ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Learning &
Adaptive Systems
Prof. Andreas Krause

Semester Thesis

Urban vehicle trajectory learning via transformers

Autumn Term 2022

Supervised by:
Dr. Kenan Zhang
Xia Li
Prof. Dr. Andreas Krause

Author:
Andrea Bionda

Declaration of Originality

I hereby declare that the written work I have submitted entitled

Urban vehicle trajectory learning via transformers

is original work which I alone have authored and which is written in my own words.[1]

Author(s)

Andrea	Bionda

Student supervisor(s)

Kenan Xia	Zhang Li

Supervising lecturer

Andreas	Krause

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Place and date	Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Contents

Preface

Welcome to the report on our semester project, *Urban Vehicle Trajectory Learning via Transformers*. The aim of this project was to adapt a deep learning network, originally designed for predicting pedestrian trajectories, to work with vehicle datasets and improve its performance through the incorporation of map information.
This report outlines the steps we took and the results we achieved in this process. We hope that our work will contribute to the development of more effective and reliable autonomous driving systems in the future.

Abstract

This project aims to improve the accuracy of **vehicle trajectory prediction** for autonomous driving by using transformer-based deep learning models and incorporating map information and the interaction between vehicles.
Specifically, we sought to extend the STAR network [1], previously used for pedestrian trajectory prediction, to work with vehicle datasets. The use of the pNEUMA dataset from EPFL[2], which includes various areas and times, allowed us to train and test our models.
The incorporation of map information and the consideration of the interaction between vehicles added complexity to the model, requiring it to not only predict the movement of a single vehicle, but also to consider how that movement may affect other vehicles on the road.
The results of this project have the potential to improve the efficiency and safety of autonomous driving systems by providing advance warning of potential collisions and optimizing routing decisions.

[2]pNEUMA website: https://open-traffic.epfl.ch/

Chapter 1

Related works

To establish a strong foundation of knowledge and generate ideas for the project, we analyzed several works in addition to STAR.
One of these works was Social LSTM[2], which presents a machine learning approach for predicting the future trajectories of people in crowded spaces using a Long Short-Term Memory (LSTM) network. This paper was useful for understanding the basics of trajectory prediction and the importance of social interactions in crowds, but it was focused on pedestrian trajectories rather than vehicle trajectories. (Figure 1.1)
Another framework analyzed was mmTransformer[3], which proposes a transformer-based model for predicting multiple plausible future trajectories of nearby vehicles in autonomous driving scenarios. The proposed method models the multimodality of the data at the feature level using stacked transformers and a region-based training strategy, and uses trajectory, map, and social information to improve the predictions layer by layer. This paper was relevant to the scope of the project as it was trained and tested on vehicle datasets. (Figure 1.2)
The project also looked at the work LatentFormer[4], which presents a transformer-based model for predicting the future trajectories of multiple agents. LatentFormer uses a hierarchical attention mechanism to model interactions among dynamic objects in the scene, taking into account both past observations and future states of the agents. The authors also propose a multi-resolution map encoding scheme that captures both local and global scene context to guide the generation of more feasible future trajectories. (Figure1.3) This paper inspired the inclusion of map information in the project's baseline model and will be further investigated.

