CIMAT Centro de Investigación en Matemáticas, A.C.

TRAJECTORY FORECASTING FOR AUTONOMOUS VEHICLES

T E S I S

Que para obtener el grado de

Maestro en Ciencias

con Especialidad en

Computación y Matemáticas Industriales

Presenta:

Juan Luis Baldelomar Cabrera

Director de Tesis:

Jean-Bernard Hayet

[illegible]

Autorización de la versión final

Guanajuato, Gto., 22 de febrero de 2023

Agradecimientos

Al Dr. Jean-Bernard Hayet, por haberme apoyado y asesorado a lo largo de este proyecto de investigación. Agradezco tus consejos antes de iniciar mi proyecto de investigación sobre como dedicir en que orientar mi área de especialización. Gracias también por la paciencia y guía a lo largo de este trayecto que me ha mostrado una pequeña parte de lo que es el mundo de la investigación.

A la Dra. Claudia Esteves, por haberme apoyado y asesorado en mi año de formación en DEMAT previo a ingresar a la maestría. Agradezco tus consejos y el apoyo mientras me preparaba para poder ingresar a CIMAT.

A mi padre, Jose Luis Baldelomar Rivera, y a mi madre, Claudia María Cabrera Rosito, por haberme dado su apoyo incondicional desde el momento en que decidí venir a estudiar a México. Siempre estaré agradecido con ustedes, y este logro es un reflejo del esfuerzo y apoyo de ustedes hacia mí. Me han apoyado y acompañado en todo momento.

A mi tía, Alba Elena Baldelomar Rivera, por su apoyo incondicional durante toda mi vida y también durante este trayecto. Al igual que con mis padres, este logro es un reflejo del apoyo que he recibido en todo momento por parte de ti.

A mi abuelo, José Luis Baldelomar Ramírez, por haberme guiado y enseñado a luchar por lo que deseo, y haberme enseñado a proyectar mi vida a futuro para saber que debo hacer para alcanzar lo que quiero. Has sido más que mi abuelo, como otro padre y un amigo para mí.

A mi abuela, Rosalba Rivera, porque sé que a pesar de la distancia, estoy siempre en su mente y corazón y pide constantemente por mí.

A mi hermano, Juan José Baldelomar Cabrera, por haberme guiado para poder entrar a CIMAT y haberme ayudado y apoyado en todas mis dudas académicas cuando lo necesitaba. Gracias por el apoyo.

Al Dr. Adrián Pastor López y al Dr. Mariano Rivera, por formar parte del comité de evaluadores de este proyecto y darnos un poco de su tiempo para poder retroalimentarnos sobre el trabajo realizado. Además, agradezco a ambos que fueron mis profesores en cursos del área de Aprendizaje Máquina y Procesamiento de Lenguaje Natural, y ambos cursos le dieron un giro y un impulso a mi desarrollo profesional.

Al Centro de Investigación en Matemáticas, A.C (CIMAT), por haberme dado la oportunidad de pertenecer a esta honorable institución y haberme abierto las puertas. Estudiar aquí me dio la oportunidad de realizarme académicamente en uno de las áreas que me apasionaba desde que ingresé a mi licenciatura.

Finalmente, al Consejo Nacional de Ciencia y Tecnología (CONACyT), por haberme apoyado econó- micamente a través de su programa de becarios lo cual me permitió realizar mis estudios aquí en México.

Abstract

The following work is focused on the area of Trajectory Forecasting for autonomous driving vehicles. This problem has been tackled from different perspectives and for different contexts. Here, we focus on the prediction of trajectories for driving agents. The problem has two main dimensions that are relevant to accurately model it and obtain good results: the temporal dimension and the spatial dimension. Sequence-to-sequence models have been massively used to model the temporal dimension of the problem as we can see with models based on LSTM networks. In more recent approaches, Transformers attention mechanisms [13] have been explored, after the incredible results they achieved on the NLP field. To model the spatial relationships among agents, graph neural networks have been proposed in several works. However, mechanisms like the one from the Transformer have not been widely explored to also model the spatial dimension of the problem, and we carried out that task as the main contribution of our work.

We propose a model based on the Transformer architecture to tackle both, the temporal and spatial dimensions of the problem. We use two Transformer Encoders at two dimensions of the problem, the spatial and the temporal dimensions. The model receives as input a scene with all the neighbors present in it at specific time steps and outputs the prediction of all the trajectories for each agent in the scene, which means we are doing the joint prediction of all the agents rather than predicting the trajectory for each agent by itself. This allows us to take into consideration spatial relations in the sequence. The model works in the following way. The first Transformer along with a handcrafted CNN modules are used to extract spatial features. In this case the input is constructed in a way that we expect the first encoder to process spatial relations between the agents present in a scene. Those spatial features are then used as input for what we call a Temporal Transformer, because it works at the temporal dimension of the problem. This is achieved by doing a transposition of the temporal and spatial dimension of the output of the first encoder. The decoder then receives the output of the second encoder as a traditional Transformer model. The model is trained in an auto-regressive manner as the AgentFormer model [15] because it showed significant improvements over the more classical Teacher Forcing approach.

We worked with two datasets to train and test the model. NuScenes is an autonomous driving dataset commonly used for this task. However, for such complex models as Transformers, more data is required. Therefore, we did a little data augmentation process on this dataset by performing random rotations to the original inputs, and as a result we obtained better metrics for this dataset. The other dataset used in this work is Shifts. Shifts is a

dataset used for the uncertainty estimation problem for several tasks, among which we have trajectory forecasting for autonomous driving systems. This dataset contains much more data, and it proved useful to carry out the main training and testing of the models. We performed ablation studies to assess how much each part of the model contributed to the final result.

Contents

List of Figures

List of Tables

Acronyms

ADE	Average Displacement Error. 16, 58
CNN	Convolutional Neural Network. 8, 48, 55, 56, 58, 59
CVAE	Conditional Variational Auto Encoder. 12, 13
FDE	Final Displacement Error. 16, 58
GAN	Generative Adversarial Network. 13
GRU	Gated Recurrent Unit. 13, 15, 37, 39, 44
LSTM	Long Short Term Memory. 12, 13, 15, 37, 39, 44
NLP	Natural Language Processing. 12–15, 33, 37, 51, 52
RNN	Recurrent Neural Network. 12, 13, 15, 17, 35, 37, 39, 44
VAE	Variational Auto Encoder. 13

Chapter 1

Introduction

Human error is one of the factors that contributes the most to accidents while driving. Several reasons could be enumerated that cause this problem, but one of them is undeniably the lack of ability from humans to process big amounts of information for this specific task. To begin with, human vision is limited and it is at the same time the main source of information we use while driving. Added to this is the fact that humans, contrary to popular belief, are not good at multitasking, which leads some of the time to the aforementioned problem. Therefore, a lot of efforts have been invested recently into removing the human factor from the driving task.

Making car driving a task for machines is not a simple thing to do. First, cars need sensors to perceive the real world even better that how humans do. Once the information is available, signal processing algorithms need to be applied to filter the data that represent reality from noise. It is also needed to give an interpretation of these data in high level representations. Then, motion planning algorithms need to be implemented so that the self-driving vehicle can make decisions as a human being would do (or even better). Among the data and algorithms that the motion planner uses to make its decision, lays the field in which this thesis project aims to contribute: **Trajectory Forecasting**.

The Trajectory Forecasting field has a clear objective: To carry out the predictions of where agents of interest are going to be in the near future. For autonomous driving systems, trajectory forecasting aims to predict where agents surrounding the autonomous vehicles (e.g., other vehicles or pedestrians) will be in the near future to take that information into account and push the motion planner module to make better decisions. With the rising popularity and improvement of the performance of deep learning algorithms, a lot of deep learning models have been proposed to tackle this prediction task. These previous works have served as an inspiration for the work developed in this project, the **ST-Transformer**, which explores more in depth the use of Transformer neural networks along the main two dimensions of the problem: the spatial and the temporal dimensions. The name of the model refers to these two dimensions (S from spatial and T from temporal).

1.1 Related Work

In the following, we review some related works in the field of trajectory prediction. Note that we do not pretend to be exhaustive, given the huge volume of literature in this area.

1.1.1 Deterministic and Generative Models

The prediction of trajectories for agents of interest is a problem that concerns more than just the autonomous driving industry. Robots, as a general concept, are getting more and more involved in human activities and this leads to the need of predicting the motion of the agents that surround them to avoid collisions and accidents.

The problem has been modeled from different perspectives, from deterministic approaches to generative and probabilistic models. The deterministic models usually fall in the range of **deterministic regressors**, which use the inputs of the model to produce an output that indicates the future position of the agent. One of the earliest deterministic models was named Social Forces [19] and it used a physical approach to model the motion of persons using the concept of "social force". This model takes into account the agent's goal and the policies to avoid other agents and obstacles (represented as forces) to predict the trajectory.

With the computation power achieved in the last years and with Internet and data availability, deep learning models have reached incredible results and have soon become the state of art. CoverNet [2] frames the trajectory prediction problem as a classification problem over a diverse set of trajectories and make use of a convolutional module. More recent models represent the trajectories as sequences and work with them within the time series framework. Recurrent Neural Networks (RNN) models like Social LSTM [22] and Trajectron++ [20] use LTMS cells to incorporate the time variable of the trajectories and produce their outputs. Even more, both approaches incorporate a probabilistic approach, outputting a *distribution of trajectories* instead of a deterministic output. The work in [11] uses convolutional modules to process videos and get information about the persons behaviours and the persons interactions. Then it feeds those visual features to a LSTM module and predicts the trajectory. It also predicts the activity that the person is carrying out, framed as a classification problem from a predefined set of activities.

Other models that have gained popularity for the high quality of the results they obtained have been the Transformers. These models have emerged in the natural language processing (NLP) field, but they have soon permeated to the trajectory forecasting area. Giuliari has proposed using these architectures for the trajectory forecasting task in [13], using the observed past trajectory to encode information and then to decode the target trajectory. He has also proposed to cast the forecasting problem as a classification approach (ny) and confirmed that regression models outperform classification models. The AgentFormer [15] involves temporal and spatial information with a conditional variational auto-encoder (CVAE) module to output a distribution and account for multimodality.

