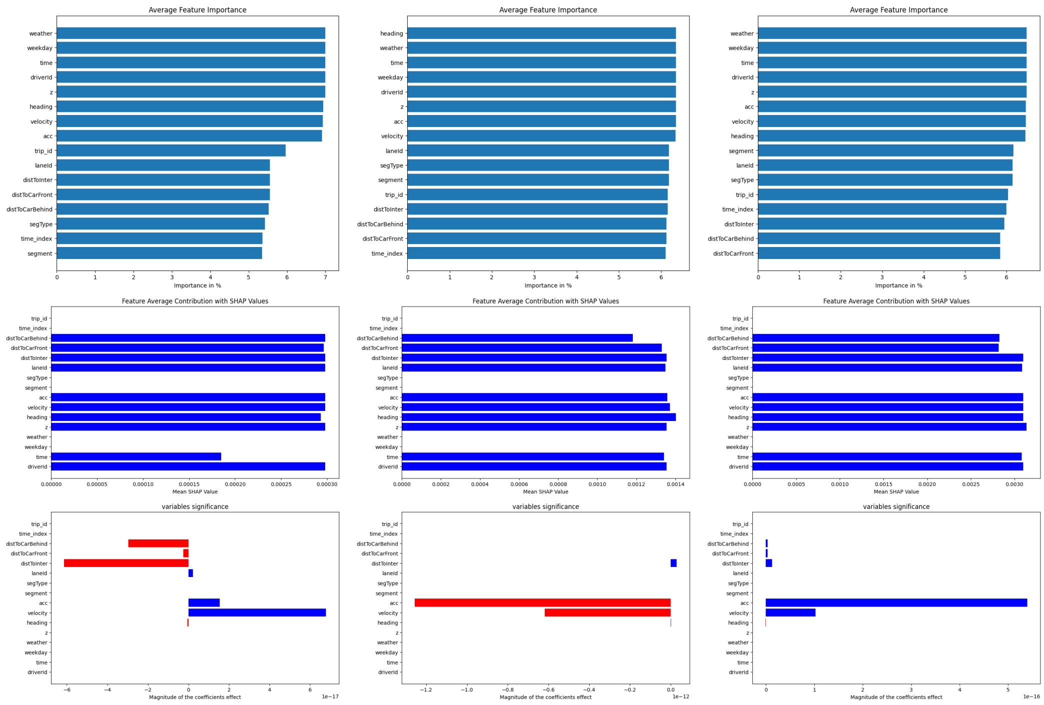

Figure 5.7: Results from evaluating TFT on benchmark 1 and 2

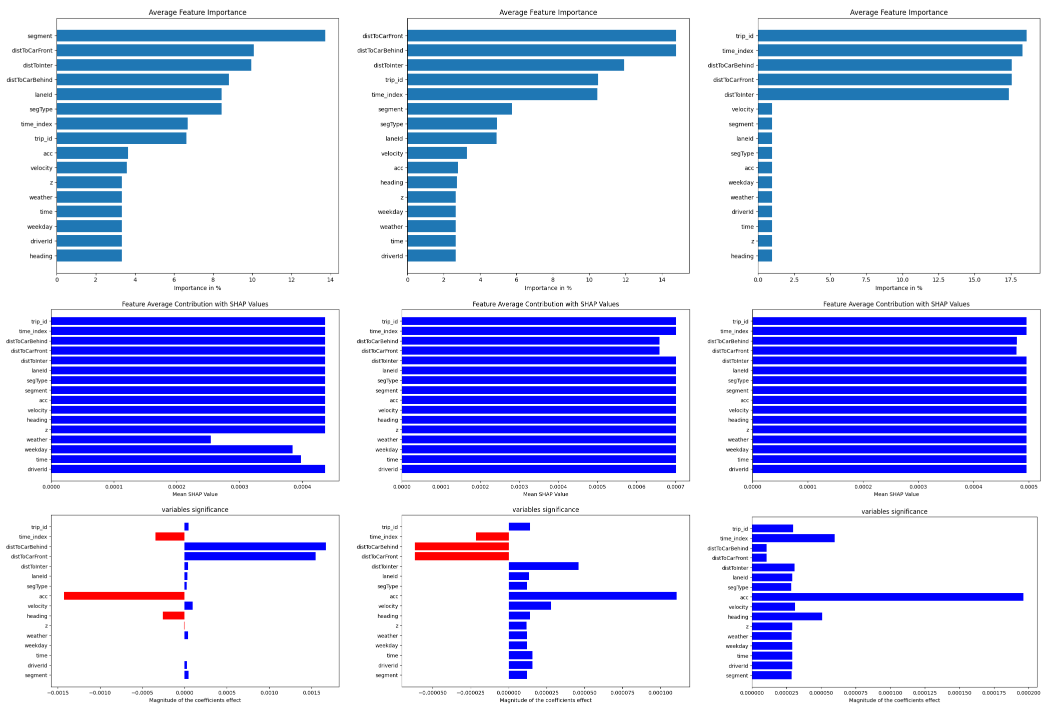

Figure 5.8: Results from evaluating MTF on benchmark 1 and 2

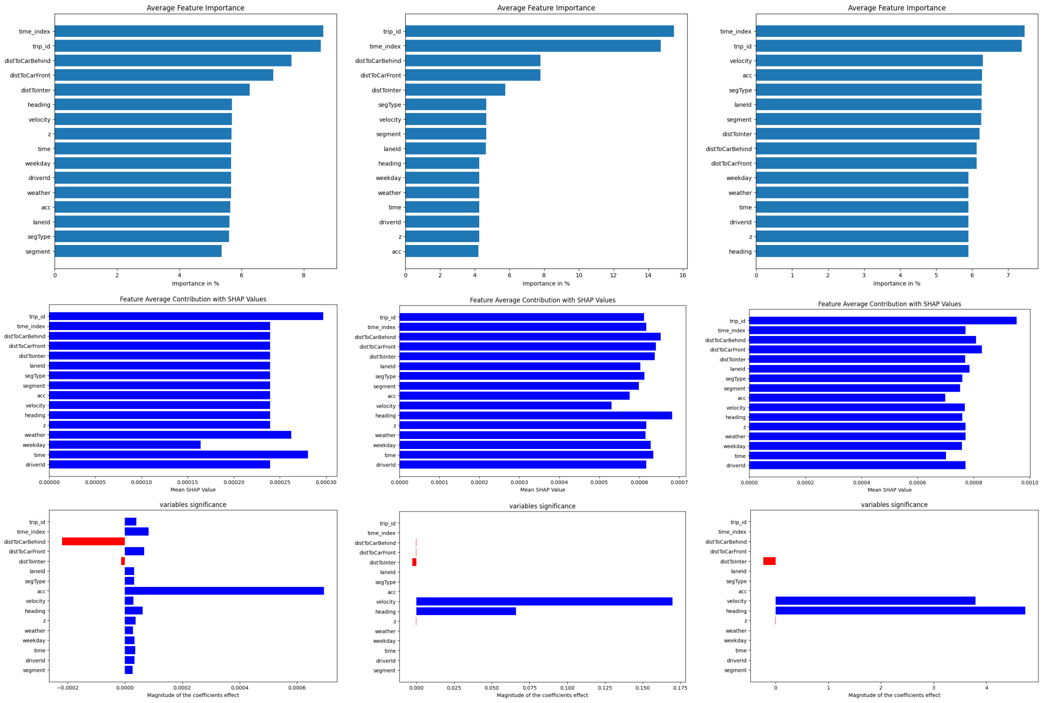

Figure 5.9: Results from evaluating LSTM on benchmark 1 and 2

IMT for TFT seems to grade a majority of the features as important. This is a strange but possibly correct result, however, it does show that the model finds something important with every feature which, in itself, is good. IMT for MTF differs a lot from TFT, it is a lot better at differentiating features. That makes it easier to determine what the model finds important but could also result in the model having a hard time capturing behaviors related to, what it deems less important features. For benchmark 1 and 2 however, it favors the expected features. Looking at IMT for LSTM, it can be observed that the benchmark 1 performance is quite good and similar to both TFT and MTF. However, looking at benchmark 2, both the accuracy in the prediction and the feature importance are wrong. This is a good example of how poor model performance can affect the results of interpretability.