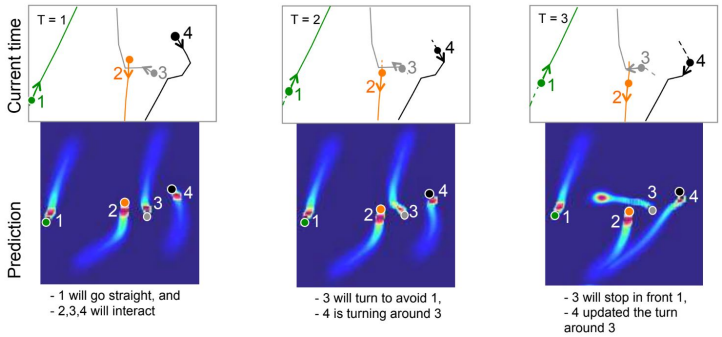

Figure 1.1: LSTM predictions: shows the effect of social interactions

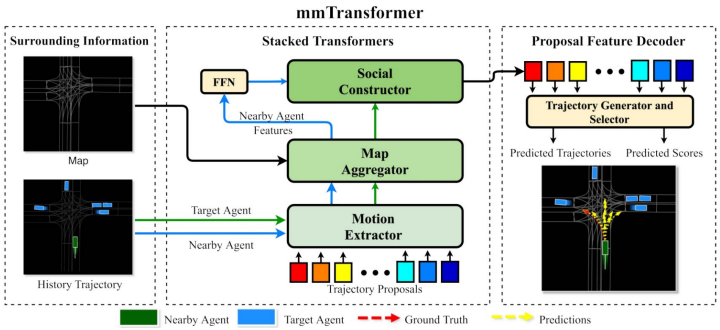

Figure 1.2: mmTransformer network structure

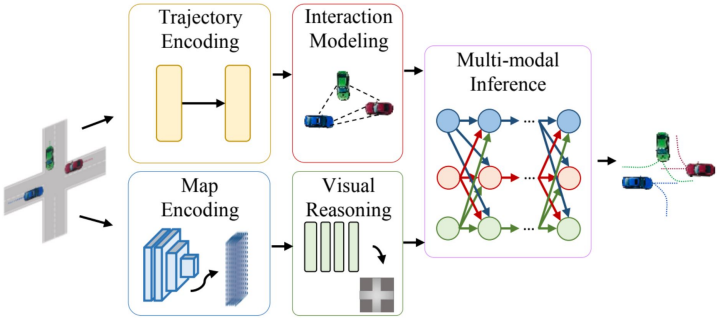

Figure 1.3: LatentFormer network structure

Chapter 2

Introduction to Original Components

2.1 Problem Statement

The problem can be formulated mathematically as follows: Given a sequence of observed positions of N vehicles $\{p_t^i\}_{i=1}^N$ for time steps $t = 1$ to T_{obs}, predict their future positions for time steps $T_{obs} + 1$ to T_{pred}. The positions of each vehicle are represented by their xy-coordinates (x_t^i, y_t^i) in a top-down view map. The task is to find a prediction model, represented as a function f, such that the predicted positions $\{p_t^i\}_{i=1}^N$ for time steps $T_{obs} + 1$ to T_{pred} are as close as possible to the true positions.

$p_t^i = (x_t^i, y_t^i)$ = position of i-th pedestrian at time t

T_{obs} : Observation Time

T_{pred} : Prediction Time

f: prediction model

In the project, we used two evaluation metrics to measure the performance of the trajectory prediction model: Average Displacement Error (ADE) and Final Displacement Error (FDE).

ADE measures the average error between the predicted positions and the actual positions of all pedestrians over all time steps. It is calculated as the mean square error (MSE) between the predicted positions and the actual positions for all pedestrians and all time steps. Mathematically, it can be represented as:

$\text{ADE} = \frac{1}{N \times (T_{pred} - (T_{obs}+1))} \sum_{i=1}^{N} \sum_{t=T_{obs}+1}^{T_{pred}} (x_t^i - \hat{x}_t^i)^2 + (y_t^i - \hat{y}_t^i)^2$

where (x_t^i, y_t^i) is the actual position of the i-th pedestrian at time t, and $(\hat{x}_t^i, \hat{y}_t^i)$ is the predicted position of the i-th pedestrian at time t.

FDE, on the other hand, measures the error between the predicted final position of each pedestrian and the actual final position. It is calculated as the Euclidean distance between the predicted final position and the actual final position for each pedestrian. Mathematically, it can be represented as:

$\text{FDE} = \sqrt{(x_{T_{pred}}^i - \hat{x}_{T_{pred}}^i)^2 + (y_{T_{pred}}^i - \hat{y}_{T_{pred}}^i)^2}$

where $(x_{T_{pred}}^i, y_{T_{pred}}^i)$ is the actual final position of the i-th pedestrian, and $(\hat{x}_{T_{pred}}^i, \hat{y}_{T_{pred}}^i)$ is the predicted final position of the i-th pedestrian.

Lower values for both ADE and FDE indicate better performance of the model.

2.2 STAR model

The Spatio-Temporal grAph tRansformer (STAR[1]) framework is a method for predicting the future trajectories of pedestrians in crowded spaces. The authors propose using Transformers, which are attention-based models, to provide an efficient solution to this task.
The pipeline for the STAR framework can be divided into two parts: data preprocessing and the network itself. In our project, we used the STAR setting as a baseline due to its simplicity.