1.1.2 Social Interactions

So far, we have discussed models that have addressed the problem from the temporal dimension. Trajectory forecasting, unlike other problems involving sequences like NLP tasks, depends heavily on spatial information as well as the temporal dimension. When driving, drivers incorporate spatial information about what other agents around them are doing, to figure out what they should do next (e.g. to avoid collision). Therefore, modeling the problem just from the temporal dimension may omit some relevant and valuable information.

Social LSTM [22] models the social context with what they call a Social Pooling layer, which consists on using different LSTM networks for each trajectory in a scene, and then using a layer (Social Pooling) in which these LSTM models share information about close neighbors, hoping that this layer captures spatial relationships between the agents. Social-GAN [3] uses a similar approach, encoding information from LSTM cells and then passing that information through a pooling layer, which then feeds (just once) a decoder to generate the trajectories. They also use a GAN architecture using the discriminator from the GAN to tell which trajectories are socially acceptable or not. Sophie [23] and Social-Ways [21] work with a similar architecture incorporating GANs, with the discriminator working at the decoder and both using an attention module between the encoder and the decoder.

Some substantially different types of work have been done in this area as well. The work in [1] addresses the social attention problem using a spatio-temporal graph, where the nodes of this graph represent the persons in a crowd of people, the spatial edges connect two different humans at the same time step, and temporal edges connect the same human at adjacent time steps. The Trajectron [5] and Trajectron++ [20] also use a spatio-temporal graph to model the relations between agents, altogether with LSTM cells to encode the past information and a CVAE generative model to get a distribution of the trajectories. PECNet [12] infers distant trajectory endpoints through a VAE and then conditions the trajectories on these endpoints by using a novel social pooling layer. The aim of conditioning the trajectory on the inferred endpoint it to obtain socially compliant trajectories. Bitrap [9] uses a similar approach, with a CVAE for inferring the endpoint and a GRU cell to encode information of the past trajectories. Then the endpoint predicted is passed as an input to the decoder, which uses a bidirectional RNN to generate the trajectories. SGNet [7] is a novel model that estimates the agent's goals at multiple temporal scales and feeds those goals as an input to a decoder to take into account several objectives, arguing, based on psychology and cognitive science, that people base their actions not on a single long-term goal, but a series of goals at different time scales.

More recent approaches involve Transformer architectures. The AgentFormer [15] involves spatial and neighbors information to account for the social interactions between them by putting all the agents belonging to the same scene at the same time step as an input (instead of each input representing each agent) and performing the join prediction for all the agents. The work in [24] is and end-to-end model that receives as inputs the clouds of lidar points and models the interaction between agents with a spatial transformer. Then it produces the target sequence by incorporating a recurrent approach, but it is important to highlight that the Transformer used in this model does not work along the temporal dimension.

1.2 Background

Trajectory forecasting can be framed as a sequence-to-sequence problem. Sequence-to-sequence problems are time series problems that consist on predicting a target sequence based on an observed source sequence. Suppose that we have a source sequence $S = \{S_1, S_2, \cdots, S_n\}$ and a target sequence $T = \{T_1, T_2, \cdots, T_m\}$ where each S_i and T_j are states that encode the needed information for a specific problem. We look for a model $\mathcal{F}$ that is capable of predicting the target sequence as precisely as possible. Hence, the problem can be formulated as minimizing a sum of loss functions $\mathcal{L}(T, \hat{T})$ where $\hat{T}$ are the sequences predicted by our model, $\mathcal{F}$,

$$\hat{T} = \mathcal{F}(S) = \{\hat{T}_1, \hat{T}_2, \cdots, \hat{T}_m\}.$$

Sequence-to-sequence (usually abbreviated as seq2seq in the literature) has its background in the Natural Language Processing (NLP) field. Several problems from the area of NLP have a sequence-to-sequence nature. Machine Translation, Question Answering and Text Summarization are all examples of problems that consist on using a source sequence and producing a target sequence based on that source sequence. For example, in the Machine Translation task, that consists on generating a translation for a text in some language to some other language, the source sequence is the text (sequence of words) in the original language, and the target sequence is the translation of this text to other desired language (i.e., another sequence of words). Therefore, several models and techniques used in machine learning for seq2seq problems also arose in the NLP area.

With the computation power achieved nowadays, Deep Learning models are the state of art for most of the Machine Learning problems. Transformer architecture models [25] are not the exception. These models emerged in the NLP area and are categorized as attention models; they became popular among the scientific community for the results they achieved for several NLP tasks. Transformer models have been around since 2017 and are the main research field for several deep learning concerning tasks.

In Chapter 3, the Transformer Architecture is discussed. Some concepts regarding the basis of these models will be succinctly explained, given the fact that the text would become unnecessarily long otherwise. If the reader does not feel comfortable with the following concepts, a quick review of them would make that chapter easier to understand. Those concepts are:

1. **Embeddings**. General representations of words emerged in the field of NLP. The idea is to have a global, vectorial representation for words that could be used and transferred from task to task. One of the most influential work related to this topic is

the paper from Tomas Mikolov, **“Efficient Estimation of Word Representations in Vector Space”** [8].

2. **Seq2seq models with RNNs.** Transformers are definitely not the most natural way to process a sequence with deep learning models. Recurrent Neural Networks (such as LSTMs or GRUs) are intuitive models that process a sequence element by element and incorporate the time factor in an inherent way to the architecture. **“Sequence to Sequence Learning with Neural Networks”** [16] proposed a novel approach by the time it was published as how to tackle these sequence-to-sequence problems, considering that the source and target sequences might not have the same length.

3. **Attention Mechanisms**. Sequence-to-sequence models usually process the source sequence through an encoding phase (encoder) and then produce the target sequence through a decoding phase (decoder), using a context vector retrieved from the encoder. The context vector acts as a summary of the source sequence, therefore limiting the information it can provide to the decoder. Attention mechanisms allow the decoder to retrieve more information, encoded from the source sequence, and to be able to determine which parts of that information are more valuable for the decoding phase. **“Neural Machine Translation by Jointly Learning to Align and Translate”** [4] illustrates this concept.

1.2.1 Trajectory Forecasting

So far, we have discussed seq2seq problems in the context of NLP, due to the impact this field has had on the development of models to tackle those problems. The trajectory forecasting area has some components that can be analogous to some sequence-to-sequence tasks in NLP, but there are other components that are quite different.

In the trajectory forecasting field, a simple formulation of the problem is to represent each source and target state as the position $(p_x, p_y)_t$ of the agent at a specific moment in time t and in a fixed 2D coordinate system. It is important to note that, for this specific problem, the target sequence has to have the form $T = \{T_{n+1}, T_{n+2}, \cdots, T_{n+m}\}$ because the points predicted belong to the future and go right after the last n observed points in the past. Then, the past trajectories of all the agents are used as the observed sequences and the objective is to predict the future trajectories that indicate where each agent in the scene is going to be in the future.

More complex models can emerge when taking into account more information than just the position of the agent along the trajectory. It is easy to realize that providing the velocity of the agent, its acceleration and its angle of rotation could form part of the state of each agent at all the time steps. And more complex information can be used as well. For example, bitmaps indicating the area in which the car can drive can be given as inputs as well, hoping that the models learn which is the drivable area, which are the lanes etc.

Once a model to tackle the problem exists, we need a way to evaluate how good or bad the model is performing. Given the nature of the problem, it results intuitive to use the distance between each point in the predicted trajectory and the real future trajectory as a metric to assess how good the model performs. The ideal scenario is when the distance from all the points in the predicted trajectory are the same as their respective points in the real trajectory. This is of course impossible due to the uncertainty and the nature of the problem, but a good model might predict in a fairly reasonable horizon of time a trajectory with an acceptable error. For this, we use the Average Displacement Error (ADE) metric. Noting that $|T| = m$, the ADE can be defined as follows

$$ADE(X_{pred}, X_{real}) = \frac{1}{m}\sum_{t=1}^{m} ||p_t^{pred} - p_t^{real}||_2, \tag{1.1}$$

where X_{pred} and X_{real} are defined as

$$\begin{aligned} X_{pred} =& \{p_t^{pred} \mid p_t^{pred} \in \mathbb{R}^2 \text{ and } t = 1, \cdots, m\}, \\ X_{real} =& \{p_t^{real} \mid p_t^{real} \in \mathbb{R}^2 \text{ and } t = 1, \cdots, m\}, \end{aligned}$$

and they represent the set of points that constitute the predicted and real trajectories, respectively. Note that $|| \cdot ||_2$ in equation 1.1 is basically the Euclidean distance formula in $\mathbb{R}^2$.

The ADE basically tells us how accurately the model predicted the trajectory, on average. A metric that tells how the model performs on the long term from the time perspective could be useful. For this, we can use the Final Displacement Error (FDE) which represents how far is the last point in the prediction horizon predicted by the model to the actual end point in the real trajectory. The FDE then looks like

$$FDE(X_{pred}, X_{real}) = ||p_m^{pred} - p_m^{real}||_2. \tag{1.2}$$

Finally, to evaluate models in this area, several datasets are usually used as benchmarks to assess a model's performance. Those datasets usually fall into one of two categories:

1. Datasets in which the data is obtained by a fixed camera and the trajectories are projected into a fixed coordinate system, like the ETH-UCY dataset.
2. Datasets in which the data is obtained by ego-vehicles equipped with several cameras, radar and lidar sensors and the trajectories are projected into a fixed coordinate system, but also contain the position of the agents in relation to the ego-vehicle that retrieved the data, like NuScenes dataset.

1.3 Objectives

Transformer models have made a huge impact on tasks that require attention mechanisms while processing sequences. The first work to frame this in the trajectory forecasting area is described in [13]. As we have stated before, the problem of predicting where an agent is going to be in the future involves both temporal and spatial variables. Based on this, we have the following objectives:

General Objective: To design and explore attention models at the spatial and temporal dimension of the problem and perform the joint prediction of the trajectories of the agents present in a scene to take into account how the actions of several agents in it influence the actions of the others.

Specific Objectives

- To propose and design a model that takes advantage of Transformer networks for doing the joint prediction of several agents in a scene that are close to an ego-vehicle.
- To consider spatial and temporal information to make the predictions.
- To explore a new model, named the ST-Transformer, to assess how viable it is to keep working with these models in the future and evaluate how well does the spatial transformer captures spatial relations between the agents and its surroundings.
- Testing and performing ablation studies of the proposed model to assess how each module of the model contributes to the final result.
- To propose future modifications to the designed model to make it better, based on what the experimentation and ablation studies reveal.