Again, looking at IMT for LSTM, it seems to grade both *trip_id* and *time_index* as two important features. This unfavorable result is an indication of the model learning the specific trip rather than the behavior on that trip. The same happens when IMT-SHAP is applied to the prediction. However, IMT-LIME also performs badly but not for the same reason. It seems to favor the continuous features such as acceleration and velocity, even though their impact has to be less than what IMT-LIME suggests. LSTM is the worst performer both for trajectory prediction but also interpretability. Consequently, the remaining interpretability results will be focused on TFT and MTF.

5.3.2.2 Impact of Restrained Training on Model Interpretability

Here, figures 5.10 and 5.11 present the results of evaluating the interpretability methods for respective trajectory prediction in Section 5.3.1.2. The order is row for

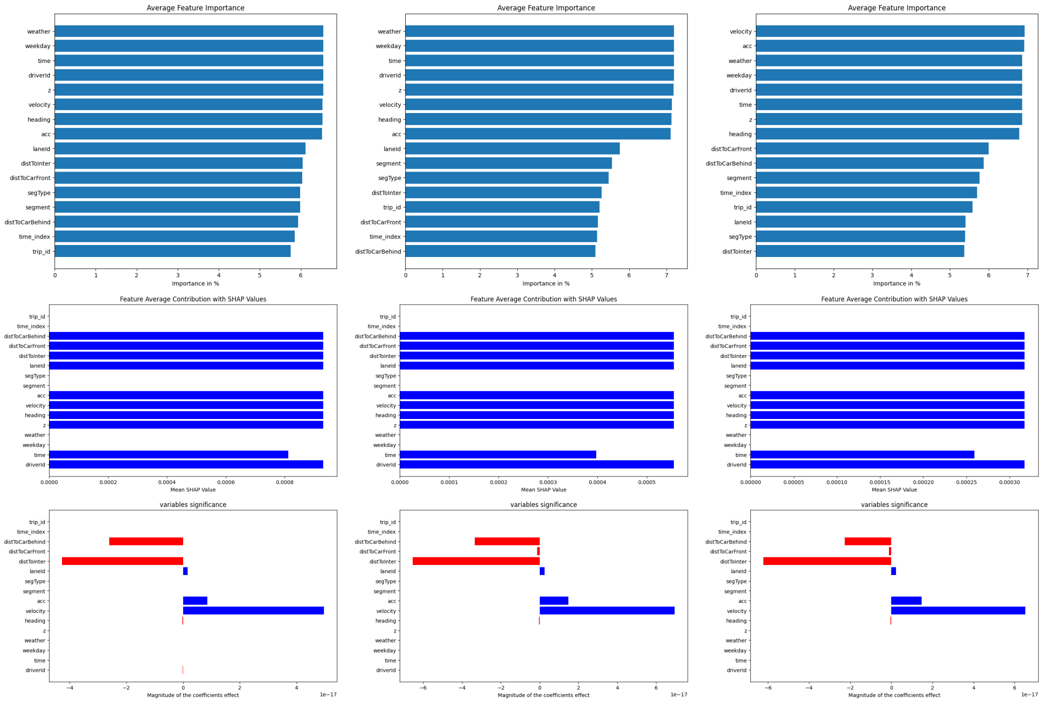

Figure 5.10: Results from evaluating a limited TFT on benchmark 1 and 2, from left to right: 25, 50, and 75 percent

row, IMT, IMT-SHAP, and IMT-LIME.

The interpretability plot for the TFT differs significantly between the three different percentages. In the two leftmost plots, IMT finds weather and weekdays to be important, two features that should not be interpreted as important for this benchmark. However, the last interpretability plot, captures the important features effectively, as acceleration will be a sign of congestion. Looking at IMT-SHAP it performs poorly across all three limitations, no features protrude and the top performers are all similar in value. The only feature it deems less important is time, and given the description of the benchmark, time should be interpreted as important. IMT-LIME barely changes between the time steps and again favors acceleration and velocity. Two potential reasons for this found importance are that IMT-LIME focuses more on explaining the model, and the modifications it does to the features are so large that they can apply to any predictions and score similarly. However, both acceleration and velocity are favored by IMT and could be the accurate result.

Looking at IMT for the multivariate transformer, it finds the important features and captures these features earlier in the training process than TFT. Looking at IMT-SHAP and IMT-LIME for MTF, IMT-SHAP still has a hard time differentiating between features. LIME works a lot better with MTF than with TFT but does not agree with IMT on the order of the feature importance. It does however agree with how LIME scored the feature importance of TFT, suggesting that maybe the predictions are accurate, or rather as accurate as they can get with IMT-LIME.

Comparing the two, it is evident that the multivariate transformer seems to be

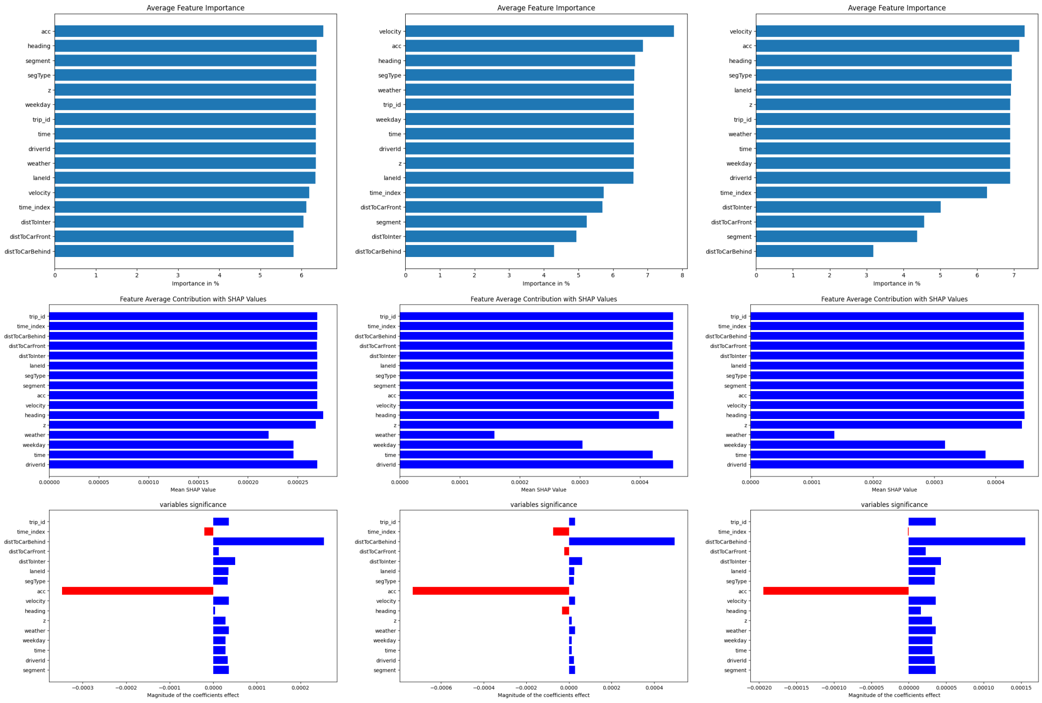

Figure 5.11: Results from evaluating a limited MTF on benchmark 1 and 2, from left to right: 25, 50, and 75 percent

more interpretable for both IMT-SHAP and IMT-LIME. However, when looking at IMT, it seems to score the features somewhat alike. Suggesting again that IMT, rather than taking the importance of the model, takes the feature importance of the predictions. Furthermore, the limitation of data seems to have a great impact on the retrieval of accurate interpretations.