2.2.1 Data preprocessing

For data preprocessing, the researchers used five pedestrian datasets from various sources, including ETH and UCY. They selected one file for validation and used the remaining files for training. They extracted the trajectory data from these files and created two dictionaries: one containing the pedestrians present in each frame, and the other with the full trajectory for each pedestrian.
They then shuffled all the frames in the datasets and created starting batches for each frame. To do this, they selected all pedestrians present at that frame, as well as those present at the end of the sequence length (which was set to 20 frames, with the first 8 for observation and the last 12 for prediction). For each present pedestrian, they created a trajectory fragment of 20 frames, and then selected only the valid trajectory fragments to create the starting small batch. They combined these small batches to ensure a balanced number of trajectory fragments in each batch and optionally rotated the trajectories for increased generalization.
Finally, they shifted the trajectories in each batch to the coordinates of the last frame in the observed sequence in order to have the predictions starting from the origin. Both the original and the shifted batch were kept, along with the shift value and other additional data, and stored in a pickle file for use by the network during training and testing.

2.2.2 Transformer Model Background

The Transformer[5] is a neural network architecture that has achieved great success in natural language processing (NLP) tasks such as machine translation, sentiment analysis, and text generation.
The architecture is based on the encoder-decoder structure, which is widely used in recurrent neural network (RNN) sequence-to-sequence models. The core idea behind the Transformer is to replace recurrence with a multi-head self-attention mechanism.
The self-attention mechanism of the Transformer learns query, key, and value matrices for all embeddings in a sequence. It then computes attention by taking the dot product of the query and key matrices, scaled by the square root of the dimension of the query, and applying a softmax function.
Mathematically, it can be represented as:

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

where Q, K, and V represent the query, key, and value matrices respectively, and d_k is the dimension of each query.
This allows the model to learn temporal dependencies over long time horizons, in contrast to RNNs which have limited memory. Additionally, by decoupling attention into query, key, and value tuples, the self-attention mechanism can capture more complex temporal dependencies.

The multi-head attention mechanism allows the model to combine multiple hypotheses when computing attention. It allows the model to jointly attend to information from different representations at different positions, with k heads.
Mathematically, it can be represented as:

$$MultiHead(Q, K, V) = f_O([head_i]_{i=1}^{k})$$

where $head_i = Att_i(Q, K, V)$, and f_O is a fully connected layer merging the output from k heads.
Finally, the Transformer outputs updated embeddings by a fully connected layer with two skip connections, and uses positional encoding to add positional information to the embeddings. (Figure 2.1)
However, current Transformer-based models are limited to non-structured data sequences, such as word sequences. STAR extends Transformers to more structured data sequences, such as graph sequences, and can be applied to trajectory prediction.

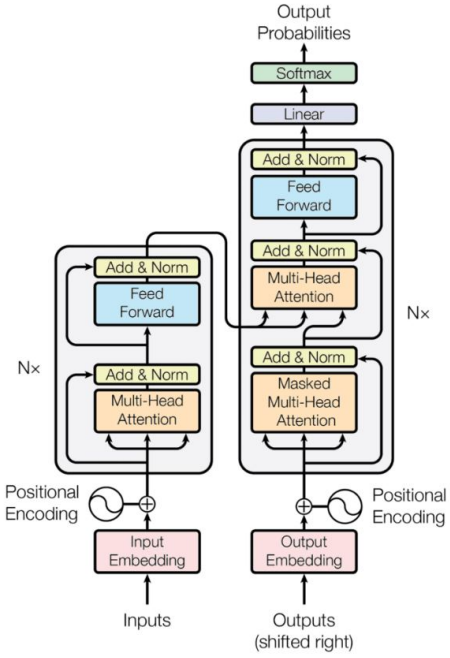

Figure 2.1: Transformer architecture structure

2.2.3 Network

This framework (Figure 2.2) uses temporal Transformers and spatial Transformers to capture the temporal dependencies and spatial interactions among pedestrians, respectively.
The temporal Transformers are applied independently to each pedestrian to capture their individual motion dynamics, while the spatial Transformers use a Transformer-based graph convolution mechanism called TGConv to better capture complex social interactions. TGConv is similar to other attention-based graph convolution methods, but uses the self-attention mechanism of Transformers. The authors of the STAR framework demonstrate that TGConv tends to perform particularly well on datasets with high pedestrian densities and complex interactions.