1.4 Thesis Structure

This document is structured as follows

1. Given the fact that we are using a Deep Learning approach, data is a main factor to take into account. Chapter 2 describes the datasets we have used to train the proposed model. The chapter starts by taking a brief look into the Relational Database Model because one of the datasets is structured in that way, and having a little background in that area might make it easier to understand the dataset.
2. Chapter 3 introduces the Transformer model. It is necessary to point out that a basic understanding on how neural networks work is mandatory. This should not be a problem considering that the background gave material of more complex models like RNNs. After the Transformer model is introduced, a description of the core of this work, the ST-Transformer, is presented.
3. Chapter 4 addresses the performance of the model and it explains and comments the ablation studies that were conducted to evaluate how each part of the model contributes to the final result.

4. Finally, the conclusions of the work are presented and what is left as future work will be discussed in this part as well.

Chapter 2

Datasets

Deep Learning models require big amounts of data to be trained successfully. It might be clear that the data used to train a model, depends on the purpose of the model. In our case, the model requires data that contains trajectories of agents (vehicles and persons) of interest. Trajectories in the driving context are retrieved by an **ego-vehicle**. An ego-vehicle is a car equipped with different sensors such as cameras, lidar and radar sensors to obtain data about what surrounds the vehicle. Then, the data is processed and translated into useful information such as the position of the agents, recordings and images retrieved by the cameras, a 3D reconstruction of the scene, etc. It is this ego-vehicle that provides the agent's trajectories that will be fed into the model. It is necessary to highlight that the data is sampled at a given frequency (that depends on the sensors), so we have information at certain moments or time steps of the entire scene. Not all the sensors sample at the same frequency, therefore it is necessary to unify the retrieved data to get uniform time steps.

In this Chapter, we describe the datasets (NuScenes and Shifts) used for the experimentation and training of our models as well as the way data is preprocessed to get the information needed for the tasks of training and predicting. A brief summary of the relational database model is presented as well, to facilitate the understanding of one the datasets.

2.1 Relational Database Model

Datasets are basically databases that hold very specific information about some area, usually a practical problem or field of study. It is then useful to have a basic understanding of some database concepts. In the following, a quick introduction of these concepts is going to be made for those readers that are not so familiarized with the database terminology. If the reader feels comfortable with the topic, he/she can skip the following section, and if the reader wants to go deeper in the subject, [10] is a good reference.

A database has the purpose of storing useful information in a structured way so that we can later do something with it. Most databases can be seen as a collection of several tables holding **records** (rows) of objects in the real world. These records are described by **fields** (columns) of these tables representing which attributes are necessary to store from these objects. Most of the tables have a field for storing a unique id for each record that identifies it among the other ones. This field is known in the database context as **primary key**.

Consider the following example to fully understand the previous concepts. The reader can imagine an owner of a store that wants to save information about his employees. This makes it necessary to have an EMPLOYEE table. He decides that it is useful to store the name, date of birth and phone number of each employee. These attributes become the fields or columns of the table. Each employee is then a record or row in it. Finally, to identify each employee in a unique way, a worker id number is assigned to each one of them as a primary key. Using the name as a primary key would be a mistake because two employees could have the same name and therefore the uniqueness of this field would be violated. So, this table could look similar to Table 2.1.

EMPLOYEE			
Worker id	Name	Date of Birth	Phone Number
001	John Fletcher	05/22/1995	473 1000 123
002	Pierre Cardin	06/15/1994	473 2000 456
003	Lois Hestenes	07/28/1997	552 2251 423

Table 2.1: Employees table with 3 records.

The **Relational Database Model** is a way to store the information in a database in an efficient manner. The idea of this model is to avoid **redundancy** in the stored data to save space and to ensure **consistency**. Redundancy could be defined in a practical way as storing the same information or field for a record in different tables. At this point, the reader could ask himself why it would be necessary to store the same piece of information in different tables in a database. This is usually a modelling mistake and it will be addressed with an example in the following paragraphs. The consistency part follows then from avoiding redundancy. If a database stores the same fields for a record in several tables and then these fields are modified in one of the tables, then it has to be done in the same way in all the other tables holding the same information. If not, then the database is not consistent because it contains different values for a given field for the same record, as a consequence of having duplicated fields in different tables.

A relational database model is comprised by two main components: entities and relationships. **Entities** model the objects (tangible or abstract) in the real world from which we want to store and retrieve information and they become tables in the database. An **instance** of an entity is then presented as a record (row or tuple) in the respective table. Entities hold **relationships** between each other. From a technical point of view, these relationships are defined by the presence of a field in a table, known as **foreign key**, which refers to an instance of another entity (another table) and is in fact the primary key of the instance in the other table. The foreign key can then be seen as a pointer to a record in another table. Suppose that two entities, entity A and entity B, exist in a database. The relations these entities could hold between each other can be classified as:

1. One to one. One instance of entity A is related to one instance of entity B.
2. One to many. One instance of entity A [B] is related to several instances of entity B [A].
3. Many to many. One instance of entity A is related to several instances of entity B, and these instances of B are as well related to several instances of entity A.

As we did before, an example will make it easier to understand these concepts. Consider again the previous example of the store and the employees. Take now into account the following considerations due to the business model:

- The store has several branch offices. It is useful to keep track of the following office attributes: office id, address, and phone number.
- The store sells technology products. Each product has the following attributes: product id, description, price, stock units. Suppose that the stock units are stored in a central place and therefore each office displays the same number of stock units for each product.
- An employee can work in only one office.
- Clearly, an office has several employees. It is useful to know at which office an employee works.
- It is useful to keep track of which products have been sold by an employee.

The first two points immediately yield two more tables, call them OFFICE and PRODUCT tables. In the figure 2.1, the reader can see the initial diagram of the database.

As it was stated, it is useful to know in which branch office of the store the employees works. It is also important to consider that a branch office can have several employees and therefore is related to several employees (one-to-many relation). Imagine that, at one point, the owner of the store wants to know the address of the office at which one of his employees works. A naive way to model this problem, and then to be able to retrieve this information, would be to store all the fields of the branch office in the EMPLOYEE table. That enables the owner to know the address of the office where his employees work. This obviously leads to redundancy, because the attributes of the office are saved into the OFFICE table as well as into the EMPLOYEE table. The way the relational database model tackles this is by adding a field to the employee table that refers to the office in which the employee works. This field contains the **primary key** of the office it is referring to and is the **foreign key** in the EMPLOYEE table. In this way, the information of the store in which the employee works can be retrieved. Table 2.2 shows how this information would look in a table view in a database.

The employees-products relation is a little bit different. A product can be sold by several employees and an employee can sell several products (many to many relation). In this

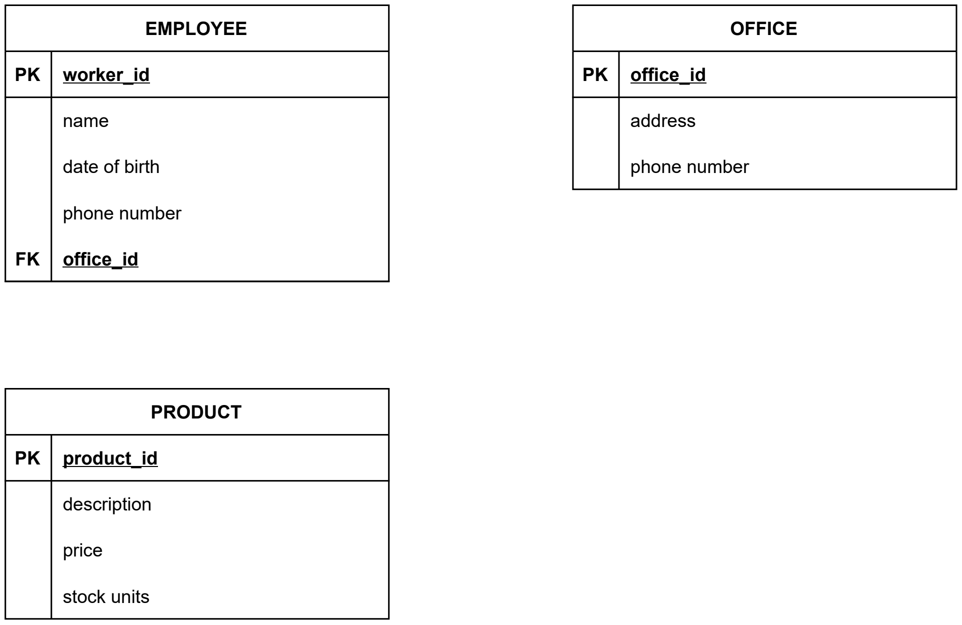

Figure 2.1: Initial database diagram. The PK symbol indicates that the field is in fact the primary key.

OFFICE		
Office id	Address	Phone Number
off-01	address office 1	800 1001
off-02	address office 2	800 1002
off-03	address office 3	800 1003

EMPLOYEE				
Worker id	Name	Date of Birth	Phone Number	office id
001	John Fletcher	05/22/1995	473 1000 123	off-01
002	Pierre Cardin	06/15/1994	473 2000 456	off-02
003	Lois Hestenes	07/28/1997	552 2251 423	off-01

Table 2.2: OFFICE and EMPLOYEE tables with 3 records each. Notice that not all the offices in the OFFICE table need to be referenced in the EMPLOYEE table. However, the office id must exist in the OFFICE table.