5.3.2.3 Effect of Adaptability on Model Interpretability

Here, figures 5.6 and 5.13 present the results from evaluating the interpretability methods for respective trajectory prediction in Section 5.3.1.3. The order is row for row, IMT, IMT-SHAP, and IMT-LIME. The order, column for column, is the same as in Section 5.3.1.3.

For the TFT, it is evident that the interpretability successfully captures the new behavior, this can be seen by comparing the predictions and interpretations of the results from evaluating towards benchmarks 1 and 2. When training with 100% of the restaurant data, an accuracy of 80% is reached by evaluating against unseen restaurant data. The interpretability seems to, again, find all the features to have an equal importance.

Similar to TFT, the multivariate transformer again manages to capture the new behavior, however, this can be seen already at 50%. Looking at the interpretation furthest to the left, with the worst prediction, shows that features like `segType` and `Z` are important. Which is bad because they should be nowhere near the top. Moving on in the adaption, 50 and 100 % are both notable performers and seem to find the

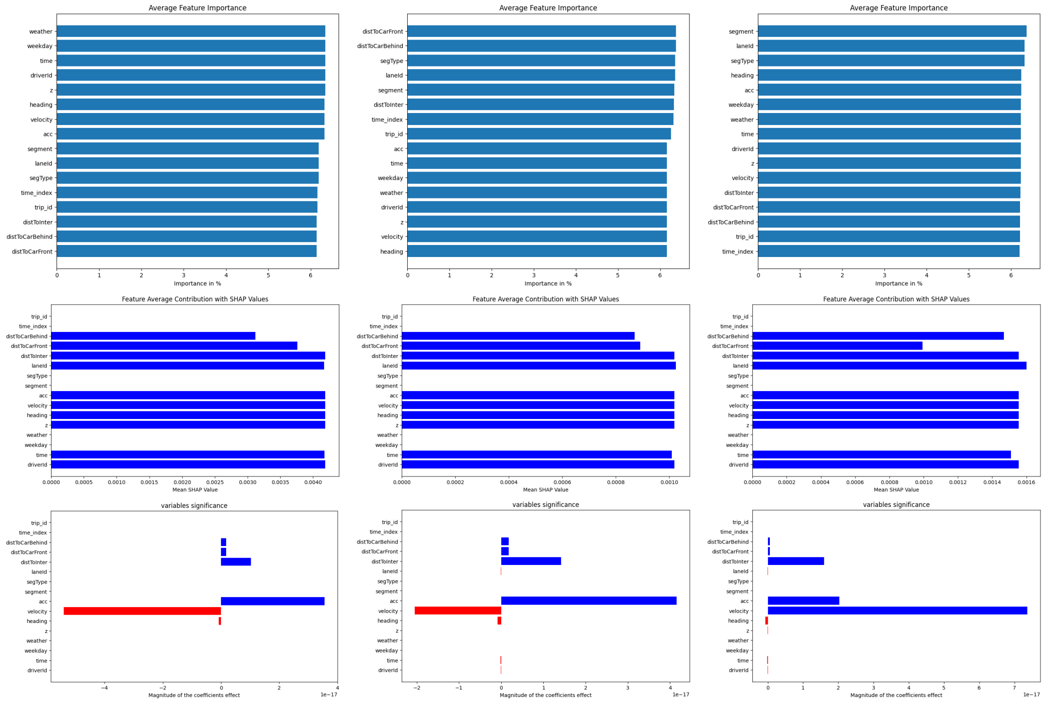

Figure 5.12: Results from evaluating TFT on benchmark 3

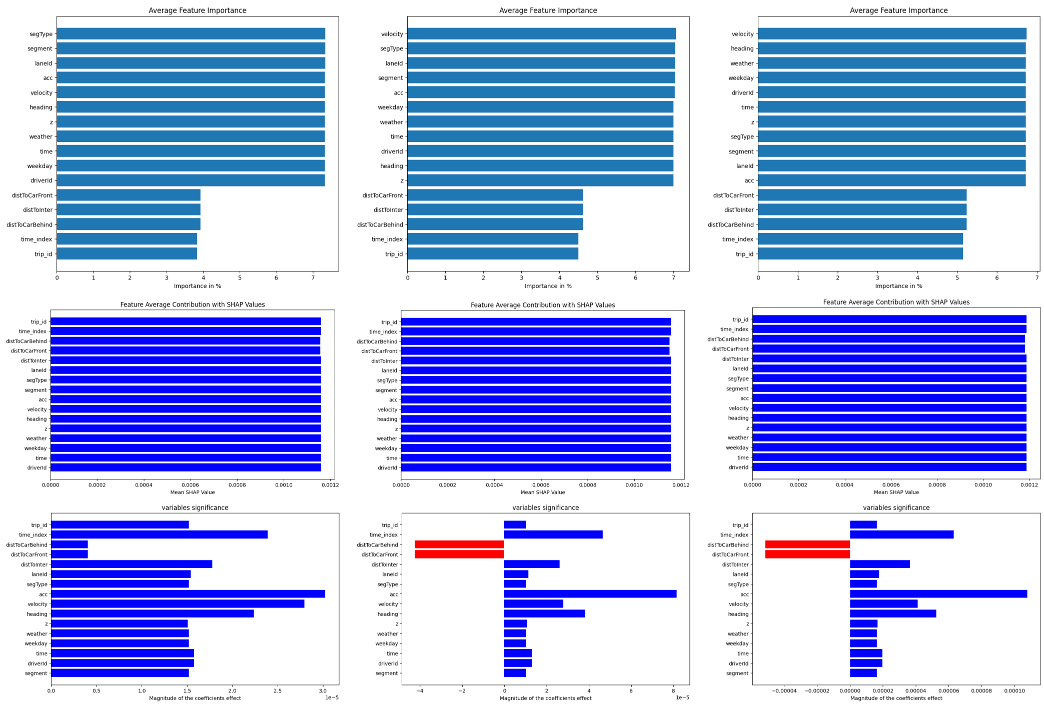

Figure 5.13: Results from evaluating MTF on benchmark 3

relevant features. The order is not the same, but the top features are. In the plot, however, it looks like the values are equal for the top 8 features, but looking closer at the values, see Table 5.4, it becomes clearer that there is a difference. In favor of the 100% adaption, it is a more interpretable order and this shows that even though the prediction is pretty good for 50%, the last 50 makes a great contribution to the interpretability.

Velocity	Heading	Weather	Weekday	driverId	Time	...
72.46	72.38	72.38	72.37	71.98	71.98	...

Table 5.4: Expanded values for the interpretability plot furthest to the right in Figure 5.13

5.3.2.4 Post-hoc Interpretability versus Intrinsic Interpretability

To see how well the interpretability from the framework performs, it can be evaluated against the intrinsic interpretability of TFT. The intrinsic approach is not necessarily the ground truth for the feature importance, but gives us insight into what the model itself deemed important, here the goal is to see how close our post-hoc approach can get. Benchmarks 1 and 2 will be revisited to compare IMT, IMT-SHAP, and IMT-LIME to the TFT's intrinsic interpretability. To determine how the interpretability varies across models, the methods will also be compared for the MTF (excluding the intrinsic approach), and evaluated on the same benchmarks.

Looking at the uppermost plots of Figure 5.14, it can be observed that the intrinsic interpretability heavily favors the segment. This is likely due to the model being trained on the environment data first, and the environment data training is there to capture the relations between segments. Another reason for the intrinsic to have this favoritism but not the other methods is that by using attention weights, we are simply recovering the information from how the model weighted the different input features. This is not the same approach as our post-hoc method, for example, IMT, where the feature importance values are based on how easily the model can reconstruct the same prediction once we have masked or permuted a feature. It can also be observed that the intrinsic approach is similar across multiple different predictions, this is an indication that the attention weights do not change. However, looking at IMT/IMT-SHAP/IMT-LIME, it is evident that they differ quite a lot between predictions. The interpretations made by them are more prediction-specific than model-specific, which is in line with how the post-hoc approach works and makes it more suited in instances where the goal is to know the importance of features for a single prediction rather than simply interpreting a model.

