Project

UNIVERSITY OF SKÖVDE

Collision Prediction in a roundabout based on Trajectory Data

LSTM, CNN & GRU Approach

Project in Data Science

15 credits

Spring term Year

Student: Kim Wurster, Lorenz Koch, Juan Alarcón

Supervisor: Joe Steinhauer

Examiner: Göran Falkman

Abstract

This project explores the effectiveness of deep learning models in predicting collisions between cars and bicycles at roundabouts using traffic surveillance data. The study compares three neural network architectures—LSTM, GRU, and CNN—across two tasks: trajectory prediction and collision detection. Evaluation metrics include Mean Absolute Error (MAE), inference time, accuracy, precision, recall, and F1 score.

Results show that the GRU model performed best in trajectory prediction with the lowest MAE and fastest inference. The LSTM model achieved the highest collision prediction accuracy (92.1%) with balanced precision and recall, while CNNs offered the fastest computation but suffered from lower accuracy and high false positive rates.

Overall, each model has unique strengths: GRU for efficient trajectory forecasting, LSTM for reliable collision detection, and CNN for rapid processing. A hybrid approach combining these strengths may offer an optimal solution for real-time traffic safety systems.

Distribution of Work

Kim Wurster - LSTM

- Chapter 2.1.2 LSTM
- Chapter 2.2.2 LSTM for Collision Prediction
- Chapter 4.2 LSTM
- Chapter 5.1 Results LSTM

Lorenz Koch - GRU

- Chapter 2.1.3 GRU
- Chapter 2.2.3 GRU for Collision Prediction
- Chapter 4.3 GRU
- Chapter 5.2 Results GRU

Juan Alarcón - CNN

- Chapter 2.1.4 CNN
- Chapter 2.2.4 CNN for Collision Prediction
- Chapter 4.4 CNN
- Chapter 5.3 Results CNN

The rest of the chapters were a joint effort of all three team members:

- Chapter 1 Introduction
- Chapter 2.1.1 Vehicle Collision Prediction
- Chapter 2.2.1 Collision Prediction with imbalanced dataset
- Chapter 2.3 Why is it important to have a look at different approaches
- Chapter 3 Method
- Chapter 4.1 Dataset
- Chapter 6 Discussion
- Chapter 7 Conclusion

Table of Contents

List of Tables

List of Figures

1 Introduction

Analysing traffic and making use of traffic surveillance cameras opens up new possibilities for pre-emptive traffic safety interventions and other areas concerning traffic. Predicting the future movement of vehicles is a critical use-case for a variety of different applications like autonomous driving or an early alert system for potential crashes. Cyclists in particular are in big danger when it comes to driving in a roundabout, as they are comparably weak to cars and trucks but also fast while not being as adaptable in their movement as pedestrians. To further reduce the number of accidents happening and increase the general safety of roundabouts, this project focused on the prediction of trajectories for bicycles and cars which then were used for collision prediction. To make use of the available data, this project used different approaches which then were compared in their different aspects. It was expected that each approach excels at a different task of the process. Different types of Recurrent Neural Networks (RNNs), namely Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) were in the focus of this paper, as they offer different capabilities when it comes to handling sequential data, but also an experimental approach with Convolutional Neural Networks (CNNs) was thoroughly analysed to explore their strengths and weaknesses.

1.1 Problem definition

The entire problem of collision prediction in a roundabout can be broken down into two different parts. The first part is the prediction of the movement of the different road users. It is essential to precisely predict the movement as it is crucial for determining whether a collision will happen or not. The trajectory prediction is based on a sequence of past information about the vehicle which then is used to predict the position in the short-term future.

The second part is the actual collision prediction. Based on the predicted trajectories, it should calculate if a collision is happening or not, so in essence it uses similarly shaped input data as the trajectory prediction. This problem of collision prediction specifically is very hard to tackle, as collisions are not that frequent to actually work out a proper collision prediction model. To address the problem of having no collisions, different approaches were used. The approaches contain using positional and temporal thresholds and removing the timestamps.

Successfully solving the two main parts of the problem of collision prediction in a roundabout contributes to developing intelligent systems which enable a safer traffic environment.

While prior research made important discoveries in trajectory prediction using similar approaches in highway environments (Jeong et. al. 2021, Benterki et. al. 2020). These approaches were focussing on continuous, lane-based motion and might fail to generalize in the different traffic layout of a roundabout. Recent approaches to traffic accident risk prediction already used the temporal dynamics of vehicle interactions, however they were applied in a large-scale scenario which also included highways, national roads and urban roads, resulting in over 350.000 vehicles (Li et. al 2025). Our project on the other hand is designed for a comparably small scale, with only 13.500 cars and 450 bicycles. This project is also focussing on the interaction between cars and bicycles, whereas other projects only focus on cars.

This report looks at different algorithms and answers the following research question: How

effective are the different algorithms in predicting collisions from trajectory data based on performance metrics and inference time?

In chapter 2 the different approaches and architectures of the used models will be explained in detail as well as the different approaches on how to tackle the issue with having no collisions present in the actual data. Chapter 3 focuses on the overall method that is used for this project. It will go into detail about why a comparison between the three different model types was chosen. Chapter 4 will talk about the concrete implementations of the different model architectures which were discussed in chapter 2 as well it will address the dataset and general preprocessing steps of the data that were done before working on the different implementations. The next chapter will contain the results of the different approaches, with the focus being on accuracy as well as the inference time. Chapter 6 will then discuss the findings, results and implementation as well as ethical and societal aspects. In the end the conclusion will contain everything summarized to the very essence and also include future work that could be done based on the results of this paper.

2 Background

Collision prediction is a key component in autonomous driving and traffic safety systems, aiming to forecast potential accidents before they happen. Achieving accurate predictions requires models that can understand dynamic behaviour over time and predict trajectories. Deep learning techniques such as LSTM networks, GRUs, and CNNs are used for this purpose. This chapter introduces the foundational concepts and relevant research on how these models are used for trajectory prediction and collision prediction, providing the context for the method presented in this project report.

2.1 Preliminaries

2.1.1 Vehicle Collision Prediction

Hema et al. (2024) investigate the potential of LSTM-based frameworks in improving crash risk prediction, especially in rear-end collision scenarios. Their optimized LSTM architecture, enhanced through CNN-based feature extraction and fine-tuned using the Improved Grasshopper Optimization Algorithm (IGOA), significantly outperformed traditional methods in precision, sensitivity, and false alarm rates. Similarly, Saravanarajan et al. (2023) emphasize the importance of real-time accident detection in AVs and suggest that while many approaches aim at prevention, systems must also focus on live detection of crash scenarios, including those involving single vehicles. Although their study centers on ensemble learning, it underlines the broader relevance of sequence-based models like LSTM, which can process temporal sequences of vehicle trajectories to anticipate and detect collisions before they occur. Candela et al. (2023) propose a risk-aware decision-making framework using Reinforcement Learning (RL) and Gaussian Process-based collision prediction. This work underlines the necessity of accurate, real-time modeling of dynamic environments—an area where LSTM models are highly applicable. By integrating LSTM-based prediction into such decision-making frameworks, Autonomous Vehicles (AVs) can achieve improved safety and responsiveness in complex traffic situations. Collectively, these studies demonstrate the effectiveness and adaptability of LSTM networks in vehicle collision prediction tasks,

establishing them as a foundational component in the development of autonomous driving systems.

2.1.2 LSTM

LSTM networks are a specialized type of RNN designed to address fundamental limitations in learning from sequential data. They fix the vanishing and exploding gradient problems that occur during training in standard RNNs (Saxena, 2023; T.J.J., 2020; Manaswi, 2018). The core innovation of LSTM networks lies in their architecture, which introduces a structured memory component that enables selective retention, updating, and output of information across time steps. This is achieved through the interaction of two key states: the cell state, which serves as the long-term memory across the sequence, and the hidden state, which captures transient, short-term information. These states are regulated by three gates: the input gate, which controls the incorporation of new data; the forget gate, which decides which information should be discarded; and the output gate, which determines what part of the memory should be propagated forward. The use of sigmoid and tanh activation functions within these gates enables fine-grained control over information flow (T.J.J., 2020). Their design ensures a more stable gradient propagation, making them suitable for applications that require understanding long-term dependencies such as natural language processing, time-series forecasting, and speech recognition (Saxena, 2023; Manaswi, 2018). Moreover, due to their architectural advantages, LSTMs have become a foundational tool in deep learning workflows for sequential data. They are commonly implemented using high-level frameworks such as Keras, which facilitate batch training and efficient handling of large-scale datasets (T.J.J., 2020).