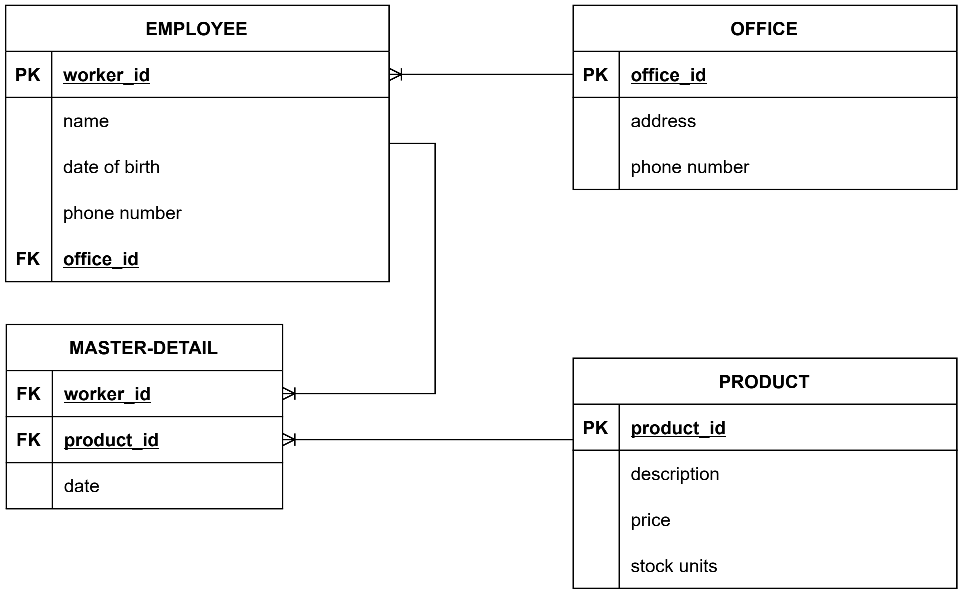

Figure 2.2: Final database diagram. The FK symbol indicate that the field is a foreign key. Notice the direction of the lines. The side of the line with the fork indicates the "many" side. One office is related to several employees, so the fork side of the line is in the EMPLOYEE table. One employee is related to several products, and one product is related to several employees, then a Master-Detail tabled is created.

case it is not as straightforward as to add a field to the EMPLOYEE table referencing the PRODUCT table and doing the analogous for the PRODUCT table. This is because if an extra field for the EMPLOYEE table is added containing the reference of the product which was sold by the employee, then it only contains one of the many products the employee could have sold. An analogous thing happens to the PRODUCT table, it could only save one of the many employees that could have sold the product. To solve this little problem, the creation of an extra table, called the **Master-Detail** table, is mandatory. This table contains all the pairs of <employee id, product id> that indicate which employee sold which product and an extra field containing the date the product was sold. The final diagram can be seen in figure 2.2.

An important thing to highlight is then that whenever a many-to-many relation exists, a specific Master-Detail table needs to be created to model in a correct way this relation.

2.2 nuScenes

NuScenes [6] is a dataset developed by Motional (formerly nuTonomy). The dataset is comprised by 1000 driving scenes, where 850 of these scenes are used for training and validation, and the other 150 scenes are used for testing. Each scene is 20 seconds long and has data collected by a series of sensors (Lidar, Radar, and Cameras) to provide the information necessary to formulate several problems and use the information to train models for specific tasks. As it might be obvious, one of those tasks is the trajectory forecasting for autonomous vehicles, the focus of this thesis.

In the figure 2.3, we can see the dataset schema as a relational database model. This makes it easier to understand how the dataset is organized and how the information can be retrieved. The entire model will not be discussed here, but the main entities and their fields that aid to retrieve the information needed for the trajectory forecasting task will be discussed.

To understand the model, it is necessary to point out some specific details of the dataset schema that can be seen in figure 2.3. The information is presented and can be handled as a relational database model. The arrows are used to indicate that two entities are related, but they do not specifically indicate the cardinality of the relation. However, based on the diagram, it seems like the side without the arrow triangle indicates the "many" side of the relation. Although cardinality is not stated explicitly, it can be deduced from the foreign keys and a basic understanding of what the data represents. Every entity has an implicit field that is not stated in its attributes, and that is the **token** field as a primary key. So, for example, the scene entity has a field named **token** that identifies the scene in a unique way. The relations are defined as well by the presence of a foreign key. We can see that the instance entity is related to the category entity, therefore it contains the category_token field as a foreign key that points to the category it is referencing. All the foreign keys follow the same format for the name of the field, that is **"<entity_name>_token"**. The **instance entity** is not to be confused with the concept of **instance** discussed in the Relational Database Model section. Instance is used in this context as an instance of an agent, for example a specific vehicle or pedestrian in a scene. Finally, given the nature of the problem, positions and orientations of the agents are the most valuable information to be used in the trajectory forecasting task, so it is useful to know how this information is presented. Positions are encoded in a 3D fixed coordinate system as (x, y, z) and rotations are encoded as quaternions.

As it was stated before, the complete model will not be discussed. Furthermore, some entities are basically self-explanatory from their name and attributes and provide a way to limit the values some fields could take. For example, the **category** entity contains a limited number of options in which an agent could be classified. In the next lines, the entities and their fields that provide useful information needed for the trajectory forecasting task will be explained.

1. **Scene entity**. The scene entity is used to model the 20 seconds long driving scene.

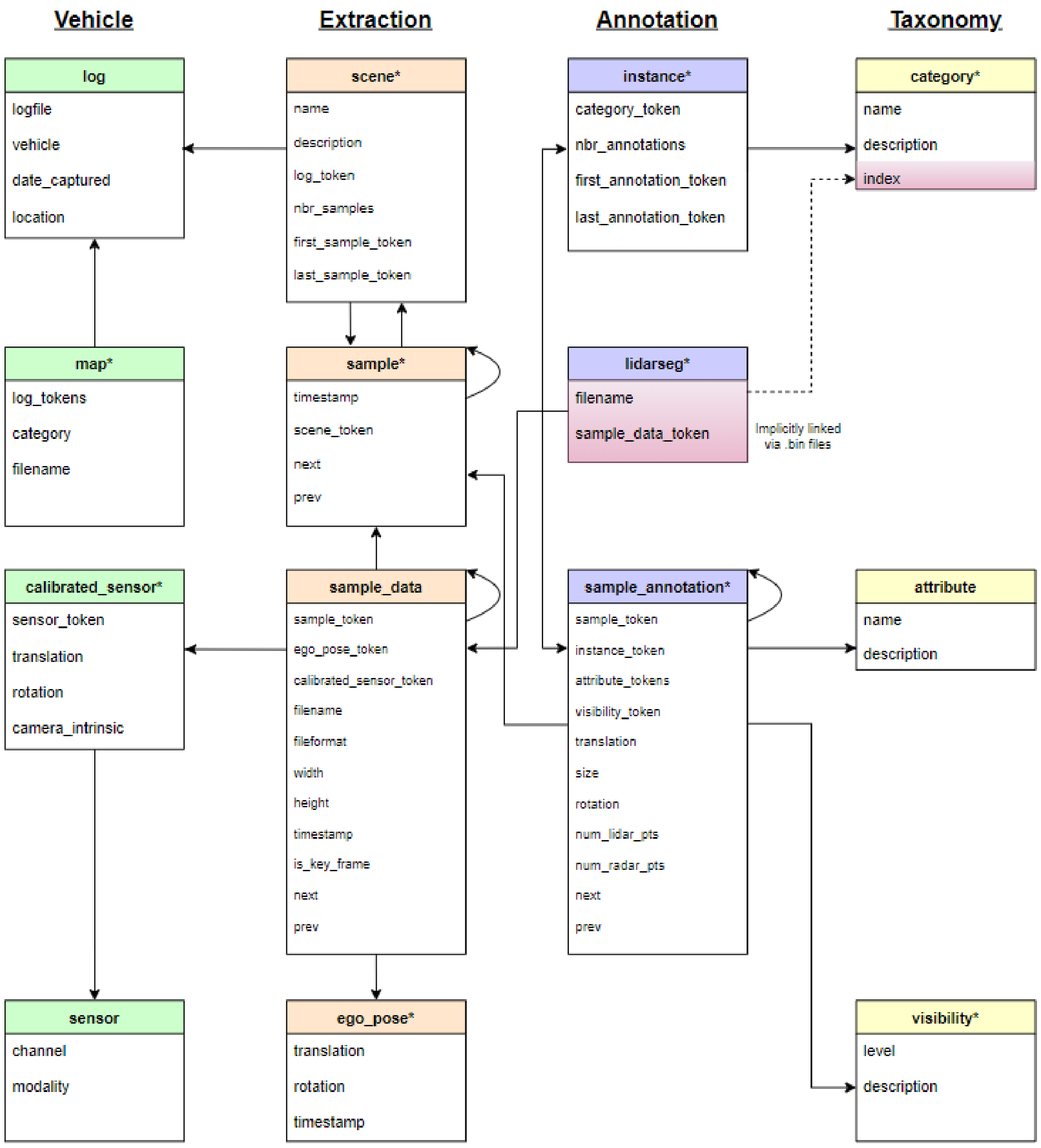

Figure 2.3: Nuscenes Database Schema. Obtained from: `https://www.nuscenes.org/nuscenes?`

Each scene is comprised by several instances of the **sample** entity. It can be deduced then that the scene has a one-to-many relation with the sample entity. The reader can also notice that there is an arrow pointing to the sample entity. That is because the scene entity has two fields pointing to the first and last samples in it.

2. **Sample entity.** A sample represents an annotated keyframe at 2Hz. It could be seen more informally as an “instant” in time. The word “annotated” means that, at this moment, neighboring vehicles and their information are annotated and stored. Something interesting to note is that it has a self-relation indicated by the arrow pointing to itself. The **next** and **prev** attributes define this self-relation. These foreign keys point to the next sample and the previous sample, respectively. One of them contains an empty string when the sample is the first or last of the scene. As it was mentioned before, a scene is comprised by several sample instances, and these two attributes allow to know the order of these annotated time steps.

   Now that the sample entity has been described, we can address the **first_sample_token** and **last_sample_token** fields in the scene entity. These fields contain the references to the first and last sample of the scene, respectively.

3. **Instance entity**. An instance models the information of particular agents or objects in the scene, e.g a particular vehicle or pedestrian. It is important to know that instances are not tracked across different scenes. The field category_token contains a reference to the **category** entity which basically contains the categories in which the agents can be classified, e.g vehicles, pedestrians, etc. To discuss the attributes **first_annotation_token** and **last_annotation_token**, it is necessary to first clarify what the sample_annotation represents.