Similarly to the other benchmarks, one can observe how IMT does not single out a couple of features that are of greater importance. This issue is likely in the models rather than the interpretability method, this can be seen by looking at how IMT works in Figure 5.15. However, there is a correlation between how many permutations and masking are done for IMT and how accurate the interpretations become, thus, being close in percentage after performing a couple of hundred permutations does not necessarily mean that they are close in value.

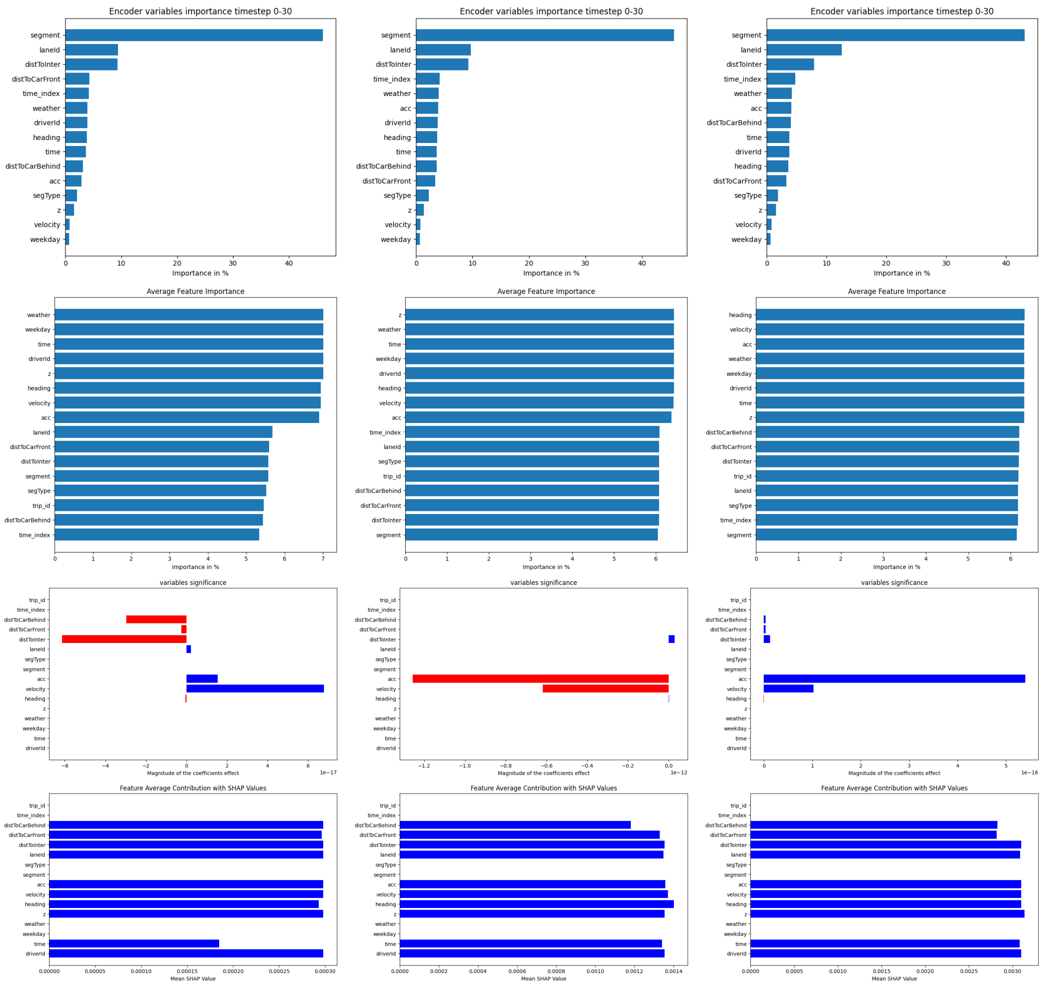

Figure 5.14: Results from evaluating the interpretability of TFT

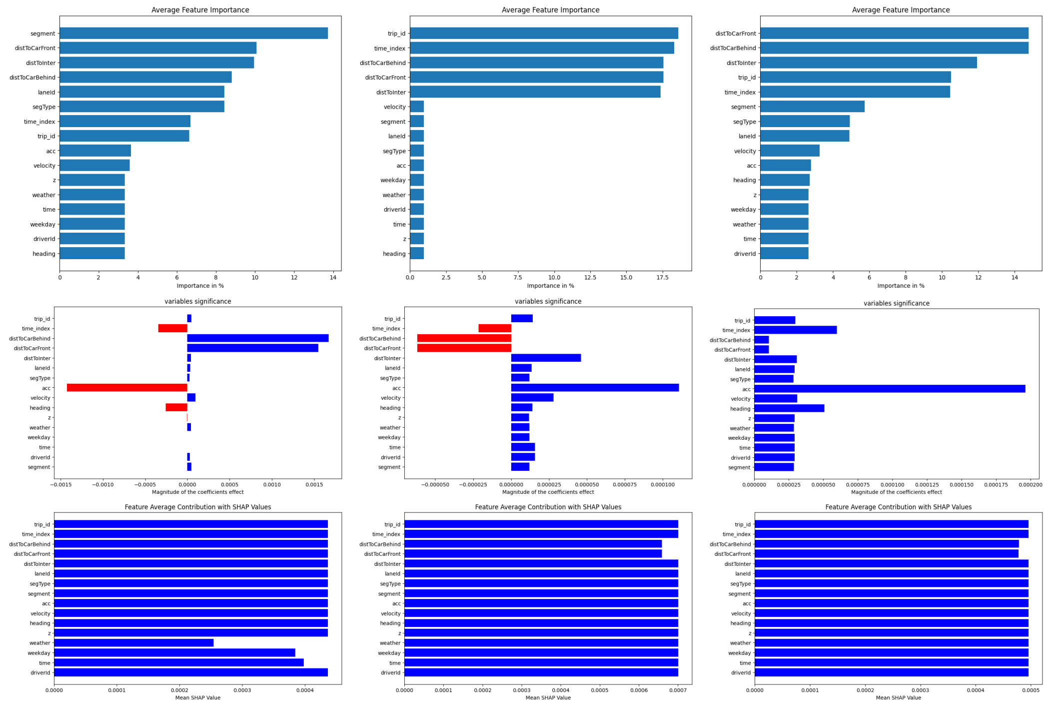

Figure 5.15: Results from evaluating the interpretability of MTF

IMT-LIME can distinguish features that do not contribute a lot from the high contributors, which is pretty close to the intrinsic. However, as observed here and described in Section 4.2.2.2, IMT-LIME can not compute the importance of classification features when using TFT, and thus can not directly be compared to the other results. IMT-SHAP is also suffering from this issue. Similarly to IMT, IMT-SHAP does a lot of iterations and the close importance scores are a result of that. The main issue with IMT-SHAP is the complexity, to get feature importance for a single prediction the amount of permutations had to be lowered by a large factor. Reducing the amount of time from a couple of days to around 2 hours, which is still a lot of time compared to both IMT and IMT-LIME finishes in minutes.

In Figure 5.15, the available interpretability methods are evaluated on the MTF. Firstly, it can be observed that there is a change in IMT, where it now has no issues distinguishing important features from less important ones. Showing that TFT, the previous model, was the culprit to the lack of distinguishable importance. IMT's feature importance now is also quite similar to the intrinsic, even though they are two completely different models but evaluated on the same trips, another argument for IMT being more in favor of interpreting predictions rather than models.