2.1.3 GRU

GRUs are a specific type of a RNN architecture designed to model sequential data by maintaining a hidden state that evolves over time. It is similar to LSTM units, as they also incorporate internal gating mechanisms to retain or discard information. This helps to capture long-term dependencies while also mitigating the vanishing gradient problem. In a GRU there are two gates, the update gate and the reset gate, which control to what extent the current hidden state is preserved and how much of new information is added to that hidden state at each time step. In comparison to LSTMs, they are computationally more efficient because of their simpler structure with one gate less, while still obtaining competitive performance (Chung et. al., 2014). The core gating mechanism of GRUs which controls how much information is kept makes them robust for short and noisy sequences, which is beneficial for real world traffic data. Instead of learning every detail, GRUs learn which part of the sequence is important for making accurate trajectory predictions. In this paper, GRUs are both applied to trajectory and collision prediction, as the input data for those cases is similar. The GRUs receive an input sequence which is one trajectory sequence from the dataset, for each time step, there are 8 features, X-position, Y-position, speed, acceleration, angle, change of angle, distance to nearest crossing and timespan from first datapoint to current datapoint.

2.1.4 CNN

A Convolutional Neural Network (CNN) is a type of neural network that learns features from

data using layers of filters performing mathematical convolutions, enabling the network to extract hierarchies of spatial features from grid-formatted data, such as images or matrices in general (Niu, Saad Saoud, & Hussain, 2024).

In addition, CNNs can also be used to work with sequential information, and one of the main advantages of this type of model over others is their remarkable speed, enabling easier real-time analysis and data processing by leveraging parallelisation techniques, such as the use of batches during model training (Gehring, Auli, Grangier, Yarats, & Dauphin, 2017).

2.2 Research background

2.2.1 Collision Prediction with imbalanced dataset

A central challenge in the development of robust collision prediction models is the inherent class imbalance, where crash events are rare compared to normal driving behaviour. Li & Zhang (2025) address the challenge of unlabelled and imbalanced data by proposing an active generative oversampling framework that synthesizes realistic minority-class samples (i.e., pre-collision events) using a combination of Query by Committee (QBC), Auxiliary Classifier GAN (ACGAN), and Wasserstein GAN (WGAN). The model dynamically balances training epochs between the generator and discriminator based on loss differences. Basso et al. (2021) propose a multi-input deep learning architecture that fuses traditional aggregated traffic data with a novel barcode dataset—a high-resolution, image-like representation of vehicular micro-behaviors. This dual-input approach is processed through separate CNNs and concatenated to capture both macroscopic and microscopic traffic patterns. To handle the severe data imbalance, SMOTE and DCGAN with random undersampling are used as oversampling techniques.

In contrast, model interpretability and robustness under faulty conditions are the focus of the hybrid collision prediction framework presented by Lee et al. (2024). Their method integrates scenario-based trajectory prediction via Kalman filters with CNN-based data-driven analysis, visualized through a simplified bird's-eye view (SBEV). The use of active learning enables the model to continuously improve by learning from prior misclassifications. Aditya et al. (2022) emphasize architectural innovation by incorporating Squeeze-and-Excitation (SE) blocks into the ResNeXt architecture, enhancing feature recalibration and learning efficiency. The model, trained on synthetic GTACrash data and tested on real-world datasets, attained a ROC-AUC of 0.9115, outperforming models such as VGG16, VGG19, and ResNet50.

2.2.2 LSTM for Collision Prediction

LSTM networks have become a cornerstone of trajectory prediction models, due to their ability to capture temporal dependencies in sequential data. Song et al. (2021) improve pedestrian trajectory prediction in dense crowd environments by introducing a Deep Convolutional LSTM Network (DCLN). This model replaces traditional 1D vector inputs with spatially structured tensors, enabling spatiotemporal encoding through stacked Conv-LSTM layers. An additional Global Deepening Layer (GDL) enhances feature abstraction while maintaining computational efficiency. Trained with the Adam optimizer and evaluated using

mean squared error (MSE), the architecture achieves low final prediction errors, proving effective in realistic evacuation or counterflow scenarios.

For vehicle trajectory prediction, Dai et al. (2019) propose a Spatio-Temporal LSTM (ST-LSTM) that explicitly models spatial interactions among vehicles in dense traffic. The model embeds inter-vehicle relationships using a structure inspired by Structural-RNNs, while shortcut connections (akin to ResNet) mitigate vanishing gradients and improve training over long sequences. Meng et al. (2023) integrate temporal, local, and global interaction modelling for collision risk assessment using the CSP-GAN-LSTM architecture. This hybrid model combines an LSTM encoder-decoder with Convolutional Social Pooling (CSP) and a Graph Attention Network (GAN) to capture both proximal and distant interactions in highway traffic. The model computes risk metrics such as Time-to-Collision (TTC) and Minimal Distance Margin (MDM) after trajectory prediction. Evaluated on public highway datasets, the model demonstrates superior predictive accuracy and probabilistic reliability, although future work could improve multi-agent dynamic forecasting. Xie et al. (2020) address spatial-temporal accuracy for unmanned vehicle systems through a CNN-LSTM hybrid model. The model processes kinematic features using CNNs for spatial extraction and LSTM for sequence modelling. After robust preprocessing, the model is trained to minimize Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). Altché et al. (2017) focus on medium-term trajectory forecasting in highway settings using a single-layer LSTM model paired with fully connected layers. Their model incorporates a bypass connection that directly feeds vehicle state inputs to the output layer, allowing the LSTM to focus on dynamic variations. Contrary to common practices, stacking LSTM layers did not yield performance improvements, highlighting that a shallow LSTM architecture suffices when designed with complementary components. Finally, Rossi et al. (2021) tackle two underexplored challenges in LSTM-based trajectory prediction: multimodality and generalization. Using Floating Car Data (FCD) from multiple Italian cities, they compare a stateless LSTM for unimodal predictions with a GAN-based model (GAN-3) for multimodal outputs. Their LSTM model, featuring two stacked layers followed by dense outputs, performs well in deterministic scenarios but struggles with trajectory diversity. Evaluation using normalized average and final displacement errors (N-ADE, N-FDE) shows that GAN-based models outperform LSTM in capturing multiple plausible futures, while LSTM remains effective in low-variance contexts.

2.2.3 GRU for Collision Prediction

Previous research on trajectory forecasting gives a good idea on why using GRU is a suitable approach for the research problem. Jeong et. al. (2021) applied GRUs to predict highway speeds, concluding that the GRUs are outperforming auto-regressive integrated moving average model (ARIMA) and LSTM models in both accuracy and training speed, which shows that a GRU based approach is suitable for traffic time-series scenarios. Another previous work, Benterki et. al. (2020) used a combined approach using LSTM for maneuver prediction and GRU for trajectory prediction. This work showed that GRUs can effectively model future positions of vehicles. Zarzycki et. al (2022) deployed GRU and LSTM within a Model Predictive Control framework. This work comes to the result, that GRU based models offer a favourable trade-off between accuracy and computational efficiency for real time control systems. In another context, Chung et. al. (2014) conducted a foundational experiment where GRUs and LSTMs were compared on general sequential tasks, like music and speech. While this is not in the context of neither trajectory prediction nor collision prediction, it highlights

GRUs general efficiency. So far there has been no relevant research done for using GRUs in binary collision prediction. While existing works use GRUs for continuous outputs like speed or position, collision prediction has been left out so far. However since the prediction of collisions can be based on the sequential behaviour of a vehicle, GRU based models seem to be an appropriate approach to tackle that problem.

2.2.4 CNN for Collision Prediction

The application of CNNs in the field of collision detection and trajectory prediction is well established, primarily due to their capability to work with images or grid-formatted data. Submodels such as MAD-CNN (Modularized Attention-Dilated CNN) have been used for collision detection in highly sensitive robots, achieving good results with low execution and inference times (Niu, Saad Saoud, & Hussain, 2024). Additionally, as demonstrated by Shilpa et al. (2023), CNNs can be used in combination with other algorithms to improve overall performance, such as in the case of GRU+CNN for real-time collision analysis and detection in vehicle videos.

However, the use of CNNs without explicitly having images or video within the data to be processed is less common. Nonetheless, this is not a limitation per se, as although images and videos are the primary input for CNNs, they are not theoretically restricted to these. As previously explained, it is sufficient to use information with a certain type of spatial distribution to leverage the ability of CNNs to detect inherent patterns and extract information. For example, it is possible to generate influence maps over sequential trajectories for traffic accident detection (Zhang, Zhu, & Sun, 2023).

2.3 Why it is important to have a look at different approaches

The main reason for taking a look at the different approaches is the different strengths and weaknesses they offer. Optimally this paper can offer important preliminary work which could be used as a foundation for future work, where the strengths of the approaches are combined to create a solution that improves the safety of people in the traffic.