4. **Sample_annotation entity**. As the reader might guess, an instance can be tracked in several samples because agents might show up (and most likely they do) in several annotated keyframes. Obviously, a sample has several instances (e.g., several vehicles and pedestrians showing up in a specific moment in time) in it. So it can be noted that the sample and instance entities hold a many-to-many relation. It is important to remember from the **Relational Database Model section** above that many-to-many relations require a specific Master-Detail table to deal with them. In this specific case, the **sample_annotation** table fulfills that function. Basically, this table contains the record of each agent in time for a given scene. Visually, this entity can be seen as a 3D box surrounding the agent in the video taken from the cameras, and the **size** field contains the dimensions of this box. This entity contains the main information to build the trajectories that are going to be given as an input to the forecasting model. The **translation** attribute contains the position of the agent in the world coordinate system, and the **rotation** field contains the rotation of the agent respect to the world coordinate system axis (as a quaternion). This entity also contains self-referencing attributes pointing to the next and previous sample_annotations of the agent in time.

Once again, one of them contains an empty string if the sample_annotation is the first or last across the scene.

Once again, now that the sample_annotation entity has been described, we can address the first_annotation_token and last_annotation_token fields in the instance entity. These fields contain the reference to the first and last annotation of the instance in the scene respectively.

5. **Sample_data entity**. In each sample, data is retrieved from different sensors. To be more specific, vehicles are equipped with 5 radar sensors, 3 in the front and 2 in the back; 6 cameras, 3 in the front and 3 in the back; and one Lidar sensor. All these sensors have different frequencies, which means that they read information from the real world at different points in time. The data read by these sensors is then stored in files in the computer of the vehicle. The sample_data entity is used to hold the information of these files to be able to retrieve it when necessary. The **filename** field stores the path where the respective file is located. The **width** and **height** fields only apply when the file is in fact an image. The **timestamp** field contains the time stamp in Unix format. This entity also contains self-referencing pointers to the **next** and **previous** sample data from the same sensor that follows or precedes this in time. The **is_key_frame** field (boolean field) indicates if the data belongs to a key frame. If this is the case, the timestamp should be really close to the timestamp of the sample it points to.

6. **Ego_pose entity**. This table contains the ego_pose of the vehicle retrieving the data with the sensors. The ego_pose contains the translation and rotation of the vehicle and a timestamp.

Figures 2.4 and 2.5 show some illustrations that can give an easier way to understand how a scene, or more precisely a sample from a scene, can be seen in the real world. In the **data type** subsection of the appendix, the reader can find a data dictionary which contains the data type for every field in the aforementioned tables.

2.3 Shifts

Shifts [18] is a dataset proposed by Yandex for the field of uncertainty estimation. The dataset is comprised by 3 sub-datasets for the tasks of weather forecasting, machine translation and trajectory forecasting for autonomous driving systems. Our work will focus on the trajectory forecasting dataset and, for the rest of the section, this dataset will be referred to as the **Shifts dataset**.

Shifts dataset is easier to understand and to retrieve the data from. In figure 2.6, we can see the scheme of how the data is stored. The dataset has 378000 scenes to be used for training and 36000 scenes used for validation/testing. Each scene consists on a list of 50 time steps, and each time step consists on a list of the agents present in it with their

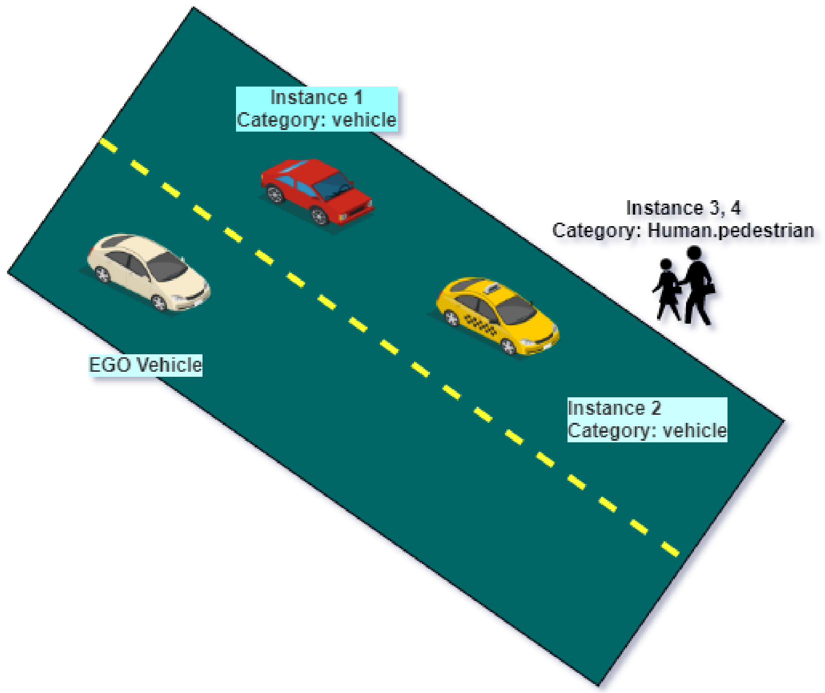

Figure 2.4: Nuscenes scene at specific time (sample table) t_n. We can visualize how an instant in time looks for a scene. The ego-vehicle is the one that retrieves data from the real world through the sensors. There are two instances of vehicles, and two instances of pedestrians. Notice how the category can have subcategories separated by a dot. Instances are stored in the instance table, and their position and information through time is saved in the sample_annotation table.

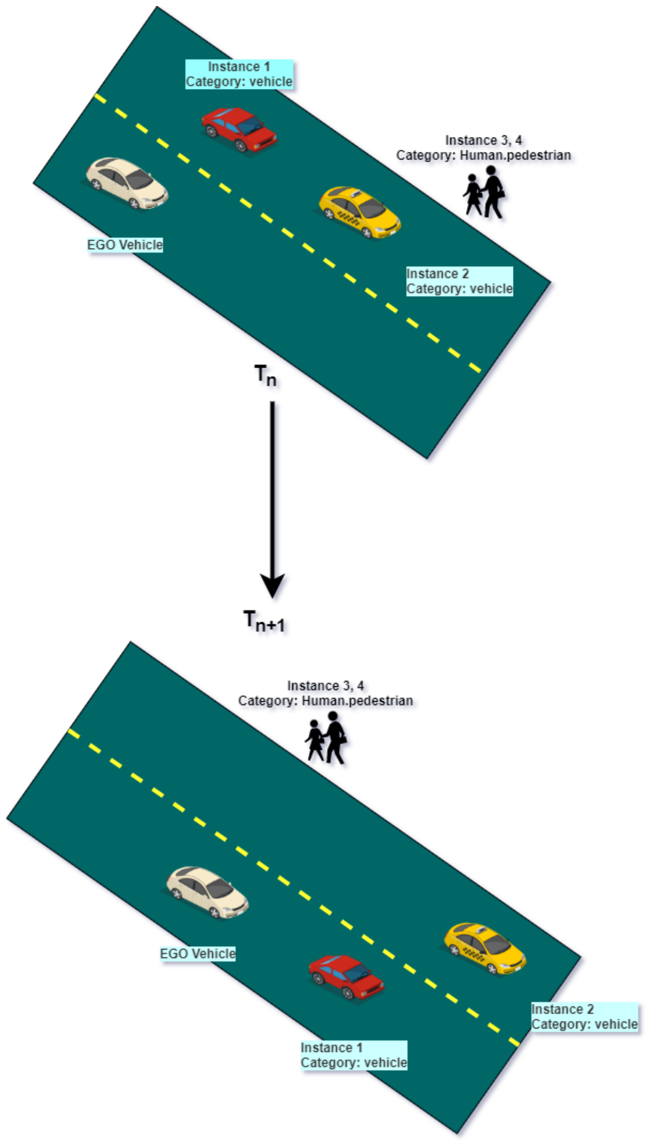

Figure 2.5: Nuscenes transition from instant t_n to t_{n+1}. The transition from t_n to t_{n+1} is stored in the sample_annotation table. Instances have the same token (primary key) but the sample is the one that is different. Keep in mind that more instances could have appeared or disappeared, e.g. the red car.

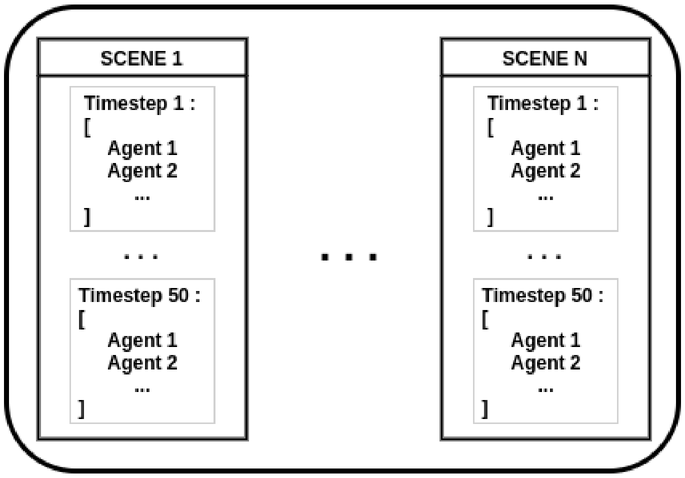

Figure 2.6: Shifts Scheme. Each scene consists of 50 time steps. The first 25 of them are used as the past, and the other 25 as the ground truth future.

respective information (position, rotation, etc...). The ego-vehicle information is stored in a similar format, a list of 50 time steps. The first 25 time steps are used as the past, and the other 25 time steps are used as the ground truth future to be compared with a model prediction. Each time step is sampled at a frequency of 5 Hz, which implies that the task consists on looking ahead 5 seconds into the future.

2.4 Data Format

As it has been seen in the previous subsections, data is organized in different ways through each dataset. Nevertheless, neural network models (and machine learning algorithms in general) receive information in a structured and standardized format through their input layer. This section has the purpose of describing this format and it should be understood as a preprocessing stage of the data before it is fed into the model pipeline. Figure 2.7 shows how the data flows from the datasets to the pipeline model. It can be seen that preprocessed data is stored on the disk as pickle files (a specific Python protocol) to save time as data is read from these files when possible to avoid preprocessing the data again.

Preprocessed data is presented as a model comprised by several Python dictionary objects, which are very similar to a '.json' structured document. Figure 2.8 shows the central scheme in a UML diagram of how both datasets are stored to retrieve the inputs for the model.