To extract spatio-temporal interactions, the STAR framework interleaves between the spatial and temporal Transformers, using a simple yet effective strategy. In order to address the challenge of modeling time series data with strong temporal consistency, the authors also introduce a read-writable graph memory module that continuously performs smoothing over the embeddings during prediction.
In the temporal encoding process, the shifted batch is used, while the original absolute trajectories are used in the spatial encoding process. For each frame of the 20-frame sequence in the input batch, the pedestrians fully present in the scene are identified. Then, only the frames up until the current one are considered. The ground truth is used for the first observed sequence of frames, while the predictions computed previously are used for the following frames. The trajectories are then shifted to the mean x-y values of the pedestrians present in the scene for input into the spatial encoding process.
The input batches are then passed through a small feedforward network and input into the temporal and spatial encoders. The structure of these encoders is similar to a simple transformer encoder, with the exception of the use of TGConv in the spatial encoding process, which requires an interaction graph mask with 0s for neighbors and -inf for all other pedestrians.
The embeddings from the two encoders are then concatenated and input into a sequence of one spatial and one temporal encoder. The output embedding is saved in the graph memory for future use in the temporal encoding process, and then concatenated with gaussian noise and used in the decoder, which is a simple linear layer to produce the x-y prediction.

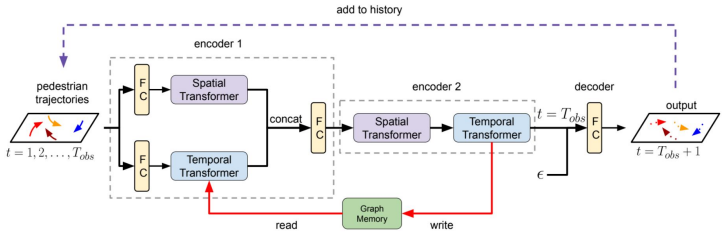

Figure 2.2: STAR network structure

2.3 pNEUMA

For our project, we aimed to apply the Spatio-Temporal grAph tRansformer (STAR) network to the pNEUMA vehicle datasets from EPFL. pNEUMA is a large-scale dataset of naturalistic vehicle trajectories collected in the congested downtown area of Athens, Greece using a swarm of 10 drones that hovered over the central business district for five days. The drones recorded traffic streams in a congested area of 1.3 km^2 with over 100 km of road network.
To access the pNEUMA dataset, we downloaded a specific CSV file from the website that contained the area and time of interest (Figure 2.3). We utilized functions from the github repository pNEUMA to process the raw data, including converting the CSV files to HDF5 format, adjusting trajectories by groups and standardizing timestamps, map-matching with the Athenian road network, and extracting data from the stored HDF5 files. These functions made it easier to work with the pNEUMA datasets and prepare them for use with the STAR network.

Figure 2.3: pNEUMA datasets

2.4 Map Encoding

For our project, we decided to incorporate the map encoding from the LatentFormer network model into our baseline model, the Spatio-Temporal Graph Transformer (STAR) framework. We chose to do this because we had access to the necessary map information as input data, and because we believed that incorporating the map information could improve the performance of the STAR network.
The map encoding scheme proposed by the authors of LatentFormer utilizes a vision transformer module to effectively capture both local and global scene context in order to guide the generation of more feasible future trajectories. To encode the scene context, the authors use a multi-resolution approach where local and global patches are extracted from the map. To address the location invariance bias of convolutional encoding methods, the authors utilize a vision transformer to focus on parts of the map representations that are relevant to each agent. The maps used in this work are represented as binary masks of drivable areas, and are augmented with the absolute pixel indices and the Euclidean distances between each pixel and the center of the map in three separate channels. (Figure 2.5)
To process the map, the authors use three convolutional layers with 8, 16, and 6 filters with sizes of 3×3, 3×3, and 1×1 and strides of 1, 2, and 1, respectively. The final output of these layers has a dimension of $32 \times 32 \times 6$. The map is used at two different levels in the model, including a global representation that

captures the entire map and local representations based on overlapping patches extracted from the map. The concatenation of these two representations is fed into a position embedding layer followed by a vision transformer to generate the final map encoding features. The model uses the map information while decoding the previously computed sequence of states of the vehicles. (Figure 2.4)