For this specific scenario it was not sure whether GRU or LSTM is better to use, since some subsets of the data are very sparse. The GRU based models outperform LSTM models when working on small datasets, according to Yang et. al. (2020), which is why using GRU for the sparsely populated datapoints is advantageous, however for the other parts of the data the advantages of LSTM can come in handy. CNNs have rarely been used for trajectory and crash prediction and are usually used for image-like data, but there is work done on using CNNs for sequence learning. Gehring et. al. (2017) manages to outperform LSTM in a different scenario, but based on that, it is still promising to try a similar approach in the traffic scenario.

3 Method

This study focuses on predicting and preventing collisions between cars and bicycles using trajectory data collected from a roundabout. We implement and evaluate three deep learning models—LSTM, GRU, and CNN—to perform two key tasks: trajectory prediction of vehicle positions and collision prediction based on those forecasts. Each model is trained and tested separately for both tasks. Trajectory prediction is evaluated based on two metrics: average inference time per sample and MAE, which together provide a measure of both the computational efficiency and the accuracy of predicted trajectories. MAE is well-suited for this task as it directly quantifies the average deviation of predicted positions from the true positions, providing an interpretable measure of trajectory forecasting accuracy (Goodfellow et al., 2016). Inference time is crucial for assessing the models' feasibility for real-time applications, where quick response times are vital (Fridovich-Keil et al., 2019). Collision prediction is evaluated using accuracy, precision, F1 score, and recall. These metrics offer a comprehensive view of model performance, balancing the need to correctly identify collision scenarios (true positives) while minimizing false alarms (false positives) and missed detections (false negatives) (Sokolova & Lapalme, 2009). Accuracy provides an overall correctness measure, precision highlights how often predicted collisions are correct, recall focuses on capturing actual collisions, and F1 score combines precision and recall into a single balanced metric. Comparing these approaches allows us to analyse trade-offs between predictive performance and computational efficiency, thus ensuring a robust and practical solution for real-time collision prediction systems.

4 Implementation

4.1 Dataset

The dataset used belongs to the company Viscando AB. It consists of position data for different types of vehicles at a roundabout in Gothenburg, represented as pairs of X and Y coordinates, along with timestamp information, speed, vehicle type, ID, and an indicator specifying whether the data was computationally estimated or directly captured. The original raw dataset comprised around 1.1 million rows, where each row was a point of a vehicle. The scope of our project was specifically defined around the interaction between bicycles and light vehicles (cars) at the two crossing points of the roundabout located in the south-eastern section. In detail, these are the features of the original dataset:

Feature	Description
ID	Unique ID of the vehicle. Each ID appears in multiple rows, with each row corresponding to a datapoint in the trajectory of a unique vehicle.
Time	Timestamp of the row.
X	X position of the vehicle. It ranges from -37.21 to 37.21, with positive values meaning east and negatives west. Measured in meters.
Y	Y position of the vehicle. It ranges from -43.76 to 43.76, with positive values meaning south and negatives north. Measured in meters.
Speed	Speed of the vehicle at the current point.
Type	Type of vehicle. 0 for pedestrians, 1 for bicycles, 2 for light cars, and 3 for heavy cars.
Estimated	Binary value. 0 means the point is a camera reading, 1 means it was calculated by interpolation.

Table 1: Features of the dataset

The data provided already included a clean subset corresponding to vehicles approaching from the south and turning right towards the east. In addition to this subset, our group focused on cleaning the data for cars exiting the roundabout towards the south and joining the parallel road, as well as bicycles crossing the southern zebra crossing and the eastern one.

To achieve this, the original raw dataset had to be pre-processed, cleaned, and handled collectively. Overall, the objective was to gain clarity on the available trajectories, eliminate erratic data, and properly segment the various intersections or “risk points” present in the study area.

Regarding the features used, all columns from the original dataset were relevant except for the “Estimated” column, which indicated estimation. This column was not used in any of the group’s implementations.

In detail, to clean the trajectories, simple logical filters were initially applied, such as retaining only those vehicles with at least two data points. Each sector was then modelled as a polygon to allow for the logic of keeping only vehicles that passed through specific areas. The image below illustrates the implementation of polygons to determine the bicycles crossing the eastern zebra crossing:

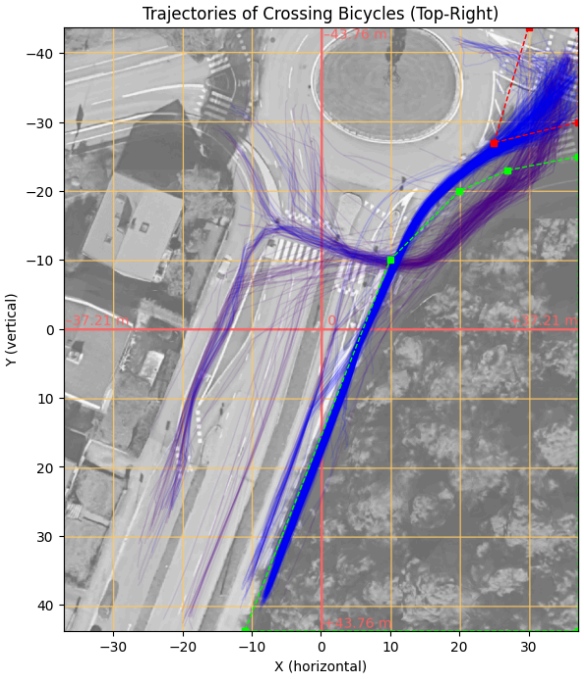

Figure 1: Trajectories of some crossing bicycles

Here, we can see that the southern and northern areas of the zebra crossing are highlighted with green and red polygons, respectively. The idea was to retain only those trajectories that contained at least one point within both of these sections, thereby ensuring that they were indeed “crossing the street” in some way. The distinct blue and purple colouring refers to possible secondary trajectories that bicycles could take: the purple trajectories correspond to bicycles that also crossed the southern zebra crossing, while the blue trajectories are those of bicycles continuing straight to or from the south.

Thus, we ended up with four clean data subsets: cars approaching from the south and turning east, cars exiting the roundabout towards the south, bicycles crossing the southern zebra crossing, and bicycles crossing the eastern zebra crossing. The vehicle IDs belonging to each of these subsets were combined to create the final dataset used in the following sections of this report. This final dataset was around 446.000 rows distributed in around 14.000 vehicles.

4.2 LSTM

4.2.1 Trajectory Prediction

Three distinct architectures were implemented: a simple LSTM model, an enhanced LSTM with batch normalization and early stopping, and a more complex sequence-to-sequence LSTM model with TimeDistributed layers. This chapter details the methods applied, the rationale behind model selection, and the training procedures for each model.

To prepare the data for model training, sequences were constructed from vehicle trajectories. Sequences were padded to standard lengths to accommodate variable trajectory lengths, using a masking value of 0.0 to indicate padding. This allowed the models to distinguish meaningful data points from padding during training. For Model 1, a smaller subset of the dataset was used (first 1,000 samples), with input shapes of (1000, 5, 4), representing sequences of 5 points with 4 features (x, y, speed, type). The target output was the subsequent point in the sequence (the 6th point), in (x, y) format. For Models 2 and 3, the full dataset (404,911 samples) was used. The sequences were dynamically constructed, progressively incorporating more points (starting with 2 points to predict the 3rd, 3 points to predict the 4th, etc.) to model more complex movement patterns. Sequences were padded and masked to (404,911, 126, 4) for inputs. Model 2's output was a single future point (404,911, 2), while Model 3's output was a sequence of points including the future point at the end (404,911, 126, 2).

Model 1 was designed to establish a baseline performance using a simple LSTM architecture. This model began with a Masking layer to handle padded inputs, followed by two LSTM layers—one with 256 units and another with 128 units. These layers were configured to capture the temporal dependencies in vehicle trajectories, with the first LSTM layer set to return sequences and the second to return only the final output. To prevent overfitting, Dropout layers with a dropout rate of 0.2 were added after each LSTM layer. The final stage of the model consisted of a dense layer with 64 neurons and ReLU activation, followed by an output layer predicting the next x and y coordinates. The model was compiled using the Adam optimizer and mean squared error (MSE) as the loss function, with the smaller dataset used for training.

- Masking(mask_value=0.0, input_shape=(X.shape[1], X.shape[2])),
- LSTM(256, return_sequences=True),
- Dropout(0.2),
- LSTM(128, return_sequences=False),
- Dropout(0.2),
- Dense(64, activation='relu'),
- Dense(2)
- Output shape: (1000, 2)

Building on Model 1, Model 2 introduced additional techniques to improve generalization and accuracy. It employed a similar LSTM architecture but added Batch Normalization layers after each LSTM to stabilize learning and accelerate convergence. Dropout layers with a 0.2 rate were retained to control overfitting. Unlike Model 1, this model was trained on the full dataset, enabling it to learn from a richer set of trajectory patterns. To optimize training, two advanced techniques were incorporated: EarlyStopping, which halts training when validation

loss no longer improves (with a patience of five epochs), and ReduceLROnPlateau, which reduces the learning rate by half when the model's performance plateaus (with a patience of three epochs). These enhancements were designed to maximize model accuracy while avoiding overfitting or excessive training.