The dataset class has four attributes. The agents attribute is a dictionary that contains pairs of the form {<id>: agent object} in which the id identifies in a unique way each agent present in the dataset. The ego-vehicles and non-pred-agents (i.e., those agents that are not predicted) dictionaries have a similar structure. The ego-vehicles dictionary stores information about the ego-vehicles and obtain the information through their sensors. It is

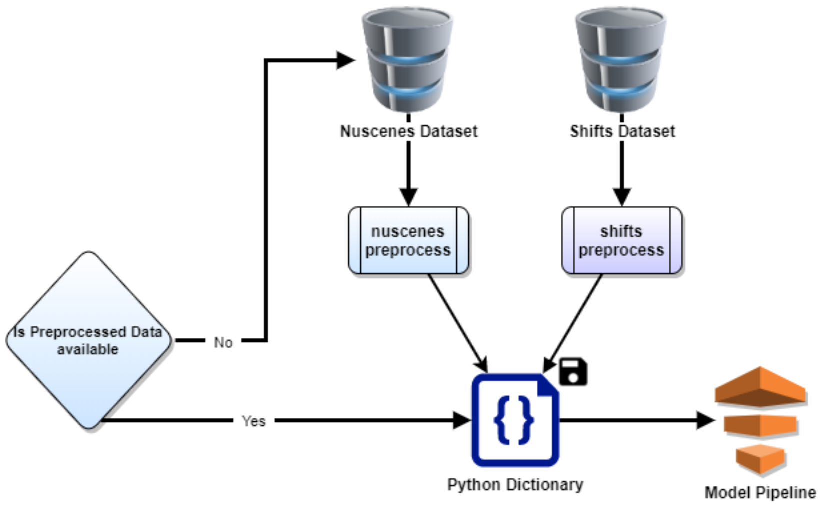

Figure 2.7: Dataflow Diagram. Preprocessed data can be used if it is available. In case it is not available or it is necessary to preprocess the data again, data is read from scratch from the datasets and after the preprocessing stage, a Python dictionary (similar to a .json file) is stored in disk and in memory and fed into the model pipeline.

important to note that there exists a one-to-one relation between the ego-vehicles and the scenes they record. The non-pred-agents dictionary stores agents that are not candidates to have their trajectories predicted. This, in particular, depends on factors like the fact that some agents may have too few points to build a decent trajectory or to be fed to the model as input.

The agents class represents the structure for all the agents and ego-vehicles in the dataset. The ego-id field refers to the id of the ego-vehicle that observed that particular agent. In the case that the agent itself is an ego-vehicle, then that field contains the same value as the agent-id field.

Each agent has a dictionary of their time steps identified as well by an id for each of them in the entire dataset. The Timestep class models the structure of each time step for a specific agent, and contains basic information such as the x and y coordinates, and the yaw rotation in the XY plane.

Finally, to model the neighboring relations of each agent, the Context class is used. The context-id is the same as the time step id used for the time steps for each agent, so once again there exists a one-to-one relation between each time step and the context they define. The neighbors dictionary of a context contains the ids of each agent present in a time step for some scene. This allows to retrieve easily the neighbors each agent has at a specific step in time.

This central scheme allows a flexible yet easy way to access the information and build complex inputs as desired for the model. For example, if a model uses as input all the agent trajectories, the agents dictionary can be used to traverse the timesteps dictionary of each agent and obtain the needed information.

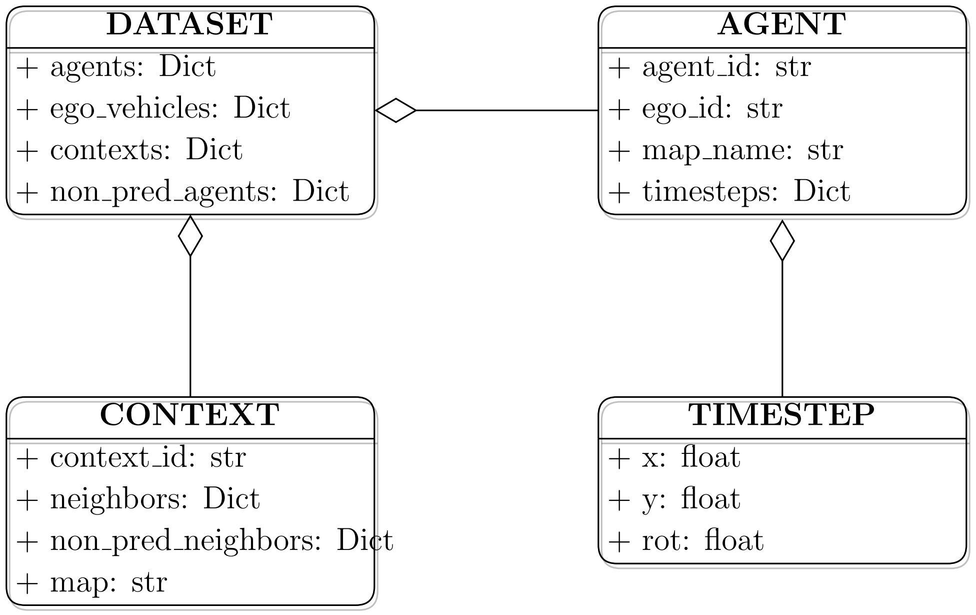

Figure 2.8: Dataset Model.

Chapter 3

ST-Transformer

As it has been mentioned before, seq2seq models have their roots in the NLP field, and Transformers are not the exception. More explicitly, the Transformer architecture was initially proposed for the machine translation task, a kind of problem that benefits greatly from attention models. The model proposed in this work is heavily based on Transformers, so it becomes mandatory to dig deeper into their architecture to understand our proposal.

3.1 Transformers

The Transformer architecture was proposed in 2017 in the now famous **"Attention Is All You Need"** [25] paper. As the name probably gives it away, it describes an attention model. Attention models are those models that can "pay attention" to parts of a sequence to make their prediction. This is an abstract concept and, in most cases, it can be seen as a score that the model assigns to an element of a sequence to indicate how relevant this element is for the final prediction.

3.1.1 Architecture

Transformer models for sequence-to-sequence problems involve an **encoder-decoder** architecture. The encoder is used to obtain information from the source sequence, and the decoder is used to produce the target sequence based on the information provided by the encoder.

Figure 3.1 shows the Transformer model architecture. The block on the left is the encoder, and the block on the right is the decoder. Let us consider the Positional Encoding and the Multi-Head Attention modules as black boxes for the moment (in the following sections, we will go deeper into those modules), and also keep in mind that the Multi-Head Attention module receives 3 inputs that we will refer to as X_q, X_k, X_v; they correspond to the three arrows that point to the Multi-Head attention block.

Both, the encoder and the decoder, receive as input in their first layer the input word embeddings (keep in mind that it was proposed in the NLP field). Then they have a point-

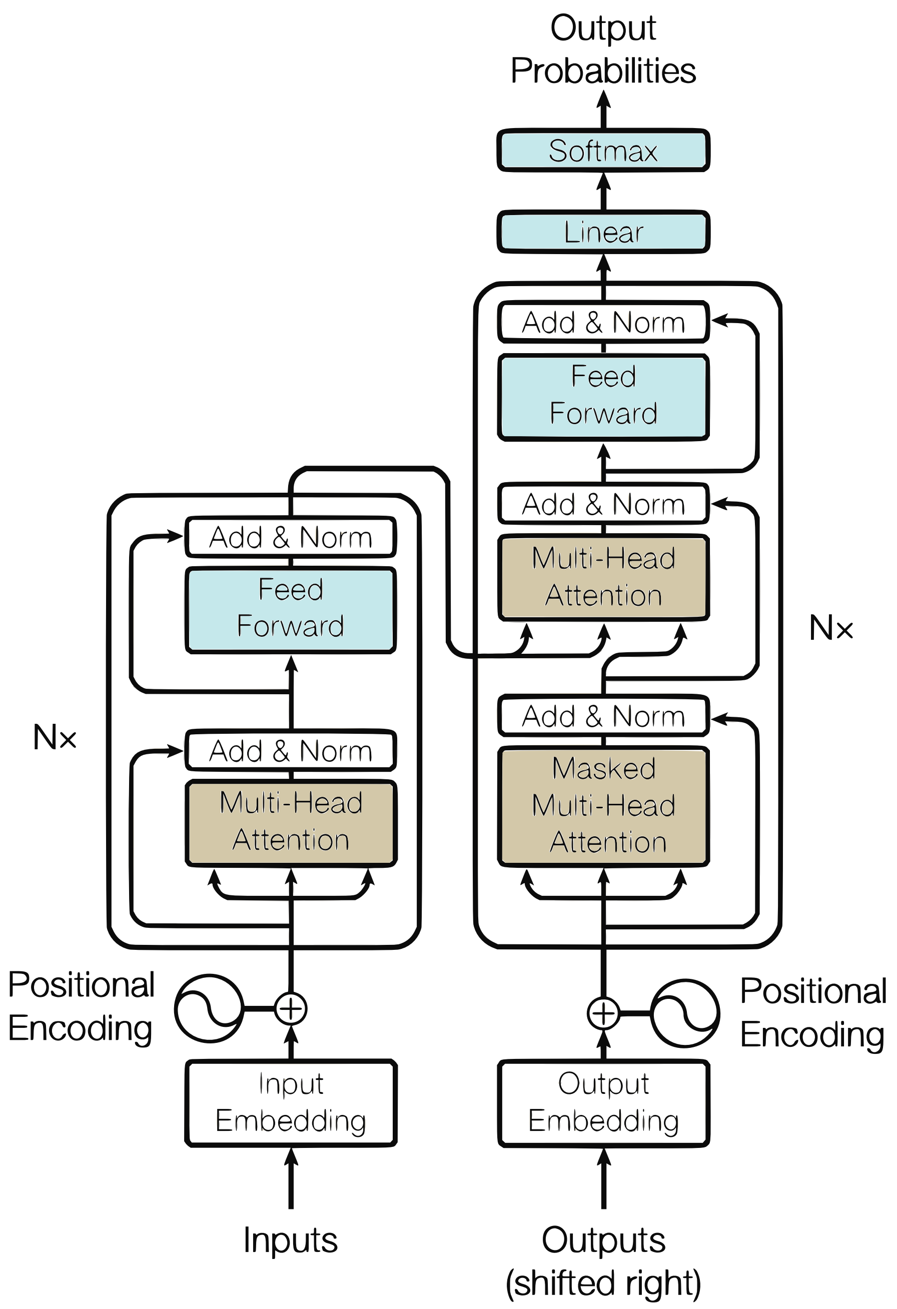

Figure 3.1: Transformer Architecture for word sequence classification. The block on the left is the encoder, and the block on the right is the decoder. The figure was obtained from the original paper.

wise addition operation with a positional embedding vector that, as remarked before, will be explained in another section. In the following paragraphs, both modules are briefly explained.