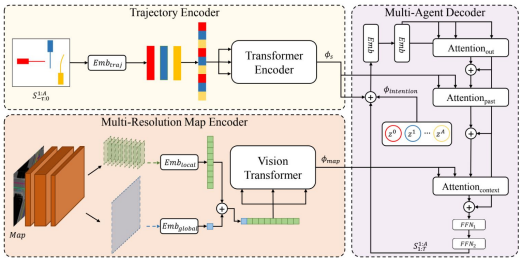

Figure 2.4: LatentFormer network structure

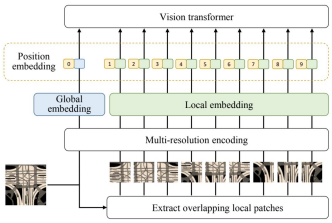

Figure 2.5: Map encoding explanation

Chapter 3

Project Implementation

3.1 Adapting STAR for the pNEUMA Dataset

In this project, our goal was to incorporate the vehicle datasets from the pNEUMA package into the STAR framework. However, we faced a number of challenges during this process due to differences in data storage and other issues that arose. To address these challenges and streamline the data preprocessing before training, we created a new Python notebook called *dataprocess.ipynb*.

3.1.1 Datasets and CSV files

One significant difference we encountered was the size of the vehicle datasets, which were much larger than the pedestrian datasets used in the past. This made it impractical to use multiple datasets together, so we had to select a single dataset for training and generate two additional datasets for validation and testing.
To do this, we wrote custom functions that took the original CSV file as input and created new CSV files, which were saved in a separate folder. One function was specifically designed for debugging purposes and generated a small dataset, while another function could be used to create a validation/test dataset or divide a single original dataset into multiple parts. The third function removed motorcycle data, as their trajectories can sometimes be misleading.
These functions provided a convenient and flexible way to preprocess the data, as it was possible to customize their use by modifying variables at the start of the script. This allowed us to choose whether to use the functions or not, specify the number of samples desired, and select the original CSV file to be used.

3.1.2 Issues and adaptations

During our efforts to integrate the vehicle datasets into the STAR framework, we encountered several issues that required modification to the existing codebase. One key issue was the need to modify the way dictionaries with vehicle trajectories were computed, specifically by modifying the *traject_preprocess* function in the *Trajectory_Dataloader* class. This was necessary because the data in the vehicle datasets was stored differently than the pedestrian datasets, requiring the use of pNEUMA functions for processing and map matching before it could be loaded directly from the CSV file.
Additionally, we found that the list of vehicle IDs provided by the processing function did not match the list of trajectories. To resolve this, we had to retrieve the number of vehicles directly from the dimensions of the dataframes.
Next, we had to extract the relevant dataset from the HDF5 file and obtain a list of

dataframes containing the trajectories of each vehicle. We then selected the correct columns containing timestamps and x and y coordinates, and converted the timestamps from milliseconds to seconds. To avoid potential errors caused by vehicles starting at non-round seconds, we also rounded these values as some vehicles were shifted by a few milliseconds due to processing in the pNEUMA package functions. Another issue we encountered was the presence of missing frames in some trajectories, which caused errors when trying to extract trajectory fragments using the *get_seq_from_index_balance* function. To solve this, we decided to skip these vehicles as they were a small percentage of the overall dataset.
Given the unique nature of vehicle scenes compared to pedestrian scenes, we initially considered disabling batch balancing for the vehicle datasets. However, we found that creating a separate batch for each frame in the dataset resulted in an excessively large number of batches that the network was unable to process. As a result, we decided to continue using batch balancing, but modified the number of trajectories to be included in each batch.
Through these modifications, we were able to successfully generate the train and test batches of vehicle trajectories needed for use in the STAR model.

3.2 Visualizing Training Progress and Test Results

To effectively monitor the performance of our model during training and testing, we decided to use wandb.com to plot our results. This provided a convenient and intuitive way to visualize the progress of the model. Additionally, we added a command line option that allows users to disable the use of wandb and instead rely on the pre-existing method of printing results. This option is useful for those who do not have an account or prefer not to log in to wandb.
To implement these features, we modified the *processor.py* file. During the training phase, we plotted the values of various metrics at each epoch, including the training loss and the average displacement error (ADE) and final displacement error (FDE) on the validation set. These plots were updated in real-time, allowing us to quickly identify if the model was overfitting. The wandb platform also offered post-processing options such as the ability to remove outliers, smooth the data, and apply normalization formulas.
During the test phase, we plotted the first element of each batch, with different colors and labels for the observation and prediction frames of the ground truth trajectory and the actual network prediction. Using command line options, it was also possible to plot the absolute coordinates instead of the normalized ones and to include the trajectories of neighboring vehicles. The results of these plots were saved in the *src/Results* folder, even if the wandb option was disabled.