- Masking(mask_value=0.0, input_shape=(X_padded.shape[1], X_padded.shape[2])),
- LSTM(128, return_sequences=True),
- BatchNormalization(),
- Dropout(0.2),
- LSTM(128, return_sequences=False),
- Dropout(0.2),
- Dense(64, activation='relu'),
- Dense(2)
- Output shape: (404,911, 2)

Model 3 employed a similar LSTM backbone but configured both LSTM layers to return sequences. Batch Normalization and Dropout layers were included to enhance model robustness and reduce overfitting. A distinguishing feature of this model was the use of TimeDistributed dense layers, which applied a dense transformation to each time step of the sequence. This allowed the model to produce an output of the same length as the input, predicting the next x and y coordinates at every point in the sequence. The model was trained on the full dataset, employing the same EarlyStopping and ReduceLROnPlateau strategies as Model 2 but with a longer patience of ten epochs to accommodate the model's greater complexity.

- Masking(mask_value=0.0, input_shape=(X_padded.shape[1], X_padded.shape[2])),
- LSTM(128, return_sequences=True),
- BatchNormalization(),
- Dropout(0.2),
- LSTM(128, return_sequences=True),
- BatchNormalization(),
- Dropout(0.2),
- TimeDistributed(Dense(64, activation='relu')),
- TimeDistributed(Dense(2))
- Output shape: (404,911, 126, 2).

Model 1 was trained for 50 epochs with a batch size of 32 on the smaller dataset (1,000 samples). The simple architecture and limited data allowed rapid convergence. Model 2 was trained for up to 80 epochs with a batch size of 64, incorporating EarlyStopping (patience=5) and ReduceLROnPlateau (factor=0.5, patience=3) to optimize training. These strategies ensured the model stopped training at optimal performance while dynamically adjusting learning rates. Model 3 required a more extensive training regime, running for up to 100 epochs with a batch size of 32. EarlyStopping with a longer patience (10 epochs) and learning rate adjustments were used to stabilize the training of this more complex model. All models were compiled with the Adam optimizer and MSE loss. The training and validation losses were monitored to identify overfitting and underfitting patterns.

The selection of LSTM architectures across all models was motivated by their proven ability to capture temporal dependencies, which are crucial for modeling vehicle movement over

time. The enhancements in Models 2 and 3, including batch normalization, dropout, and sequence-to-sequence prediction, were implemented to improve generalization, reduce overfitting, and provide more detailed forecasts, respectively. Training strategies for Models 2 and 3 also included dynamic learning rate adjustments and early stopping to prevent overtraining and to restore the best weights based on validation loss. This iterative approach allowed for a comprehensive exploration of model performance across different complexities and data scales. The inclusion of vehicle type and speed as features was motivated by their direct influence on trajectory behavior. Cars and bicycles exhibit different movement dynamics, and speed plays a crucial role in trajectory changes. Masking and padding were essential for handling variable-length sequences and maintaining batch uniformity during training.

4.2.2 Collision Prediction

Collision detection aims to identify situations where a vehicle and a bicycle are at risk of colliding, enabling proactive safety interventions. This chapter details the data preparation, model design, training process, and the reasoning behind the selected methods for collision detection. Each entry in the dataset includes x and y positions of cars and bikes, speeds of cars and bikes, the Euclidean distance between the car and the bike at a given time and a collision label.

To speed up the algorithm training, only the first 8,000 rows of the initial dataset were used for collision prediction. For each vehicle-bicycle pair (ID_car, ID_bike), the data was grouped, and for each group, a sliding window of size 10 was applied. The sequence comprises seven features: [X_car, Y_car, Speed_car, X_bike, Y_bike, Speed_bike, Distance] and the target label for each sequence is the collision indicator at the subsequent time step, simulating a predictive rather than reactive approach. The resulting dataset has an input shape of (1001, 10, 7) and a target shape of (1001,), with each entry representing a sequence of temporal observations and its corresponding collision label.

The model for collision detection leverages a deep LSTM network to effectively model the temporal dependencies in the data. The architecture consists of:

- An LSTM layer with 64 units, returning sequences to capture long-range dependencies across the 10 time steps.
- A Dropout layer with a dropout rate of 0.2 to mitigate overfitting.
- A second LSTM layer with 32 units, returning the final output sequence embedding.
- A Dense layer with 16 units and ReLU activation to map the sequence embedding to a lower-dimensional representation.
- A final Dense layer with 1 unit and sigmoid activation, outputting a collision probability for each input sequence.

The model was compiled with the Adam optimizer (learning rate 0.001), binary cross-entropy loss, and accuracy as the primary metric. The model was trained for 10 epochs with a batch size of 64. The model was trained using the model.fit() method, with validation data provided to monitor performance. The selection of an LSTM-based architecture is motivated by its demonstrated strength in modelling time-dependent phenomena. Including distance and speed features allows the model to capture both absolute movement patterns and relational dynamics between vehicles and bicycles. The use of binary cross-entropy aligns with the

binary nature of the task (collision vs. no collision).

4.3 GRU

For the GRU based models, different implementations were tested. The first implementation is a very basic setup, consisting of:

- GRU-Layer with 64 units
- GRU-Layer with 32 units
- Dense-Layer with 16 units, ReLU activation
- Output-Layer, Linear for trajectory prediction, Sigmoid activation for collision prediction

The model bases are equal for both trajectory and collision prediction; the difference is in the output layer and the loss to appropriately handle the task. For the trajectory prediction, MSE loss is used and for the collision prediction the binary cross-entropy loss is used. The optimizer used is Adam, as it is a general solid optimizer that works well in many different scenarios.

In addition to this, an early stopping mechanism was used to train the model efficiently. Also, learn-rate scheduling, keras' ReduceLROnPlateau was used to overcome local minima in the training process and create a more optimized model.

Different model architectures were tried; however, this specific setup has the most promising results. This specific implementation balances model capacity and computational efficiency. Using a single GRU-Layer resulted in less accurate predictions, while using more than two GRU-Layers did not significantly increase the quality but increased the computational needs. This setup is enough to capture the temporal patterns in motion without overcomplicating the model.

Another GRU implementation has a similar setup but uses bidirectional GRU layers instead of the unidirectional GRU layers which are used in the first approach. The reason for testing this approach was to investigate whether using future context within the fixed input sequence would benefit the model. This approach did not show any positive effect on the results because of the length of the input sequence, which is too short with a length of 5. With this limited input, the backward part of the bidirectional GRU has too little additional information to learn anything meaningful. The input sequence length of 5 was chosen because that is the lowest number of trajectories before reaching a crossing. It is important to only account for the trajectories before a crossing, as the focus lies on collisions at a crossing and not on how the car or bike behaves after passing a crossing.

Since the training of the trajectory models did not show signs of overfitting, dropout layers or other methods to prevent overfitting were not implemented as they would increase the computational time to train the models without having significant benefits.

The output of the trajectory prediction model was set to predict the mean amount of trajectories that were found inside one crossing area, which were 5.

To use the data for collision prediction, it is necessary to first identify collisions or situations where it comes close to a collision. Since there were no collisions captured in the dataset, a different approach was needed. To determine if there was a situation close to a collision, the trajectories were modified by adding a temporal and spatial threshold. The time threshold was set to 2.5 seconds and is based on findings by Lubbe et. al. (2014). Their results showed that Time-To-Collision (TTC) ranged from 2.1 to 4.3 seconds with a median of 3.2 seconds. 90% of drivers began braking at 2.5 seconds or before, which is why this specific time threshold is suitable for a nearby collision, as this is the last point of time before the collision where most drivers can react.

For the distance threshold 5 meters seemed to be suitable for this scenario, incorporating the average size of a car combined with the size of a bicycle. The findings of Fakhoury et. al. (2023) support the use of a 5-meter distance threshold in low-speed driving environments. This threshold balances sensitivity and specificity allowing for detection of potential collisions without excessive false positives. Using these thresholds alone did not result in any collisions, which is why another measure was taken to generate synthetic collisions. To finally get collisions while still maintaining the recorded behaviour of both cars and bicycles, the absolute time which is present in the dataset was removed and instead the relative time was used. The relative time was created by first setting the time of each trajectory to zero and then adding the time difference between each trajectory on. This ensures that the trajectories keep their original shape and amount of information.