Encoder. This module consists on N identical layers connected to each other, where the output of the previous layer is used as an input on the following layer. Each layer consists on a Multi-Head Attention module, a residual connection with a normalization layer (Add & Norm block) whose output is then fed into a feed forward module with another residual connection and normalization layer. The first layer among those N receives as inputs in the attention module the embeddings from the source sequence. They are used as the aforementioned X_q, X_k, X_v inputs. The following layers receive the output of the previous layer instead of the embeddings.

Decoder. The decoder has a similar structure to the encoder and consists as well on N identical layers. The main difference is that the decoder makes use of two different attention modules, one for the sequence that is being decoded (and that will be the output) and another one to take into account the encoded information from the source sequence. This is done by using the encoder output as X_q (queries) and X_k (keys) inputs in the second attention module. This can be seen in Fig. 3.1 through the arrows that go from the encoder output to the second attention block. Finally, the output of the decoder goes through a linear layer and a softmax layer (when the problem being solved is cast as a classification problem). In the case of solving regression problems, the softmax layer can be removed.

It is worth highlighting that the encoder always receives the source sequence as input, but the decoder input depends on the mode in which the network is being used. During training, the decoder could work under a technique called **teacher forcing**, in which the ground truth target sequence preceding the element to predict is provided as an input to the decoder. The other option would be to run the model during training in an **autoregressive** manner. In that case, the decoder generates the output sequence from scratch and uses that output at one time step as an input in the next time step, similarly as how RNNs work. During inference, the decoder always works in an autoregressive manner.

Another important thing to take into account is that the residual connections (Add & Norm block) in both the encoder and the decoder aim to attenuate the vanishing gradient problem, due to the fact that these architectures tend to go deep, meaning that they have multiple layers stacked one upon the other. Each of these blocks also contains a dropout layer as a regularizer, to avoid overfitting.[1].

3.1.2 Attention Mechanism.

The attention mechanism used in the Multi-Head Attention block in figure 3.1 will be explained in this section. We will suppose that the inputs X_q, X_k, and X_v are the same as the embedding vectors stacked in a matrix X as columns, i.e.

[1]An implementation of the Transformer architecture in the Tensorflow framework can be found on: `https://www.tensorflow.org/text/tutorials/transformer`

$$X_q = X_k = X_v = X.$$

Let us assume that we are given embedding vectors $x_1, \cdots, x_n$ of dimension m of a sequence of n elements as an input. Suppose that we also have parameters matrices W_Q, W_K, W_V that belong to $\mathbb{R}^{d_k \times m}$[2]. These matrices are used to perform linear transformations of the vectors x_i, with $1 \leq i \leq n$, to another space of dimension d_k and they are the parameters to be learned. Let us denote then

$$\begin{aligned} q_i =& W_Q\ x_i, & (3.1)\\ k_i =& W_K\ x_i, & (3.2)\\ v_i =& W_V\ x_i. & (3.3) \end{aligned}$$

The idea is to use the vectors q_i (referred to as "queries") and k_j (referred to as "keys") for all $1 \leq j \leq n$ to obtain an *attention score* for the elements x_i and x_j of the sequence. Such a score will come weighting the vector v_j (referred to as "value") in the encoding of the sequence element i. This can be done by using the **scaled dot product** between the two aforementioned vectors to obtain a denormalized score,

$$s_{i,j} = \frac{q_i^T k_j}{d_k}, \tag{3.4}$$

and then the attention scores are normalized between all the scores obtained for a fixed query vector and all the key vectors. This is done by applying the softmax function as follows

$$a_{i,j} = \frac{\exp(s_{i,j})}{\sum_{k=1}^{n} \exp(s_{i,k})}. \tag{3.5}$$

Note how in equation 3.5 the index i is fixed in all moment, reflecting the fixed query vector q_i, and the index j is used to run through all the key vectors k_j. These scores are used to obtain the attention computation by performing the weighted sum

$$z_i = \sum_{k=1}^{n} a_{i,k}\ v_k. \tag{3.6}$$

Now, given the fact that, through the implementation, cache memory operations are cheaper in a computer, vectors x_i are practically row vectors rather than column vectors. As matrices W_K, W_Q, W_V are parameter matrices to be learned and are initialized with random numbers, in practice their transpose matrix does not need to be calculated and they actually belong to $\mathbb{R}^{m \times d_k}$. This allows to express the above operations packed as matrices operations. Hence, if X denotes the sequence embeddings stacked in rows as a matrix, and Q (all n queries), K (all n keys), V (all n values) denote their respective projections, we have

[2]The subindex k in d_k is chosen because it denotes the dimension of a vector referred as "key".

$$Q = XW_Q, \tag{3.7}$$
$$K = XW_K, \tag{3.8}$$
$$V = XW_V. \tag{3.9}$$

To sum it up, we have that

$$Z = Softmax\left(\frac{QK^T}{d_k}\right)V, \tag{3.10}$$

where the Softmax operation is computed through the row axis, which means that the normalization is applied through row features rather than column features (to implement Eq. 3.5). Z contains the attention output of the sequence. This attention mechanism is named **self attention** when the attention is computed between the elements of the sequence itself (i.e. q and k have been estimated from the same x). As it was mentioned before, the operation

$$\frac{QK^T}{d_k}$$

is known as the **scaled dot product** and it was originally proposed because the authors found out that it yielded better results.

Masking

The attention mechanism explained above has an advantage over RNNs architectures like LSTMs and GRUs, and it is that the sequence length is not limited by the memory cell capacity to store information. Nevertheless, at a practical level, the frameworks used to implement these architectures usually require to establish a maximum length to the sequences, so that the tensors representing the inputs have a uniform shape. As the reader might guess, in real life problems, the sequences fed to the model will definitely not have the same length, so techniques like **padding** and **truncating** are used to deal with shorter and longer sequences, respectively, than the fixed length sequence that was chosen.

Padding consists on filling the sequence with the needed number of items so that the sequence reaches the maximum length established. The sequence is filled with a special item (named a padding token in the NLP context). The problem with padding is that the sequence that is being modified now contains information that does not represent the real input. **Masking** helps to "remove" this extra information. This is easily done with the attention matrix $Q \times K^T$ from Eq. 3.10, before applying the softmax function. The idea is that the padded elements of the sequence should not provide information for the problem, so their weights indicating their contribution to the output is forced to 0 values (or something very close to 0). The way to achieve this is to add a big negative number like $-1e9$ to the product, so that, when the softmax operation is applied, the score obtained is practically zero. This is implemented by having a matrix M that contains $-1e9$ in the position of the inputs that are padded and 0 otherwise. Then the element-wise addition between M and the attention matrix $Q \times K^T$ is performed to obtain

$$Z = Softmax\left(M + \frac{QK^T}{d_k}\right)V, \tag{3.11}$$

where Z contains the output of the attention module, but discarding the masked elements. Let us imagine a toy example in which we have a sequence of length 3, but the maximum sequence length has been fixed to 4. Hence, the last element of the sequence is padded. Matrix M then looks like

$$M = -1e9 \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Let us note that the last row, which practically represents the padded element, is not actually completely masked. This is because, usually, deep learning models go too deep and it will result in the output respecting the last element to contain values that still need to be discarded, so a masking process is still needed when calculating the loss function, or whenever the output of that element is going to be used and has an impact on the next layer. Hence, the reason for not padding the last row is to avoid unnecessary operations that require extra time.

Other scenarios in which **masking** becomes useful, is when some items of the input sequence are missing. A masking matrix allowing to represent the missing data is built in a similar fashion. In that way, their contribution to the attention output is removed.

3.1.3 Multihead Attention

The attention mechanism explained in the previous section can be generalized in an easy way to have several of these attention mechanisms running in parallel. A single of these attention mechanisms is referred as an *attention head*. The way the authors justify the need for several attention heads is by stating that it allows the model to jointly attend to information from different representation subspaces at different positions, i.e. to have some multi-modality. This is achieved by making every attention head have its own W_{Q_h}, W_{Kh}, W_{Vh} parameter matrices with $1 \leq h \leq \#\ heads$. Once again, this can be packed as matrix operations. In fact, some difference actually exists at the implementation level. The idea consists in having the dimension d_k of the original matrices W_Q, W_K and W_V be a multiple of the number of attention heads that are desired in the model, and then the dimension d_{kh} of the matrices of each head is

$$d_{kh} = \frac{d_k}{\#\ heads}.$$

So, if the model has $d_k = 512$ and 8 attention heads, then every $W_{Q_h} \in \mathbb{R}^{m \times 64}$. The same applies to the W_{Kh} and W_{Vh} matrices.

Let us suppose that we are given matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{m\times p}$ and the **row** vector $x \in \mathbb{R}^m$ (or $x \in \mathbb{R}^{1\times m}$). Let us denote $[A|B] \in \mathbb{R}^{m\times(n+p)}$ the matrix that results from concatenating matrices A and B along their columns. It is known that

$$x[A|B] = [xA \mid xB],$$

where both xA and xB are vectors that belong to $\mathbb{R}^{1\times n}$ and $\mathbb{R}^{1\times p}$ respectively.