3.3 Implementation of Map Encoding

The inclusion of map information in our pipeline involved two stages. In the first stage, we extracted features from the map itself using a vision transformer. In the second stage, we added the map features to the existing features using another transformer, as implemented in the LatentFormer.

3.3.1 Feature map computation

To create the three input channels for the map, we obtained a boolean representation of the drivable areas for the entire pNEUMA map and the longitude and latitude coordinates for every pixel. We then needed to crop the map to only include the region covered by our dataset. To do this, we created a function in the

dataprocess.ipynb file that kept only the data for pixels within the minimum and maximum longitude and latitude values of the dataset. Additionally, we added an option to extend the map by a certain amount for numerical reasons related to later convolution operations. The resulting cropped map was then saved as a new CSV file.
We created two new files, *map_encoder.py* and *map_files.py*, to contain the code needed to incorporate the map information into the STAR network. In the *MapEncoder* class, we first created the three channels by reshaping the boolean map, computing the distance between each pixel and the center, and generating indices for each pixel. The resulting arrays were saved in the *output/eth/map_arrays* folder to avoid recalculating them each time.
Then, we created a feature map using the three input channels and a series of linear layers. We modified the original parameters to accommodate the larger size of our map, increasing the kernel size in the first layer from 3 to 5 and in the second layer from 3 to 4. In the second layer, we also used a stride of 3 instead of 2.

3.3.2 Vision Transformer implementation

The resulting feature map was then used as input to a vision transformer, which we modified from the implementation available at ViT repo to suit our needs. Specifically, we removed the class token and mlp head used for image classification and instead used the global map feature as input. We also increased the number of input channels to 6 and modified the dimensions of the input map and patches. To make the map dimensions easier to work with, we slightly augmented the original map to create a square map, as was done in the original paper.
Next, we used the features computed by the vision transformer as the source in the encoder of a new transformer module and the previously computed features as the target in the decoder. We modified the implementation from Towards Data Science by removing the final linear layer, as we were only interested in the features.
Finally, we used the resulting features in the original STAR decoder to generate trajectory predictions.

Chapter 4

Results and Analysis

4.1 Experiment Settings

To evaluate the performance of our model, we conducted a series of experiments using different settings. We first evaluated the original STAR file with pedestrian datasets, using ETH as the test set. Then, we evaluated the network on the original settings using vehicle datasets. Next, we used absolute rather than normalized trajectories for the spatial encoding and added map encoding to this setting. We chose to use absolute trajectories with spatial encoding because we believed it would work well in conjunction with map encoding.
For all experiments, we trained for a total of 400 epochs. For the experiments involving vehicle datasets, we used the location 1, date 24/10/2018, with the time period of 9:00 – 9:30 for the training dataset, 10:00 – 10:30 for the validation set, and 8:30 – 9:00 for the test set.
We used sequence lengths of 20 (8 frames for observation and 12 for prediction) for pedestrian datasets and 10 (4 frames for observation and 6 for prediction) for vehicle datasets, as these have been shown to be standard for testing with vehicles in other studies. Additionally, we increased the distance in every direction within which a vehicle is considered a neighbor from 10 to 100, as the larger scale of road driving scenarios requires considering a wider range of distances.

4.2 Quantitative Analysis

Table 4.1: Training and testing results for different models

Model	Best epoch	Train ADE	Train FDE	Test ADE	Test FDE
Pedestrians	201	0.355	0.642	0.358	0.637
Vehicles baseline	10	2.461	5.212	2.752	5.872
Spatial absol traj	42	4.121	0.642	2.266	4.850
Map encoding	215	9.107	15.650	11.870	20.841

From these results, we can see that the test error for the vehicle dataset baseline experiment is higher than for the pedestrian dataset. This is likely due to the fact that the vehicle dataset involves longer distances, as vehicles typically move at a higher speed.
We also observe that using absolute trajectories for the spatial encoding resulted

in better performance. This could be because it was able to train for more epochs before reaching the best test errors.
Additionally, we notice a generalization error in all vehicle experiments, although it is relatively small.

4.3 Qualitative Analysis

In this project, we used wandb.com to plot the training metrics of our four models in real-time. Figures 4.1, 4.2, 4.3 and 4.4 show the training loss, validation average displacement error (ADE), and validation final displacement error (FDE) for every epoch. To improve the clarity of the graphs, we applied small Gaussian smoothing and removed outliers from the training loss using wandb's post-processing features.