The collision prediction model was trained on a subset of the artificial collisions, because all bicycles had multiple collisions with different cars. For each of the 450 bikes 20 interactions were taken with cars, both with and without collisions, resulting in a set of 9000 unique interactions.Of that interaction dataset, 10% were used to test the model. It was ensured that there is a big variety in the car trajectories for the interactions by not reusing the same unique car for an interaction if possible, so that the model does not focus on the specific movement of one car. The trajectories used were cut off after passing one crossing, since the collision in a crossing area is the point of interest in this project. The input was set to take 10 trajectories, using 5 trajectories from inside one crossing area and 5 trajectories from before. This was chosen because of how the trajectory prediction model was structured, which takes 5 trajectories as input and predicts 5 trajectories inside the crossing area. For vehicles that are crossing both crossing areas, the trajectories were split to properly fit the model structure.

To evaluate the trajectory prediction model, the mean average error (MAE) is used, as it allows one to see how far off the model is when predicting a trajectory. Since the input data was normalized before it was used to train the model, it is important to note that the test result then is based on the normalized values instead of the actual values. The result needs to be denormalized, which is done by applying the same scale used to normalize backwards. Further it is important to note that this model predicts multiple trajectories at once. This was chosen to ensure that the vehicle's trajectories are predicted over the entire area of a crossing. The collision prediction model was evaluated using accuracy, precision, recall and F1-Score. To set up the data for evaluation, a distinct set of data, which is similarly structured as the training data, was used.

4.4 CNN

4.4.1 Trajectory Prediction

The first of the CNN-based implementations was a trajectory prediction system. Building on the previously cleaned and processed dataset, this stage involves using a 1D CNN to predict the next position of a vehicle *(x, y)* given its previous trajectory (the set of positional points associated with the same ID). The original sequence of points is chronologically ordered to ensure data integrity.

Due to the irregular nature of the data, there are several cases where vehicle trajectories vary in length. To address this issue, batch processing and padding techniques were applied jointly. The records were first grouped into batches of size 32, after which shorter sequences were padded with zeros until they matched the length of the longest sequence within the batch. This ensures that the model can process the data, as it is not possible to work with sequences of varying lengths—tensors must have uniform dimensions.

The model was designed to work under the logic of using three main features to calculate the position using two variables: slope and speed. The slope of the point to be predicted corresponds to the differences in the "X" and "Y" positions of the points immediately before the one to be predicted, and determines the direction of the prediction vector, while the speed, obtained from the “Speed” feature, determines the magnitude of that vector. Thus, the features to be used would be *[X, Y, Speed]* for each point, and the goal would be to find *[X, Y]* for the point to be predicted.

After several iterations of trial and error, fine-tuning, and general adjustments, the final model is a sequential 1D CNN that receives tensors of variable-length points, where each point has the form *[X, Y, Speed]*:

```
model = Sequential([
    Input(shape=(None, 3)),
    Conv1D(filters=64, kernel_size=3, activation='relu'),
    Conv1D(filters=64, kernel_size=3, activation='relu'),
    GlobalAveragePooling1D(),
    Dense(64, activation='relu'),
    Dense(2)
])
```

Figure 2: CNN architecture for trajectory prediction

The idea of a sequential network arises from the premise that the input data structure can be understood as a univariate time series in each dimension. This particular design is strong in pattern detection for this type of problem, while the 64-filter layers with a kernel size of 3 were found empirically and offer a good balance between feature detection and sensitivity: this number adequately detects abrupt changes in speed and direction without overfitting the

prediction, for example. A standard ReLU activation function is used.

Right after, the GlobalAveragePooling1D layer converts the output sequences from these convolutional layers into a fixed-size vector to prepare the data for the next step, the Dense layer. This pooling layer helps reduce the risk of overfitting by not introducing additional parameters into the model, reducing dimensionality and boosting generalization, which is why it was preferred over alternatives such as Flatten (Hosla, 2025). The final layer simply outputs two values, X and Y, corresponding to the coordinates of the predicted point.

The training and testing process was carried out using a random 80/20 train/test split, with the model employing Adam as the optimiser and MSE as the loss function.

4.4.2 Collision Detection

The collision detection process began by grouping the points by ID, so that each unique ID (each vehicle) was represented as a list of points; then, bicycles were separated from cars.

At this stage, we encountered a significant data-level obstacle: not only are there no actual collisions in the raw data, but there are also no scenarios where a bicycle and a vehicle are in close proximity during the same time interval. If left unaddressed, this would prevent the model from learning what a collision looks like and result in a poor model. Therefore, the decision was made to train the CNN using synthetic data, as follows:

The data used to train the model corresponds to the same trajectories from the original dataset, but all shifted to occur at the same point in time. This results in numerous cases where bicycles and cars intersect (even though, in reality, these trajectories may have occurred at very different times), allowing us to train a model capable of distinguishing between non-intersecting trajectories (no collision) and intersecting ones (potential collision). A total of 10.000 bike-car pairs were used for training.

As with the trajectory prediction model, variable trajectory lengths had to be handled. Upon examining the data, we found that the distribution of the number of points per trajectory is well populated up to around 47 points; trajectories with more points are less common. As a result, padding and truncation were applied to standardise the length of all trajectories to 47.