Looking at the importance of IMT-LIME, it is evident that it favors acceleration, which for some predictions is correct. However, for it to be always important is strange, and is likely related to how IMT-LIME calculates the importance. Trying to score the acceleration and pick a value that is quite far from the ground truth, it will receive a lower weight than, say, a value close to the ground truth. But, if the relation between the weight and the value is not proportional to the effect of the modification, then the acceleration would be scored higher than it should. This can also be observed happening to the velocity.

Additionally, IMT-LIME, by design, evaluates the model's sensitivity to perturbations in the data surrounding the original prediction. Given the centrality of acceleration and velocity in trajectory prediction tasks, the model is likely to be more responsive to changes in these features. This sensitivity means that any alterations in acceleration and velocity are reflected more prominently in the model's predictions, highlighting their importance. For instance, in an urban traffic scenario, if the aim is to explain a trajectory prediction based on input data representing sudden breaking due to traffic lights or congestion, perturbing the velocity may simulate a scenario of sudden acceleration. IMT-LIME's sensitivity analysis in such situations will likely reveal significant prediction changes for both velocity and acceleration, highlighting why it is important to accurately model these features for reliable trajectory prediction. IMT does not have this issue because it never creates any new values, only masks and permutes, and will not experience any inflation in importance.

Looking at IMT-SHAP for MTF, the performance is unfortunately still quite bad. The running time is a bit better because MTF runs predictions faster than TFT, but the permutation space still has to be lowered to finish in a couple of hours. By limiting the permutations both when evaluating IMT-SHAP for TFT and MTF, the computation time is reduced, but this constraint may introduce approximation errors, as it excludes the contributions of some feature combinations. This aspect could partially explain why the interpretability provided by IMT-SHAP seems to yield less satisfactory results.

6

Related Work

The main focus of our thesis research was the interpretability of machine learning models within the context of route improvement and behavioral trajectory prediction in the automotive domain. In this context, the use of a post-hoc method, as discussed in [8], proved to be the most suitable approach for explaining the model's output. Since our research primarily aimed at designing a framework for analyzing various types of route improvement and behavioral trajectory prediction models, employing a post-hoc method provided flexibility and adaptability. One of the primary challenges we encountered was the lack of research addressing the handling of high-dimensional data, such as car trajectory data. In this section, some works that aim to resolve similar tasks are explored, and the approaches used by the authors are examined.

6.1 Interpretability

The author's work in [40] aims to provide interpretability for trajectory predictions generated by Convolutional Neural Network models. They have implemented a framework where, given the driving history, road map, and social context of a driver, the system generates three distributions, each representing a possible trajectory. The framework uses epistemic uncertainty estimation techniques to predict the three most likely distributions and then performs a more advanced analysis of the estimated uncertainty. [40]. However, this approach tends to be more intrinsic than post-hoc, as the techniques are initially developed for specific types of models (CNNs).

The adaptation of LIME for time series classification, as detailed in [41], offers valuable insights into addressing certain challenges encountered when implementing our adapted version of LIME. These challenges include determining a meaningful interpretable representation of time series data, realistically perturbing a time series, and defining a local neighborhood around a time series. While the methods outlined in this paper, such as segmenting time series based on their shape and statistical properties to detect change points, have demonstrated effectiveness, they are primarily designed for univariate time series analysis. These change points serve as temporal indices that can be used to define time windows for analysis. Each window can then be examined individually using the steps outlined in 3.3.2.2. However, when dealing with multivariate time series data, the high dimensionality presents a challenge in achieving realistic segmentation across different feature classes.

In [42], a comparable technique involves dividing a sequence into time windows to reduce the computational complexity of calculating Shapley values for time series data. Unfortunately, this method is designed for classification tasks where the predicted label corresponds to only one point or time step, which differs from our objective of handling multi-horizon forecasting.

While the approaches outlined in [41] and [42] could provide a starting point for developing our methods (IMT, IMT-SHAP, and IMT-LIME), the concept of segmenting the input sequence being explained, into windows and then applying explanation methods may not be universally applicable to all types of deep learning models. For example, in both works, the experiments were conducted using a traditional Recurrent Neural Network (RNN), which is well-suited for sequence-to-sequence tasks. However, for other types of RNNs such as LSTMs, and for more complex architectures like Transformers, which capture long-term dependencies across time steps of the input sequence, segmenting this sequence could disrupt these dependencies, potentially resulting in less accurate or interpretable outcomes.

In the study outlined in [43], the focus is on adapting state-of-the-art interpretability techniques, originally developed for images, to multivariate time series data. The approach involves representing the input data as a 2D image, where each pixel contains abstract visual features. This transformation converts the time series into a two-dimensional format, with each row representing a feature and each column representing a timestamp. Additionally, to accommodate negative peak values observed in the samples, pixel values are rescaled to the interval [-1, 1]. Subsequently, three interpretability methods from different families such as Grad-CAM, Integrated Gradient (IG), and LIME are selected to provide insights into the input data. However, the approach outlined in [43] differs slightly from the methods developed during our project. While our methods are exclusively based on tabular data, the main idea explored in this paper could serve as a starting point for future works or comparison results.

6.2 Trajectory Prediction

In the area of trajectory prediction without any interpretability, there is a paper about destination prediction based on partial trajectory data [18] that makes use of a Long Short-Term Memory (LSTM)-based model. In contrast to many of the popular trajectory prediction papers, the author suggests that a route is represented by a set of regions instead of coordinates. This yields predictions that output a probability of a region being contained in the route, and thus no personal driving history is needed to train the model. While there are similarities between the approach outlined in our project and the one discussed in this paper [18], such as predicting the routes of a vehicle based on its recent trajectory, there are also notable differences. In their model, destination scores are generated, and subsequently, routes to the most probable destinations are calculated, with the most likely region being designated as the final destination. Another distinction is that they exclusively utilize LSTM models for their predictions.

The method explored in [44] also employs an LSTM model for predicting the next location. What sets this approach apart is how the authors represent vehicle trajectories, focusing solely on longitude and latitude coordinates at specific positions. They specifically handle spatial features without considering temporal features such as time of day and travel time along the journey. In our context, if we were to use longitude and latitude coordinates for predicting trajectories, it might be beneficial to follow the same approach explained in this paper, consisting of partitioning the coordinates into smaller regions or cells. This is necessary because when predicting a specific trajectory using latitude and longitude, there could be an infinite number of possible data points that can describe the trajectory, as longitude and latitude are continuous in space. The authors address this by defining a finite set of locations that can express all trajectories, essentially representing trajectories as sequences of small cells.

The approach outlined in [45] utilizes Large Language Models (LLMs) to bridge the existing gaps in trajectory prediction, particularly the lack of high-level scene understanding and interaction information that skilled drivers possess. The framework described in this paper enhances trajectory prediction through a structured four-step process. The first step, *Sparse Context Joint Coding*, transforms agent and scene features into a format that LLMs can process. The subsequent phase, *High-level Interaction Modeling*, is employed to model the complex interactions and contextual dynamics within the environment, such as social dynamics and movements of other vehicles. The third step, *Lane-aware Probabilistic Learning*, focuses specifically on lane information, emulating human-like navigational insights to enhance prediction accuracy and adapt to lane-specific contexts. Lastly, the *Multi-modal Laplace Decoder* is employed to generate diverse and probabilistically predictions, ensuring that the model's outputs adhere to the physical and operational constraints of real-world driving scenarios. Although LLMs have not been explored in this thesis, incorporating them could significantly improve the accuracy of models in this context as future work.