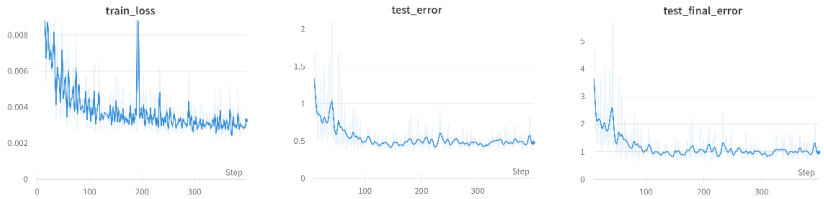

Figure 4.1: Pedestrians model results

Figure 4.2: Vehicles baseline model results

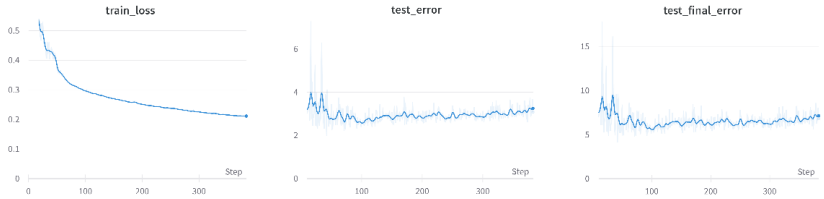

Figure 4.3: Spatial abs. traj. model results

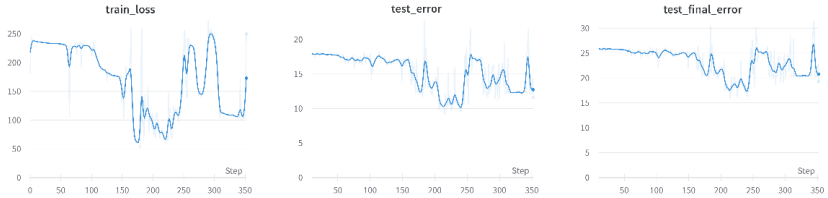

Figure 4.4: Map encoding model results

In the following figures, we present some results of the original STAR model applied to pedestrian datasets.
The plots show the first 8 frames of the observed trajectory in blue, and the ground truth trajectory in orange and our prediction in green for the subsequent 12 frames.
As shown in Figure 4.5, the model's prediction often follows the general direction of the ground truth, but may not precisely match the exact trajectory.
On the other hand, Figure 4.6 illustrates a prediction that goes in the opposite direction of the ground truth.
In Figure 4.7, we also include the trajectory of pedestrian 0 and the trajectories of the pedestrians considered in its neighborhood, which contribute to the model's social interaction component.
Overall, these examples demonstrate the model's ability to capture the general movement patterns of pedestrians, but also highlight some of the limitations and challenges in accurately predicting their precise trajectories.