The model architecture begins with the input (T, 4), where the positions of the car and the bicycle to be evaluated are used along with the sets of trajectory points. Specifically, the CNN model uses a pair of convolutional layers with different numbers of filters (64 and 128). These values were found experimentally by testing different configurations, as they yielded the best results by better learning the patterns in the data and generalizing more effectively. Additionally, two Pooling layers are used again, as in the previous trajectory prediction model, and a Dropout layer is introduced to prevent overfitting; the latter was necessary because, in the initial experimental iterations, the model had generalization issues. The output layer uses a sigmoid activation function, reflecting the binary nature of the problem (Collision vs. No Collision). Adam was used as the optimiser, and the loss function was Binary Cross Entropy, as this is a binary classification task.

```
model = Sequential([
    Conv1D(64, kernel_size=3, activation='relu', input_shape=(T, 4)),
    MaxPooling1D(pool_size=2),
    Conv1D(128, kernel_size=3, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Figure 3: CNN architecture for collision detection

The reason for choosing a sigmoid function at the output instead of, for example, a strictly binary output—despite this being interpreted as a binary classification problem—was to have an indicator of the prediction's confidence. A value very close to the sigmoid range limits (0 or 1) indicates confidence in the prediction, while intermediate values do not. This can be valuable when analyzing the model's reliability.

The training process was conducted over 20 epochs with a batch size of 64 and using early stopping to avoid additional computational cost in case the model achieved good results before the full number of epochs. The validation subset size was set to 0.2.

To test the model, a new subset of bike-car pairs was prepared from the original data. This time, the actual timestamp was taken into account, using a 10-second threshold to determine whether two vehicles were within the same "time window." Thus, for all bicycles in the dataset, only those cars whose trajectories fell within this time range were compared, effectively comparing them only with vehicles on the road at the same time.

Finally, considering that the CNN output is sigmoid, it was necessary to threshold the output into binary labels in order to compute performance metrics as a classification problem. For this purpose, a Grid Search technique was employed to select the most appropriate threshold value to maximise our target metric, the F1 Score. This method explored thresholds from 0.001 to 0.999 in increments of 0.001.

5 Results

5.1 Results LSTM

5.1.1 Trajectory Prediction

This chapter presents the results obtained from the implementation and evaluation of the three trajectory prediction models. The performance of each model was assessed based on two primary metrics: mean absolute error (MAE) of predicted trajectories, which reflects prediction accuracy, and average inference time per sample, which provides insight into computational efficiency.

Model	Dataset	MAE (meters)	Inference Time (milliseconds)
1	Small (1000 samples)	1.09	1.23
2	Full (404,911 samples)	0.59	7.614
3	Full (404,911 samples)	0.54	12.446

Table 2: Evaluation of different LSTM models

The first model, trained on a subset of the data comprising 1,000 samples, achieved an MAE of 1.09 meters and an inference time of approximately 1.23 milliseconds per sample. While this model demonstrated rapid inference capabilities, the prediction accuracy was comparatively lower. This outcome is likely attributable to the limited data used for training, which restricted the model's ability to generalize across diverse vehicle trajectories.

In contrast, the second model was trained on the complete dataset of 404,911 samples, allowing it to learn a broader range of trajectory patterns. This model achieved a significantly improved MAE of 0.59 meters, indicating a higher degree of accuracy in predicting vehicle positions. However, this improvement came at the cost of a longer inference time, averaging approximately 7.614 milliseconds per sample. The balance between accuracy and computational cost demonstrated by this model highlights its potential suitability for real-time applications where both performance and efficiency are critical.

The third model, employing a sequence-to-sequence LSTM architecture with TimeDistributed dense layers, achieved the best overall prediction accuracy with an MAE of 0.54 meters. However, the enhanced prediction accuracy came with the highest computational cost, as evidenced by an inference time of approximately 12.446 milliseconds per sample. This trade-off indicates that while Model 3 offers superior prediction capabilities, its computational demands may limit its applicability in scenarios with stringent real-time requirements.

The results highlight a clear trend: increasing model complexity and data size improves trajectory prediction accuracy but also leads to higher computational costs. Model 1, with its minimal data and architecture, offers the fastest inference time but with reduced accuracy. Model 2 represents a balanced approach, achieving substantial accuracy improvements while maintaining moderate computational efficiency. Model 3 achieves the highest accuracy and temporal detail but at the expense of increased inference time.

In conclusion, Model 2 is the most suitable choice, since it offers a practical compromise between accuracy and efficiency. Model 3 is useful for applications prioritizing precise trajectory prediction and detailed temporal context, such as high-stakes collision avoidance systems in roundabouts. Model 1, while efficient, is best suited for exploratory analysis or scenarios where rapid but approximate predictions are acceptable.

5.1.2 Collision Prediction

Key evaluation metrics include accuracy, precision, recall and F1 score, alongside a confusion matrix to illustrate the model's classification behaviour.

- **Accuracy:** 0.921 - This indicates that the model correctly classified approximately 92.1% of the sequences in the validation set.
- **Precision:** 0.667 - The model correctly identified 66.7% of the sequences it predicted as collisions, suggesting a moderate rate of false positives.
- **Recall:** 0.667 - The model detected 66.7% of actual collisions, indicating a moderate rate of false negatives.
- **F1 Score:** 0.667 - The harmonic mean of precision and recall, highlighting the model's balanced performance between sensitivity and specificity.

The confusion matrix reveals that the model correctly identified 127 out of 133 non-collision sequences and 12 out of 18 collision sequences. While the model exhibited a low rate of false positives (6 cases) and false negatives (6 cases), its balanced precision and recall indicate a trade-off between over-predicting collisions and missing true positives. The collision detection model demonstrated high overall accuracy (92.1%), underscoring its effectiveness in identifying potential collisions in the roundabout scenario. However, its moderate precision and recall (both 0.667) suggest room for improvement, particularly in reducing false positives and enhancing sensitivity to actual collisions. In practical applications, these results imply that the model can reliably detect most collision scenarios, though occasional false alarms and missed collisions may occur. Balancing these trade-offs will be essential for real-world deployment, where false positives may lead to unnecessary interventions and false negatives could result in undetected hazards.

5.2 Results GRU

5.2.1 Trajectory Prediction

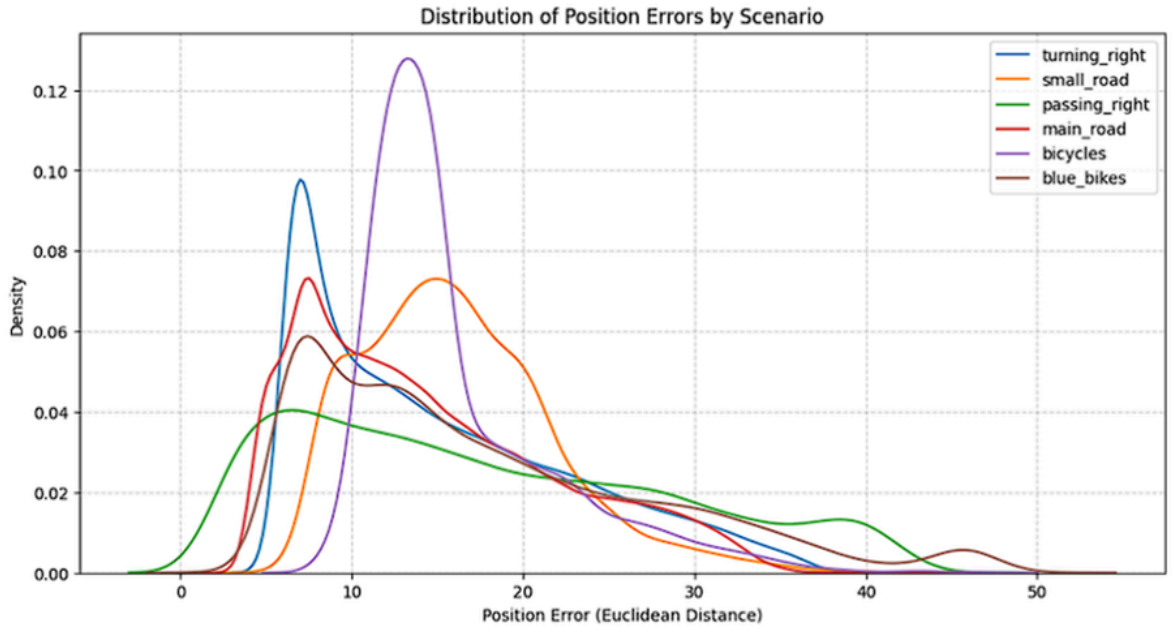

Figure 4: Comparison of position error for different scenarios

The basic GRU implementation shows promising results for the trajectory prediction. The MAE on a separated test set is at 0.0393. Calculating the MAE backwards by denormalizing it, because it has been scaled for training, this results in an inaccuracy of 40 centimeters. Comparing this to the size of a cyclist or a car, which have a bigger average size, it is a result that is suitable for further work on the collision prediction.

In figure 4 the different scenarios were separately analysed. The “position error” axis has been scaled to better see the differences for each scenario.
It is not surprising that the scenario “small_road” is not as good as the others, because there was only data on 295 cars, compared to the scenario “main_road” which had data on 11021 cars.

The average inference time for predicting five trajectories is at 0.53 ms.

5.2.2 Collision Prediction

The results for the collision prediction model using the GRU based approach are quite similar to the results of the LSTM based approach. The model achieved an accuracy of 76.7%. The model has a precision score of 0.672, meaning that it identified 303 out of the 450 non-collisions correctly, leaving 147 cases as false positives. This relatively high number of false positives however is acceptable, if the main concern is safety. The recall was at 0.828, which means that the model missed a total of 63 from the 450 possible collisions, while correctly identifying 387 collisions. The resulting F1-score is at 0.743. The collision prediction model has room for improvements, especially when it comes to wrongly detecting trajectories

as a possible collision. Having that many false alarms might possibly impact the traffic flow negatively by drivers breaking unnecessarily.

5.3 Results CNN

5.3.1 Results for Trajectory Prediction

The trajectories obtained through the use of this algorithm show an average error well centered around the expected value of 0. First, the horizontal coordinates (X) show a tighter error distribution around the mean (a lower standard deviation), but the average error is the largest in magnitude between the two coordinates (X vs. Y). In the case of the vertical coordinates (Y), the error is more dispersed and tends to underestimate the position, evidenced by the negative mean and the slight skew to the right.