6.3 Adaptability

To address the limitations of Long Short-Term Memory (LSTM) models in efficiently capturing seasonality and trends present in time series data, the authors in [46] employed a hybrid approach incorporating three key components: a machine learning model, data pre-processing methods, and an optimization algorithm. The approach outlined in this paper uses Seasonal-Trend decomposition using Loess (STL) techniques for preprocessing the data, combined with a Backtracking Search Algorithm (BSA) to optimize the parameters of LSTM networks. The time series data is decomposed into three components: seasonal, trend, and remainder using the STL technique. This decomposition isolates distinct patterns within the time series, making it easier to analyze and model each component separately. The BSA algorithm is used to improve learning ability. Separate LSTM networks are trained on the seasonal, trend, and remainder components of the decomposed time series, with the BSA optimizing the internal weights and thresholds of these LSTMs, leading to enhanced prediction accuracy. The fundamental difference with our method of adaptability lies in the fact that we do not use specific models; our approach is generalized and fits both LSTM models and transformers. Additionally, our approach does not aim to find the best configuration for the model, but instead globally aims to retrain the model with the current configuration using a new dataset. The intention is to teach the model new behavior, thereby increasing its capability to predict accurately in this specific scenario.

The work presented in [47] aims to enhance the accuracy of streamflow prediction by integrating Extreme Learning Machine (ELM) with Particle Swarm Optimization (PSO) and Grey Wolf Optimization (GWO) to create a hybrid model that facilitates adaptability in the learning process. As in the preceding paper [46], the authors use optimization algorithms to select the model that significantly reduces the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) and improves the Nash-Sutcliffe Efficiency (NSE). While the technique outlined in this paper could serve as a starting point for developing the adaptability process for our framework, it is worth mentioning that using optimization algorithms to build adaptability for different types of models can be an exhausting task. Finding the best algorithm that works well across all types of models is challenging. For instance, PSO and GWO work satisfactorily with ELM models, a type of neural network known for its fast learning speed and strong nonlinear ability. However, due to their structure, ELM models are lightweight and use a single-layer feedforward network where the hidden layer parameters (weights and biases) are randomly assigned and remain fixed. While PSO and GWO optimization algorithms have shown good performance in this scenario, they might face challenges with the high dimensionality and complexity of transformer models, for example. This is why our approach of retraining the model on unseen behavior to optimize its capability of predicting accurate results in a specific environment initially seems to be a good approach to enhancing adaptability.

7
Discussion

This chapter is a discussion of the findings outlined in the earlier sections of this thesis. The goal is to provide comprehensive reflections on the results, while also considering ethical aspects.

7.1 Design and Implementation Choices

The focus is mainly going to be on the four major designs and implementations: data generation, prediction models, adaptability, and interpretability.

The data generation process was heavily driven by the lack of data and to combat the issues of not being able to evaluate any of the work done, we looked at alternatives. Finally settling for using a probabilistic programming language, Scenic. Being able to write scenarios and automating the process of generating data that resembled the one of a real driving journal was crucial for the thesis. Something that would help the evaluation would be doing a simulation-to-reality comparison, the combination of scenic data generation and data collected by a car. This is, unfortunately, not covered in the thesis, as we opted to focus on the simulation application, completing that, and if there were time, put the effort towards bringing the framework to real driving scenarios.

Considering the prediction models, we did extensive research trying to identify a couple that suited our needs. We wanted models that worked in a multivariate setting, something that directly limited our search space. We chose to pick the most promising model that we could find, the Temporal Fusion Transformer and picked two models that traditionally have been used in similar settings, the transformer and LSTM, to compare the results from the most promising model. The results from the comparisons confirmed what we thought, that the more complex TFT performed best. However, TFT was designed to work with almost all kinds of forecasting problems, making it not tailored to our specific task. Ideally, there would exist a model specifically designed for this task or developed by us, but we opted not to because developing the framework was the purpose of this thesis.

Adaptability is the tool we used to incorporate adaptiveness in the framework. The results were conclusive that it works and successfully lets a model learn new behaviors. The design behind the adaptiveness is based on the concept of transfer learning and neighborhood analysis, with some extra steps. It works great with our framework and domain. Closely related to adaptability is the safety specification. Where we opted to represent the specification with invariants and temporal logic. The latter was picked because Scenic allowed for scenarios to have temporal requirements, which means that Scenic can not construct a scene if the requirement, the temporal logic formula, is not valid in the dynamic scenario.

Regarding interpretability, a lot of design and implementation choices were made. The process of finding methods for interpreting in a multivariate setting was exhaustive. We chose to adapt existing methods that were proven to work in the univariate setting and aggregating them to multivariate, but also to create our interpretability method. For us, this was a key decision in our thesis because it let us tinker and explore the area of interpretability in a completely new way, not just changing parameters, observing attention weights, and watching results from already constructed methods.

7.2 Research Goals

The main goal of this thesis was to create a framework that given a set of inputs can construct an interpretable prediction model. The interpretations should yield insight into the model's predictions, leading to transparency for the end users. The prediction model should accurately capture the behavior of a driver and be able to suggest routes based on real-time data. Based on our results, we have successfully created a framework that inherits these features. However, as mentioned in Section 7.1, not being able to evaluate the framework on real data is a loss for the result, but we still believe that the transition to the real setting will be quite painless because the generated data is designed with authenticity in mind.

The prediction models, which are an essential part of the framework, performed well. We did reach an accuracy of just above 60 percent during training, but, we believe that this is far from the best accuracy we could get, the issue here is not being data but instead our target forecasting. Because we opted for regular forecasting, where we predict a segment for each future time step, the model, even though it would predict all segments correctly and in the right order, would punish some outliers. For example, consider the case where the entire prediction is shifted 1 time step forward, the prediction is almost correct but would be punished by a lot, thus lowering the accuracy. Looking at our predictions, this is what happened most of the time, and it led to the model skipping over segments because predicting them usually led to a higher loss. Thus, we get predictions where not all segments are directly connected, this can be easily solved by interpolating between the segments when suggesting the route for the user. However, in hindsight, some approaches could yield a better performance, such as, instead of segmenting the roads we could segment an entire map. The accuracy would potentially be higher, albeit with less detailed predictions.

Interpretability, the other very essential part of our framework, also performed well. Unfortunately, we can not score the interpretations in any reasonable way because there is no ground truth to compare to. We believe that the work we have done and the evaluation of the interpretability methods are enough to show that the interpretations work, and based on our goals, yields favorable answers. We also believe that we have shown that for our use case, IMT is the best performer specifically for model agnostic and post-hoc interpretability. It does not need any adaptions to work with new models, it is truly a black-box explainer. It also does not inherit the issues IMT-LIME and IMT-SHAP have, such as complexity, overestimating the feature importance of certain continuous features, and not working with class features.

7.3 Ethical Considerations

Considering the rerouting system being implemented in a car, as a whole, the ethical implications of the decision-making process gain critical importance. The technology does promise efficiency and lets the user sit back and relax and trust that the car knows what it is doing but with the added interpretability. However, what if the interpretability is wrong? The dangerous part of interpretability is that it promises the end user that what you are seeing is the truth about why the prediction model predicted as it did. All the actions performed by the user after a faulty interpretation are questionable.

Another ethical concern we have is the implementation of the framework. Because we have never evaluated the framework with real driving data, some adaptions have to be made once we start taking that step. The results might not be the same once we have transferred the environment, and thus should not be used as proof for the framework working in a real setting. However, it is not necessarily something bad because if we know the framework works in a simulation setting, and someone performs some kind of simulation to reality validation or verification then we can be more certain that the results are still accurate in reality, or even better how we can transfer what we have created and explored in simulations.

7.4 Threats to Validity

As mentioned multiple times in the previous sections, opting to only use a simulation environment for the results is a potential threat to the validity. We did manage to make it work in a simulation setting and that was the goal of the thesis, but not being able to transfer that knowledge to a real setting would be a great loss. It will, however, not change the fact that it did work in the simulation, and nothing we have found indicates that it will not work in a car.

Another potential threat is data, for us, data was only trips we have generated ourselves. We know and can guarantee that that data is correct and has no issues. But, if we were to use data that is not perfect the results might look different or our results may not be reproducible.