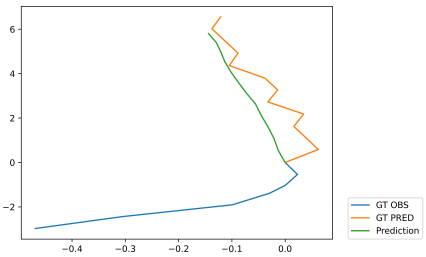

Figure 4.5: Example of good prediction

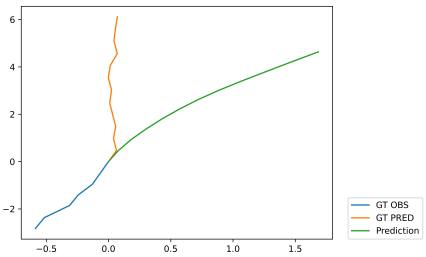

Figure 4.6: Example of bad prediction

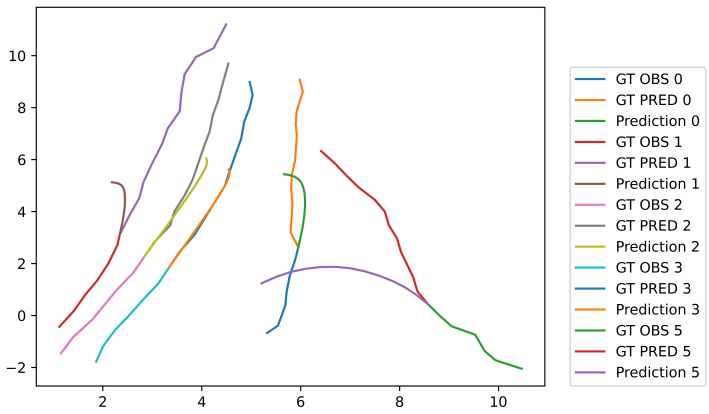

Figure 4.7: Plot of neighbors pedestrians

In the following figures we show some examples of plotted trajectories of the baseline STAR model with the pNEUMA vehicles datasets.
As for the pedestrians datasets, we can see in Figure 4.8 a good prediction, but we can also observe that the trajectory is more regular in this case. This regularity allows the model to make a more precise prediction.
In Figure 4.9, we can see that the model doesn't predict correctly the change of direction.
In Figure 4.10, we see that there is no ground truth plotted. This is because the car is stopped in every frame of the sequence, and therefore all the points are at coordinates (0, 0). The model is not able to predict this.
In figure 4.11, we plot an example of neighboring vehicles.

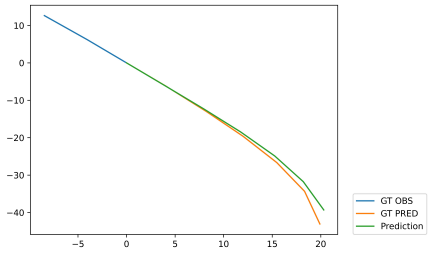

Figure 4.8: Example of good prediction

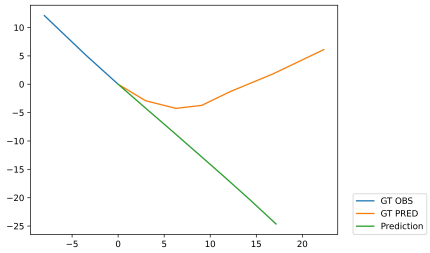

Figure 4.9: Example of bad prediction

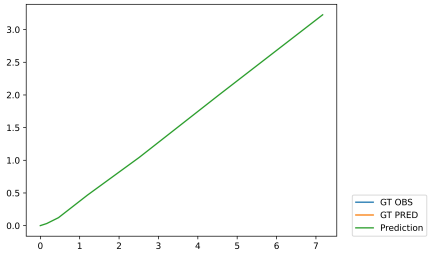

Figure 4.10: Example of stopped vehicle

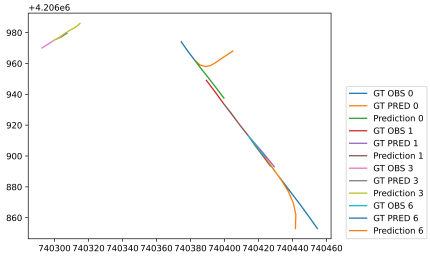

Figure 4.11: Plot of neighbors vehicles

4.4 Conclusions and Future Work

Upon analyzing the results of our experiments, we made several observations.
Firstly, it appears that the model tends to overfit quickly, with the best results being achieved in the early stages of training (particularly in the baseline case).
Secondly, the presence of stopped vehicles in the dataset significantly impacted the model's performance. This is likely due to the fact that vehicles may be stopped at stop signs, traffic lights, or due to traffic, whereas pedestrians do not exhibit this behavior.
Finally, the map encoding did not perform as expected, possibly due to the fact that the features generated by the temporal encoder (which are combined with the map features in the transformer) are based on shifted trajectories, leading to shifted predictions after decoding. This may cause errors when the attention mechanism is applied to the map, which has absolute coordinates. We also attempted to use absolute coordinates with the temporal encoding, but this made training with the STAR model impossible.
However, we believe that further investigation into these issues could lead to improvements in model performance and will consider them for future work.

Bibliography

[1] C. Yu, X. Ma, J. Ren, H. Zhao, and S. Yi, "Spatio-temporal graph transformer networks for pedestrian trajectory prediction," 2020.

[2] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 961–971.

[3] Y. Liu, J. Zhang, L. Fang, Q. Jiang, and B. Zhou, "Multimodal motion prediction with stacked transformers," 2021.

[4] E. Amirloo, A. Rasouli, P. Lakner, M. Rohani, and J. Luo, "Latentformer: Multi-agent transformer-based interaction modeling and trajectory prediction," 2022.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.