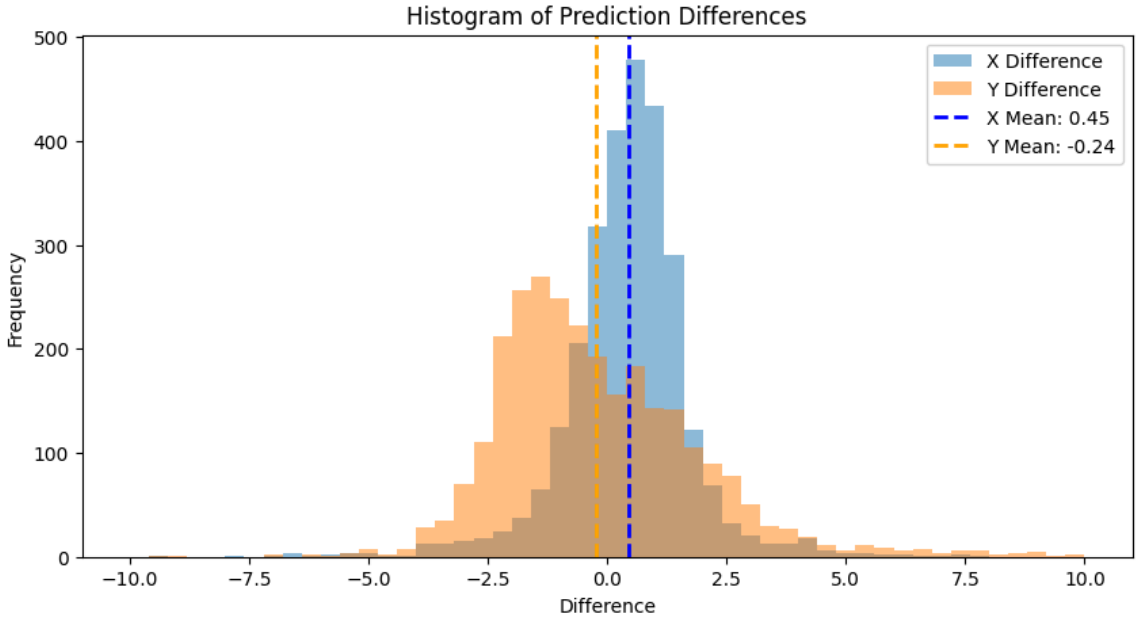

Figure 5: Histogram of prediction differences (in meters) for the final predicted point of each trajectory

The model's training MAE was 0.95, and convergence was achieved in 20 epochs. Although the results are relatively accurate, they are unstable. Nevertheless, the inference time was only 0.232 milliseconds per sample, which is notably fast.

5.3.2 Results for Collision Detection

The model was trained with a batch size of 64 for a maximum of 20 epochs; however, due to the early stopping parameter used, convergence was achieved in only 11 epochs. At that point, the accuracy reached 0.966. When the trained model was tested against real data, a grid search indicated that the optimal threshold value for binarising the output of the sigmoid function was 0.544 (any model output greater than 0.544 was classified as "1", and any output below that as "0"). With this threshold, an F1-score of 0.574 was achieved.

Regarding the other metrics, the model achieved an accuracy of 0.905, a precision of 0.355, and a recall of 0.806. Accuracy is not a particularly meaningful metric in this type of

classification problem, whereas the other two are more relevant (as they are the basis for calculating the F1-score, our target metric). First, such a low precision indicates a high false positive rate (many false alarms), while a high recall tells us that the model is successfully detecting the vast majority of collisions.

While the performance metrics are not ideal, the training process was remarkably fast (approximately 2 minutes), demonstrating the model's efficiency in leveraging parallel processing through the use of batches.

6 Discussion

6.1 Trajectory Prediction

In the three implementations presented, the trajectory prediction problem was approached in a similar way but with significantly different results. Throughout, the progressive construction of the trajectories was always taken into account, in the sense that each new point to predict depends on the previous ones. Also, the X and Y positions along with speed were the key features used to make predictions in all three models and their different versions.

In general, the task of trajectory prediction can be very useful in numerous real-world scenarios involving vehicles on a road. Here, especially with the LSTM implementation, we can see how small adjustments in the model architecture fine-tune it to better address these different scenarios: the first version of LSTM uses a tiny subset of data compared to the other two versions, but yields acceptable results for exploratory tasks or situations where maximum prediction speed is pivotal, even over high accuracy. On the other hand, the third model goes to the opposite extreme and uses all available features in the data to, through a more robust and complex architecture, achieve the highest performance at the cost of more computational power and inference time. The second version sits between the other two in this regard; it is a balanced version that captures the best aspects of the others and works well for general tasks.

However, if the goal is maximum prediction accuracy, the GRU model achieves this most effectively. Obtaining the lowest MAE among all the models presented and also having the shortest average inference time per sample, it is the leading model in that regard. As a drawback, it is worth noting the long training time required by this type of model, which means a high computational cost only during training and not really during testing.

Thus, a good application of this architecture could be in the analysis of previously collected data where ample time is available for training. Then, this model could be deployed for real-time analysis as a testing set.

Finally, the CNN model is the most lightweight of all. It is a simple but effective model. The model's architecture allowed for excellent parallelization during the training process using batches and handling trajectories of variable length, which resulted in very short training times with acceptable MAEs for these conditions. Although it does not stand out for high accuracy, this inherent flexibility is a strong advantage when adapting the model to different types of applications for this problem, such as live detection, post-analysis, alert sensors, etc.

6.2 Collision Prediction

The collision prediction task was done using three different approaches, of which two, namely LSTM and GRU are quite similar in their structure. The CNN approach vastly differs in its structure and results in very different outcomes compared to the others. On top of the inherent differences in structure, the different data preprocessing strategies make it hard to just compare the different metrics without acknowledging the different strategies.

While the LSTM model was trained on 8000 rows, the data used for testing only included 18 collision cases and 133 non-collision cases. This imbalance reflects the lack of collision data,

which was synthetically created for all three models. But it also leads to a bias in the model itself, as the model is likely focussing on the imbalance itself rather than learning the differences which actually impose a risk for collision. It achieved the highest accuracy with 92.1% and a well-balanced precision and recall value with 0.667.

The GRU model was trained on a similar sized set containing 8100 different interactions, using the real trajectories aligned at the same point of time. It achieved an F1-score of 0.743, with a notably high recall of 0.828, which means that it is effective at identifying most collision scenarios. The lower precision value at 0.672 indicates a high number of false alarms, which might negatively impact the traffic flow. However, from a safety critical perspective, this is tolerable where false alarms are preferred over missing collisions.

The CNN has a mixed performance, with a high recall of 0.806 and a low precision of 0.355. This results in even more false alarms compared to the GRU model. This model offers high computational efficiency which could make it suitable for real time applications, where speed of both training and inference is a higher priority.

While LSTM and GRU models offer more balanced and robust predictions, CNNs also demonstrated their use case for the proposed problem of collision detection. Each method offers different trade-offs between the different metrics.

6.3 Ethical and societal aspects

The dataset used in this project contains sensitive trajectory data provided by a company under strict confidentiality agreements. Importantly, the data does not contain information about identifiable individuals; it consists of videos captured by a security camera that records vehicle and bicycle movements in a roundabout, where individual identification is not possible. We have ensured that the data is handled securely and just used for the purposes of this research. This project addresses a critical safety issue—predicting and preventing collisions between cars and bicycles. If implemented, such models could contribute to reducing traffic accidents and saving lives, aligning with the UN's Sustainable Development Goal 3: Good Health and Well-being (WHO, 2025). However, ethical deployment would require transparency, validation in diverse environments, and continued monitoring to ensure the model performs fairly across all road users without bias.

7 Conclusion

This project aimed to compare different approaches on solving the problem of collision prediction between cars and bicycles in crossing in a roundabout. The study was divided in two parts, the trajectory prediction and the collision prediction. Each model was trained and tested on the same base dataset, with variations and adaptions made to further improve the model's capabilities.

For the first part, trajectory prediction, the GRU model achieved the best performance with an error of 0.4 meters and an inference time of 5.3ms. This shows GRU's efficiency and suitability for the prediction of future movement, which is crucial for collision prediction, as it is based on those predicted trajectories when deployed in a real case scenario. The LSTM also offered a strong performance with an error of 0.59m, but due to its more complex structure it was less efficient with an inference time of 7.614ms. The CNN offered the fasted computation of future trajectories, but on the other hand it was the least accurate model with an error of 0.95m.

In the context of collision prediction, it is harder to compare the results and find a definitive answer on which model is the best, because of the differences in generating the collision data. Looking at the numbers, the LSTM model has the highest accuracy with 92.1% and a well balanced recall and precision. The GRU model outperformed it in the recall, which shows its strong ability to detect actual collisions. The CNN suffers from a low precision with 0.355, which results in a large number of false positives.

The best possible result for the overall task would be a mixed model architecture which uses the strengths of the different models combined to tackle the individual tasks of trajectory prediction and collision prediction combined.

To finally answer the proposed research questions:

- How effective are the different algorithms in prediction collisions from trajectory data?

  All models demonstrated viability, showing different trade-offs at the specific tasks

- Which algorithm achieves the highest accuracy and lowest inference time in collision prediction

  The LSTM model achieved the highest accuracy, but the CNN outperformed it in terms of inference time

7.1 Future work

Although this study contributes valuable insights into the prediction of vehicle and bicycle trajectories and collisions, there are several notable limitations that must be acknowledged. One of the most significant limitations concerns the data used. The dataset did not include any real, recorded collisions. Instead, collision labels were artificially constructed based on predefined threshold values. This approach enables the creation of a training dataset, but it does not capture the complexity or randomness of real-world accidents. As a result, the accuracy of the model in detecting actual collisions in real-life scenarios is limited. In