Furthermore, the implementation of all three interpretability approaches developed within this thesis relies on the choice of loss functions. The selection of a suitable loss function becomes particularly evident when measuring the change in prediction after the perturbation and masking process. If the chosen loss function is not well-suited to the specific context or the characteristics of the data, it can potentially result in misleading conclusions regarding feature importance. Therefore, considerable attention was paid to carefully selecting the appropriate loss function that aligns with the objectives of the task at hand. Additionally, future research could explore the impact of different loss functions on the outcomes of interpretability methods to mitigate this threat and enhance the validity of the findings.

Lastly, in multivariate settings, features often interact in complex ways. IMT, IMT-SHAP, and IMT-LIME may capture these interactions differently, potentially leading to inconsistent or inaccurate interpretations of feature importance. From an end-user's perspective, receiving different interpretations for a single instance can undermine the model's reliability. To enhance consistency across these models, the next step could involve developing an aggregation method that synthesizes the outputs from all three approaches. For example, this could mean conducting a detailed analysis of the specific input data being explained and identifying the feature importance values from IMT, IMT-SHAP, and IMT-LIME where they converge. By presenting the end-user with a result that integrates these consistent values, we can provide an explanation that is both reliable and robust. This approach would ensure more aligned interpretations, thereby enhancing user trust and confidence in the model's predictions.

However, from a research standpoint, implementing these three different approaches has allowed us to gain insights into their similarities and differences, helping us understand where and when to use a specific method and for what purpose. Through developing these three methods, we discovered that, based on their structures, they can be applied to three distinct types of tasks: Instance-specific Analysis, Perturbation Analysis, and Counterfactual Explanations Analysis.

As mentioned in previous sections, IMT is fast and does not inherit the time complexity issues of IMT-SHAP or the proportionality issues of IMT-LIME concerning ground truth. IMT is well-suited for Instance-specific Analysis by examining the contribution of features to the prediction. However, when we need to slightly alter the input features to understand their specific influences on the prediction (Perturbation Analysis), IMT-SHAP proves to be the appropriate approach. Similarly, when we need to observe how various small changes in the input can impact the prediction, IMT-LIME is ideal. Therefore, from a research perspective, developing these three approaches for the thesis was beneficial. Despite that, to effectively achieve our goal of implementing interpretability methods that enhance user trust, further work on result aggregation would be important to avoid inconsistencies.

8 Conclusion

The purpose of this thesis was to evaluate and study behavioral trajectory prediction models, and how one can interpret them. The goal was to create a framework that combines these two components and facilitates good learning and transparency for end users. In this chapter, we conclude the findings of the report.

To solve issues with the lack of data, we generated a driving journal using a probabilistic programming language that has an API for a car simulator. Developed and documented the process of such a generation and established techniques for various sub-tasks to the problem.

We introduced adaptiveness to the framework by creating an adaptability loop. Here, we utilized one of the findings from the data generation process, that is, how we can train models using data from simulations. We generated new data by performing neighborhood analysis on failed predictions or failed drives, we demonstrated how training on such data yielded favorable results and a form of adaptiveness. We successfully incorporated a safety specification to guide the learning process and maintain a high quality of data while adapting the model to new behaviors.

Interpretability of the old established methods is not well suited for a multivariate setting, aggregating them means a great loss of information and an increase in time complexity, making them unusable for something so dynamic as a driving car. The intrinsic approach to interpretations, with the use of attention weights, is not perfect either and leads to interpretations that more explain a model than a prediction. To combat the issues of the above interpretability methods, we have developed IMT, our method of interpretation. It solves issues with the established techniques and focuses on truly being a black box explainer, well suited for interpreting multivariate input and output.

Prediction models in the multivariate forecasting area are sparse but functional. We have found and created models that can perform the task with high accuracy, remaining interpretable and adaptable.

In conclusion, we developed and constructed a framework that incorporates prediction models, interpretability methods, data generation, and adaptability. The framework is complete and covers the information on how we can combine the building blocks in a way that facilitates interpretable and adaptive route improvement based on behavioral trajectory prediction.

8.1 Future Work

To fully understand the logic behind the decisions made by adaptive models for route improvement based on behavioral trajectory predictions, further research is needed. This includes extending the methods for interpretability developed during this project and testing other types of machine-learning models that were not covered.

The interpretations produced by IMT, IMT-SHAP, or IMT-LIME essentially focus on understanding the significant features that played a crucial role in the predictions. An alternative method for exploring the model's behavior, as outlined in [8], could involve examining prediction uncertainty. Learning and quantifying uncertainty could provide additional insights for interpretability by allowing us to assess the model's confidence in its predictions. Understanding the level of uncertainty associated with each prediction enables end-users or researchers to evaluate the reliability of the model's output. This could serve as a global interpretability approach, complementing the local interpretability methods developed during this project.

Another approach for exploring the model's behavior could involve using the techniques outlined in [43], which were based on transforming the multivariate time series into 2D images. Utilizing these techniques would enable us to evaluate the interpretability methods we developed. While LIME [11] and SHAP [12] are not inherently suited for multivariate time series data, they excel in interpreting images. By transforming the trajectory data, originally in tabular format, into an image format, we could leverage libraries like LIME to interpret the resulting images. This would allow us to compare the interpretations obtained from our methods of interpretability.

Additionally, the use of established methods for analyzing multivariate time series data to gain insights into various aspects, such as identifying patterns and trends, could be explored. This means the use of descriptive statistical measures such as mean, median, variance, and covariance to understand the central tendency, variability, and relationships between variables over time. This becomes particularly valuable in scenarios where the structure of the training data is not known in advance. For instance, when attempting to predict the destination of a new user, conducting multivariate analysis on the user's driving data provides a comprehensive overview of trends and behaviors within the dataset. This global perspective facilitates the interpretation of results obtained from machine learning models, enabling a deeper understanding of the underlying patterns and dynamics driving the predictions.

Lastly, the framework developed in this thesis could be evaluated using methods and techniques not covered extensively in this project. This could involve exploring the potential of using, for example, Large Language Models (LLMs) for learning, as outlined in [45]. Additionally, extending the evaluation of the framework could involve using different multivariate time series datasets from sectors such as healthcare, or finance, where the data dynamics differ from what we analyzed in the context of vehicle trajectory prediction. This would allow us to assess the adaptability and robustness of the framework across various forecasting tasks. We can also explore the feasibility of performing evaluations on how experiments conducted in virtual environments can be generalized to the real world. Carla allows users to use maps from the real world for simulations. Using this functionality, we can create simulated data on a real-world map and compare it with data collected from the same map in reality. This approach would involve gathering data from simulations based on real-world environments using Carla and then collecting real-world data from these areas. By performing a parallel study, we can assess the resemblance and differences between the simulated and real-world data, and determine how well the findings from the virtual environment approximate real-world scenarios.

Bibliography

[1] A. Theissler, J. Perez-Velazquez, M. Kettelgerdes, and G. Elger, "Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry," *Reliability Engineering [?] System Safety*, vol. 215, p. 107 864, Nov. 2021. DOI: 10.1016/j.ress.2021.107864.

[2] M. Sahal, Z. Hidayat, R. Putra, M. Rizqifadiilah, and F. Saputra, "Lane keeping system using convolutional neural network for autonomous car," Oct. 2023, pp. 205–210. DOI: 10.1109/ICTS58770.2023.10330834.

[3] L. Yang, Y. Yang, G. Wu, *et al.*, "A systematic review of autonomous emergency braking system: Impact factor, technology, and performance evaluation," *Journal of Advanced Transportation*, vol. 2022, pp. 1–13, Apr. 2022. DOI: 10.1155/2022/1188089.

[4] R. Moraffah, M. Karami, R. Guo, A. Raglin, and H. Liu, "Causal interpretability for machine learning - problems, methods, and evaluation," *CoRR*, vol. abs/2003.03934, 2020. arXiv: 2003.03934. [Online]. Available: https://arxiv.org/abs/2003.03934.