addition, the models developed for this study—based on LSTM, GRU, and CNN architectures—were inherently different in their internal structure and processing logic. Due to these architectural differences, it was necessary to use distinct modeling strategies and data formatting approaches for each model. While this diversity allowed for a broader exploration of methodological possibilities, it also introduced challenges when comparing the results of the models directly. This inconsistency makes it more difficult to draw definitive conclusions about which architecture performs best under uniform conditions. Another limitation arises from the way the collision detection task was treated in isolation from the trajectory prediction task. In the current design, trajectory prediction and collision forecasting were handled as two distinct processes, rather than as components of a single end-to-end system. This separation means that the potential impact of trajectory prediction accuracy on the quality of collision forecasting could not be adequately evaluated. A more integrated approach would likely offer a more comprehensive understanding of the interplay between trajectory errors and collision prediction reliability.

Looking ahead, future work should aim to address these limitations. A key step would be the use of a dataset that includes real collision events collected over a longer observational period. Such data would enable the development and validation of models that are not only more realistic but also more robust. Moreover, incorporating video-based data rather than pre-processed trajectory points could allow models—particularly those using deep learning—to extract richer spatial and temporal features, thereby improving predictive performance. Furthermore, future research could explore the combination of different neural network architectures into hybrid or ensemble models. This approach could leverage the respective strengths of LSTM, GRU, and CNN models, potentially leading to more accurate and generalizable results. Another promising direction would be the integration of more advanced features such as acceleration, relative orientation, lane positioning, and contextual signals like traffic light states or vehicle density. These additional inputs could greatly enhance the models' ability to anticipate risky interactions and predict collisions more precisely.

References

Abbaspour, S., Fotouhi, F., Sedaghatbaf, A., Fotouhi, H., Vahabi, M., & Linden, M. (2020). A Comparative Analysis of Hybrid Deep Learning Models for Human Activity Recognition. Sensors, 20(19), 5707. https://doi.org/10.3390/s20195707

Aditya, A., Zhou, L., Vachhani, H., Chandrasekaran, D., & Mago, V. (2022). Collision Detection: An Improved Deep Learning Approach Using SENet and ResNext. *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. https://doi.org/10.1109/smc52423.2021.9659265

Altche, F., & de La Fortelle, A. (2017). An LSTM network for highway trajectory prediction. *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 353–359. https://doi.org/10.1109/itsc.2017.8317913

Basso, F., Pezoa, R., Varas, M., & Villalobos, M. (2021). A deep learning approach for real-time crash prediction using vehicle-by-vehicle data. *Accident Analysis &Amp; Prevention*, *162*, 106409. https://doi.org/10.1016/j.aap.2021.106409

A. Benterki, M. Boukhnifer, V. Judalet and C. Maaoui. (2020). Artificial Intelligence for Vehicle Behavior Anticipation: Hybrid Approach Based on Maneuver Classification and Trajectory Prediction in IEEE Access, vol. 8, pp. 56992-57002. 10.1109/ACCESS.2020.2982170.

Candela, E., Doustaly, O., Parada, L., Feng, F., Demiris, Y., & Angeloudis, P. (2023). Risk-aware controller for autonomous vehicles using model-based collision prediction and reinforcement learning. *Artificial Intelligence*, *320*, 103923. https://doi.org/10.1016/j.artint.2023.103923

Chung, Junyoung & Gulcehre, Caglar & Cho, KyungHyun & Bengio, Y.. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.

https://doi.org/10.48550/arXiv
.1412.3555

Dai, S., Li, L., & Li, Z. (2019). Modeling Vehicle Interactions via Modified LSTM Models for Trajectory Prediction. *IEEE Access*, *7*, 38287–38296. https://doi.org/10.1109/access.2019.2907000

Deva Hema, D., & Rajeeth Jaison, T. (2024). Efficient Collision Risk Prediction Model for Autonomous Vehicle Using Novel Optimized LSTM Based Deep Learning Framework. *International Journal of Intelligent Transportation Systems Research*, *22*(2), 352–362. https://doi.org/10.1007/s13177-024-00399-z

Fakhoury, S., & Ismail, K. (2023). Improving Pedestrian Safety Using Ultra-Wideband Sensors: A Study of Time-to-Collision Estimation. Sensors, 23(8), 4171. https://doi.org/10.3390/s23084171

Fridovich-Keil, D., Bajcsy, A., Fisac, J. F., Herbert, S. L., Wang, S., Dragan, A. D., & Tomlin, C. J. (2019). Confidence-aware motion prediction for real-time collision avoidance1. *The International Journal of Robotics Research*, *39*(2–3), 250–265. https://doi.org/10.1177/0278364919859436

Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In D. Precup & Y. W. Teh (Eds.), Proceedings of the 34th

International Conference on Machine Learning (PMLR 70, pp. 1243–1252). https://proceedings.mlr.press/v70/gehring17a.html

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Hosla, S. (2025, April 2). CNN | Introduction to pooling layer. GeeksforGeeks. https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

Jeong, M.-H., Lee, T.-Y., Jeon, S.-B., & Youm, M. (2021). Highway Speed Prediction Using Gated Recurrent Unit Neural Networks. Applied Sciences, 11(7), 3059. https://doi.org/10.3390/app11073059

Lee, S., Song, B., & Shin, J. (2024). Collision Prediction in an Integrated Framework of Scenario-Based and Data-Driven Approaches. *IEEE Access*, *12*, 55234–55247. https://doi.org/10.1109/access.2024.3388099

Li, H., & Chen, L. (2025). Traffic accident risk prediction based on deep learning and spatiotemporal features of vehicle trajectories. PloS one, 20(5), e0320656. https://doi.org/10.1371/journal.pone.0320656

Li, L., & Zhang, X. (2025). Addressing data imbalance in collision risk prediction with active generative oversampling. *Scientific Reports*, *15*(1). https://doi.org/10.1038/s41598-025-93851-3

Lubbe, N., & Rosén, E. (2014). Pedestrian crossing situations: quantification of comfort boundaries to guide intervention timing. Accident; analysis and prevention, 71, 261–266. https://doi.org/10.1016/j.aap.2014.05.029

Manaswi, N. K. (2018). RNN and LSTM. In *Deep Learning with Applications Using Python* (pp. 115–126). Apress. https://doi.org/10.1007/978-1-4842-3516-4_9

Meng, D., Xiao, W., Zhang, L., Zhang, Z., & Liu, Z. (2023). *Vehicle Trajectory Prediction based Predictive Collision Risk Assessment for Autonomous Driving in Highway Scenarios*. arXiv. https://doi.org/10.48550/ARXIV.2304.05610

Niu, Z., Saad Saoud, L., & Hussain, I. (2024). MAD-CNN: High-sensitivity and robust collision detection for robots with variable stiffness actuation (Version 3) [Preprint]. arXiv. https://arxiv.org/abs/2310.02573

Rossi, L., Ajmar, A., Paolanti, M., & Pierdicca, R. (2021). Vehicle trajectory prediction and generation using LSTM models and GANs. *Plos One*, *16*(7), e0253868. https://doi.org/10.1371/journal.pone.0253868

Saravanarajan, V. S., Chen, R.-C., Dewi, C., Chen, L.-S., & Ganesan, L. (2023). Car crash detection using ensemble deep learning. *Multimedia Tools and Applications*, *83*(12), 36719–36737. https://doi.org/10.1007/s11042-023-15906-9

Saxena, A. (2023, January 17). *Introduction to Long Short-Term Memory (LSTM)* (Medium, Ed.). https://medium.com/analytics-vidhya/introduction-to-long-short-term-memory-lstm-a8052cd0d4cd

Shilpa, N., Veera Kishore, K., Anitha, S., Swapna Sathi, B. S., Sai Vijay, C., & Prabhkar, C. (2023). Data science using warning systems and vehicle crash detection. Industrial Engineering Journal, 52(1), 41–44. https://siiet.ac.in/wp-content/uploads/2023/12/75.Data-Science-Using-Warning-Systems-a

nd-Vehicle-Crash-Detection_compressed.pdf

Singh, V., Sahana, S.K. & Bhattacharjee, V. (2025). A novel CNN-GRU-LSTM based deep learning model for accurate traffic prediction. *Discov Computing 28, 38*. https://doi.org/10.1007/s10791-025-09526-0

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing &Amp; Management*, *45*(4), 427–437. https://doi.org/10.1016/j.ipm.2009.03.002

Song, X., Chen, K., Li, X., Sun, J., Hou, B., Cui, Y., Zhang, B., Xiong, G., & Wang, Z. (2021). Pedestrian Trajectory Prediction Based on Deep Convolutional LSTM Network. *IEEE Transactions on Intelligent Transportation Systems*, *22*(6), 3285–3302. https://doi.org/10.1109/tits.2020.2981118

T.J.J., R. (2020, September 2). *LSTMs Explained: A Complete, Technically Accurate, Conceptual Guide with Keras* (Medium, Ed.). https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2

WHO. (2025, May 19). *Targets of Sustainable Development Goal 3* (WHO, Ed.). https://www.who.int/europe/about-us/our-work/sustainable-development-goals/targets-of-sustainable-development-goal-3

Xie, G., Shangguan, A., Fei, R., Ji, W., Ma, W., & Hei, X. (2020). Motion trajectory prediction based on a CNN-LSTM sequential model. *Science China Information Sciences*, *63*(11). https://doi.org/10.1007/s11432-019-2761-y

S. Yang, X. Yu and Y. Zhou. (2020). LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example. *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, Shanghai, China, pp. 98-101. https://doi.org/10.1109/IWECAI50956.2020.00027

Zhang, Y., Zhu, Y., & Sun, J. (2023). A CNN-based ship–ship collision risk classification method using image recognition. Mathematics, 11(7), 1743. https://doi.org/10.3390/math11071743

Krzysztof Zarzycki, Maciej Ławryńczuk. (2022). Advanced predictive control for GRU and LSTM networks. *Information Sciences,Volume 616,Pages 229-254*.ISSN 0020-0255. https://doi.org/10.1016/j.ins.2022.10.078