[5] WirelessCar, *About wirelesscar*, Accessed 12-April-2024. [Online]. Available: https://https://www.wirelesscar.com/who-we-are/.

[6] WirelessCar, *Smart ev routing*, Accessed 12-April-2024. [Online]. Available: https://www.wirelesscar.com/products/smart-ev-routing/.

[7] F. Doshi-Velez and B. Kim, *Towards a rigorous science of interpretable machine learning*, 2017. arXiv: 1702.08608 [stat.ML].

[8] M. Du, N. Liu, and X. Hu, "Techniques for interpretable machine learning," *Communications of the ACM*, vol. 63, no. 1, pp. 68–77, 2019.

[9] R. Guidotti, A. Monreale, F. Turini, D. Pedreschi, and F. Giannotti, "A survey of methods for explaining black box models," *CoRR*, vol. abs/1802.01933, 2018. arXiv: 1802.01933. [Online]. Available: http://arxiv.org/abs/1802.01933.

[10] C. Molnar, *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable.* https://christophm.github.io/interpretable-ml-book/, 2020, Accessed: 2023-09-24.

[11] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[12] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *arXiv preprint arXiv:1705.07874*, 2017. [Online]. Available: https://arxiv.org/abs/1705.07874.

[13] M. Jin, Q. Wen, Y. Liang, *et al.*, *Large models for time series and spatio-temporal data: A survey and outlook*, 2023. arXiv: `2310.10196 [cs.LG]`.
[14] J. Korstanje, *Advanced forecasting with Python*. Springer, 2021.
[15] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976.
[16] R. H. Shumway, D. S. Stoffer, and D. S. Stoffer, *Time series analysis and its applications*. Springer, 2000, vol. 3.
[17] T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom, and E. M. Wolff, "Covernet: Multimodal behavior prediction using trajectory sets," *CoRR*, vol. abs/1911.10298, 2019. arXiv: `1911.10298`. [Online]. Available: `http://arxiv.org/abs/1911.10298`.
[18] P. Ebel, I. E. Göl, C. Lingenfelder, and A. Vogelsang, "Destination prediction based on partial trajectory data," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2020, pp. 1149–1155.
[19] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, *Temporal fusion transformers for interpretable multi-horizon time series forecasting*, 2020. arXiv: `1912.09363 [stat.ML]`.
[20] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: `10.1162/neco.1997.9.8.1735`. eprint: `https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf`. [Online]. Available: `https://doi.org/10.1162/neco.1997.9.8.1735`.
[21] A. Ip, L. Irio, and R. Oliveira, "Vehicle trajectory prediction based on lstm recurrent neural networks," in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, IEEE, 2021, pp. 1–5.
[22] L. Lin, W. Li, H. Bi, and L. Qin, "Vehicle trajectory prediction using lstms with spatial–temporal attention mechanisms," *IEEE Intelligent Transportation Systems Magazine*, vol. 14, no. 2, pp. 197–208, 2021.
[23] S. Li, X. Jin, Y. Xuan, *et al.*, "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting," *Advances in neural information processing systems*, vol. 32, 2019.
[24] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[25] G. Hackeling, *Mastering Machine Learning with scikit-learn*. Packt Publishing Ltd, 2017.
[26] P. V. Balachandran, "Adaptive machine learning for efficient materials design," *MRS Bulletin*, vol. 45, pp. 579–586, 2020.
[27] M. Ben-Ari, *Principles of the Spin model checker*. Springer Science & Business Media, 2008.
[28] O. Wiki, *List of osm-based routing services — openstreetmap wiki*, Accessed 6-March-2024, 2024. [Online]. Available: `https://wiki.openstreetmap.org/wiki/List_of_OSM-based_services#Routing`.
[29] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[30] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: A language for scenario specification and scene generation," in *Proceedings of the 40th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*, Jun. 2019.

[31] D. J. Fremont, X. Yue, T. Dreossi, S. Ghosh, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: Language-based scene generation," *CoRR*, vol. abs/1809.09310, 2018. arXiv: `1809.09310`. [Online]. Available: `http://arxiv.org/abs/1809.09310`.

[32] E. Vin, S. Kashiwa, M. Rhea, *et al.*, *3d environment modeling for falsification and beyond with scenic 3.0*, 2023. arXiv: `2307.03325 [cs.PL]`.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: `https://scikit-learn.org/stable/modules/permutation_importance.html`.

[34] ASAM, *Opendrive*, Accessed 23-April-2024, 2024. [Online]. Available: `https://www.asam.net/standards/detail/opendrive`.

[35] O. Wiki, *Main page — openstreetmap wiki*, Accessed 23-April-2024, 2022. [Online]. Available: `https://wiki.openstreetmap.org/w/index.php?title=Main_Page&oldid=2301410`.

[36] P. Forecasting, *Temporal fusion transformer*, Accessed 23-April-2024, 2020. [Online]. Available: `https://pytorch-forecasting.readthedocs.io/en/stable/api/pytorch_forecasting.models.temporal_fusion_transformer.TemporalFusionTransformer.html`.

[37] F. Chollet *et al.*, *Keras*, `https://keras.io`, 2015.

[38] J. M. Haslbeck, L. F. Bringmann, and L. J. Waldorp, "A tutorial on estimating time-varying vector autoregressive models," *Multivariate Behavioral Research*, vol. 56, no. 1, pp. 120–149, 2021.

[39] A. W. Services, *What is amazon ec2*, Accessed 4-May-2024, 2024. [Online]. Available: `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html`.

[40] M. Itkina and M. J. Kochenderfer, *Interpretable self-aware neural networks for robust trajectory prediction*, 2022. arXiv: `2211.08701 [cs.RO]`.

[41] T. Sivill and P. Flach, "Limesegment: Meaningful, realistic time series explanations," in *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, G. Camps-Valls, F. J. R. Ruiz, and I. Valera, Eds., ser. Proceedings of Machine Learning Research, vol. 151, PMLR, Mar. 2022, pp. 3418–3433. [Online]. Available: `https://proceedings.mlr.press/v151/sivill22a.html`.

[42] A. Nayebi, S. Tipirneni, C. K. Reddy, B. Foreman, and V. Subbian, "Windowshap: An efficient framework for explaining time-series classifiers based on shapley values," *Journal of Biomedical Informatics*, vol. 144, p. 104 438, 2023.

[43] L. Tronchin, E. Cordelli, L. R. Celsi, *et al.*, "Translating image xai to multivariate time series," *IEEE Access*, 2024.

[44] S. C, H. Y, and J. K, "Network-wide vehicle trajectory prediction in urban traffic networks using deep learning," *Transportation Research Record Journal of the Transportation Research*, 2018. DOI: `10.1177/0361198118794735`. [On-

line]. Available: https://www.researchgate.net/publication/327524033_Network-Wide_Vehicle_Trajectory_Prediction_in_Urban_Traffic_Networks_using_Deep_Learning.

[45] Z. Lan, H. Li, L. Liu, *et al.*, "Traj-llm: A new exploration for empowering trajectory prediction with pre-trained large language models," *arXiv preprint arXiv:2405.04909*, 2024.

[46] Y. Wu, X. Meng, J. Zhang, *et al.*, "Effective lstms with seasonal-trend decomposition and adaptive learning and niching-based backtracking search algorithm for time series forecasting," *Expert Systems with Applications*, vol. 236, p. 121 202, 2024.

[47] R. M. Adnan, R. R. Mostafa, O. Kisi, Z. M. Yaseen, S. Shahid, and M. Zounemat-Kermani, "Improving streamflow prediction using a new hybrid elm model combined with hybrid particle swarm optimization and grey wolf optimization," *Knowledge-Based Systems*, vol. 230, p. 107 379, 2021.