Based on this, it can be seen that if we have r attention heads, we can pack all the W_{Q_h}, W_{Kh} and W_{Vh} parameter matrices for each attention head as

$$\begin{aligned} W_Q &= [W_{Q_1}|W_{Q_2}|\cdots|W_{Q_r}], && (3.12)\\ W_K &= [W_{K1}|W_{K2}|\cdots|W_{Kr}], && (3.13)\\ W_V &= [W_{V1}|W_{V2}|\cdots|W_{Vr}]. && (3.14) \end{aligned}$$

Using equation 3.7, we have that

$$\begin{aligned} Q &= XW_Q \\ &= [XW_{Q_1}|XW_{Q_2}|\cdots|XW_{Q_r}] \\ &= [Q_1|Q_2|\cdots|Q_r] \end{aligned}$$

and the same goes for K and V. The attention mechanism is then applied as explained before, and this yields matrices $Z_1, Z_2, \cdots, Z_r$, which are concatenated through their columns to obtain

$$\hat{Z} = [Z_1|Z_2|\cdots|Z_r],$$

which packs the attention outputs of every attention head. Finally, this matrix is multiplied by another matrix $W_O \in \mathbb{R}^{d_k\times d_k}$ that is trained in conjunction with the rest of the model to obtain the attention output

$$Z = \hat{Z}W_O. \tag{3.15}$$

3.1.4 Positional Encoding

Time is a crucial factor in time series problems. Similar elements of a sequence that show up in a different time step might end up having a remarkable difference in the overall meaning of the sequence. More so, the meaning of an element in a sequence is usually conditioned on the elements past to it. Therefore, the management of the time variable is something models have to take into account. It is clear how RNNs incorporate the time factor into their architecture. All the elements of a sequence are processed one by one, following the sequence order. In the case of gated RNNs, like LSTM and GRU cells, the output of the model at time step t is heavily conditioned on the past elements through the gating mechanisms these cells implement, and it uses the previous output in time step $t-1$ as an input of the model. The Transformer architecture involves several matrix operations, but none of them takes into account the time variable. To solve this, the authors of [25] propose a mechanism named **Positional Encoding** to indicate to the network, in some way, the *order* of the elements of the sequence.

Positional encoding aims to force Transformer models to learn time dependencies. Several ideas might come into mind to accomplish this objective. The first naive solution would be to add to the input a positive integer that indicates the position of the element along the sequence. Hence, the first element would include as an input the number 1, the second element receives the number 2, and so on. The problem with this solution is that the model might receive sequences of a length it has never seen before, or the sequences seen during testing could be even bigger than sequences during training and that would hurt the model performance.

Another solution could be to add a number $r \in [0,1]$ that indicates the relative offset of an element of the sequence in comparison with the length of the sequence as in

$$\frac{offset}{N}$$

where N indicates the total length of the sequence. The problem with this solution is that the numbers added are not spread uniformly and they depend on the size of the sequence, so it could be expected that the model behaves differently with really long sequences in comparison to really small sequences.

The solution proposed in [25] uses sine and cosine functions to tackle the problems mentioned above. The intuitive idea behind the following formula is to generalize in some way the binary counting system. Let $x_t \in \mathbb{R}^d$ be the embedding vector of an element in a sequence, where $d \equiv_2 0$. The subindex t indicates the position of the element in the sequence. The authors define the positional embedding (or positional encoding) vector as

$$p_t{}^i \triangleq \begin{cases} \sin(w_{k+1} \cdot t) \text{ if } i = 2k, \\ \cos(w_{k+1} \cdot t) \text{ if } i = 2k+1, \end{cases} \tag{3.16}$$

where $k \in \mathbb{N} \cup \{0\}$ and

$$w_k = \frac{1}{10000^{2k/d}} \tag{3.17}$$

and the super index i indicates the ith entry of the vector that is being calculated. It is easy to see from the definition of the frequency in the sine and cosine functions that, as the value i increases, the frequency *decreases*. This implies that, as i grows, the change of the positional encoding at that entry becomes smaller, which can be interpreted as a generalization of the binary counting system in the continuous space. From what we explained, the positional encoding vector at time step t has the following form

$$p_t = \begin{bmatrix} \sin(w_1 \cdot t) \\ \cos(w_1 \cdot t) \\ \vdots \\ \sin(w_{d/2} \cdot t) \\ \cos(w_{d/2} \cdot t) \end{bmatrix}. \tag{3.18}$$

A desired characteristic of the positional encoding vectors is that they should be unique for each time step. The following paragraphs have the objective to prove that the authors proposal do in fact yield unique vectors for each time step.

The period of a sine function of the form $\sin(c \cdot x)$ is $p = \frac{2\pi}{c}$. Therefore, if t is a natural number, the values y for which $\sin(c \cdot y) = \sin(c \cdot t)$ holds, are of the form

$$y = t + \frac{2n\pi}{c} = t + n\,p, \tag{3.19}$$

with $n \in \mathbb{N}$.

Theorem 1. *Suppose that we are given $p \in \mathbb{I}$ (irrational numbers) and $t \in \mathbb{N}$. Suppose that y has the form*

$$y = t + np,$$

with $n \in \mathbb{N}$. Then, for any $a_0 \in \mathbb{N}$ we have that

$$y \neq t + a_0.$$

Proof. To prove this, let us assume that the equality holds for some $a_0 \in \mathbb{N}$. It follows that

$$\begin{aligned} t + a_0 &= t + np \\ \Leftrightarrow n &= \frac{a_0}{p} \\ \Leftrightarrow p &= \frac{a_0}{n}. \end{aligned}$$

As we supposed that $p \in \mathbb{I}$, then we have reached a contradiction because p would be rational. Hence, $a_0 \notin \mathbb{N}$. ■

Theorem 1 basically states that if the period of the sine function is irrational, then the result of the sine function applied to each time step will be unique (since we deal only with integer time-steps). Now, it can be proven that $\frac{2\pi}{w_k}$ yields an irrational number. If w_k is a rational number, then it follows directly that $\frac{2\pi}{w_k}$ is irrational. We have to consider then the case when w_k is irrational. For that, we state the following

Theorem 2. *If c is irrational and is not a rational factor of π, i.e.*

$$c \neq r\pi$$

with $r \in \mathbb{Q}$, then $\frac{2\pi}{c}$ is irrational.

Proof. We will proceed by contra positive. Let us suppose then that $\frac{2\pi}{c}$ is rational. We need to prove that either c is rational or c can be expressed as $c = r\pi$ with $r \in \mathbb{Q}$.

As we have supposed that $\frac{2\pi}{c}$ is rational, then it can be expressed as the ratio of p and q with p and q being relatively prime. So

$$\begin{aligned} \frac{2\pi}{c} &= \frac{p}{q} \\ \Leftrightarrow \frac{2q}{p}\pi &= c \\ \Leftrightarrow \ r\pi &= c \end{aligned}$$

and as we can see from the last equality, c can be expressed as a multiple of a rational number and π. Hence, if c is irrational and can not be expressed as r times π with $r \in \mathbb{Q}$, then $\frac{2\pi}{c}$ is irrational. ■

The above result states that if the constant c in the period of a sine function is irrational and not of the form of $r\pi$ with $r \in \mathbb{Q}$, then the period is irrational. This combined with the result from Theorem 1 leads to the positional encoding vectors for each time step being *unique*. But there is still one piece of the puzzle missing. We need to prove that when w_k, defined as proposed by the authors, yields an irrational number, i.e. that it can not be expressed as rational multiple of π.

Theorem 3. *Given $w_k \in \mathbb{I}$ defined as in equation 3.17 for fixed values k and d. If $w_k = r\pi$ holds for a value r, then $r \notin \mathbb{Q}$.*

Proof. We will proceed by contradiction. Let us suppose that

$$w_k = r\pi, \quad r \in \mathbb{Q}.$$

This implies that there exists positive integers p and q relatively prime such that

$$\begin{aligned} \frac{p}{q}\pi &= \frac{1}{10000^{2k/d}} \\ \Leftrightarrow \pi &= \frac{q}{p}\Big(\frac{1}{10000^{2k}}\Big)^{1/d}. \end{aligned}$$

For ease of manipulation, let us define $c = \frac{q}{p}$ and $y = c\Big(\frac{1}{10000^{2k}}\Big)^{1/d}$. We can see that y is a root for the polynomial

$$x^d - \frac{c^d}{10000^{2k}}$$

and, as c is a rational number, it implies that y is an algebraic number. We have then reached a contradiction, because that would imply that π is an algebraic number, and it is known that π is a transcendental number. Hence, when $w_k = r\pi$ holds for a given $w_k \in \mathbb{I}$, r can not be a rational. ■

With these three theorems, we can now state that for any $a_0 \in \mathbb{N}$ we have that

$$\sin(w_k \cdot t) \neq \sin(w_k \cdot (t + a_0)), \tag{3.20}$$

and the same can be deduced for the cosine function. Therefore, the positional encoding vectors for each time step are unique. Even more, each positional encoding element is unique.

The authors also state that they chose function 3.16 because they hypothesized that it would allow the model to easily learn relations between relative positions of the elements in the sequence, given that for any fixed offset m there exists a linear transformation[3] $\mathcal{T}$ such that

$$p_{t+m} = \mathcal{T}(p_t). \tag{3.21}$$

The final embedding vectors are then obtained by adding the original embedding vectors and their respective positional encoding vector, so we have

$$\hat{x}_t = x_t + p_t, \tag{3.22}$$

and now it becomes clear why the positional encoding vector dimension was constrained to have the same dimension as the embedding vector. To summarize, the positional encoding vector proposed has the following characteristics:

1. It yields a *unique* position vector for each time step t.
2. It can be generalized without effort to longer sequences that were not seen during training.
3. The distance between two time steps from sequences with different lengths is uniform and consistent, i.e, the values for the positional vector are given by an equation that depends on the position of the embedding and change in a consistent manner, not depending on the size of the sequence.
4. For any fixed offset m, there exists a linear transformation $\mathcal{T}$ such that equation 3.21 holds.

[3] A proof of this claim can be found on : `https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/`

3.1.5 Advantages and Disadvantages of Transformers

As it has been seen all along this section, Transformers are attention models that aim to use the attention mechanism to get better results. There are other attention models proposed within the RNN architectures using LSTM or GRU cells, and they implement encoder-decoder architectures as well. They do not require mechanisms like positional encoding to include the time variable, so why would a complex architecture like the transformer would be preferred?

While it is true that positional encoding is not the most natural way to feed the time variable to the model, and there have been other proposals like using a multilayer perceptron (still not a natural way), there are advantages of the attention mechanism proposed with the transformer architecture. One of them is that for problems that do not require a time variable but could benefit from an attention mechanism, removing the positional encoding yields an attention model without further efforts. Note that the time factor is managed solely through the positional encoding and nowhere in the attention mechanism there exists a relation with time. One could think that the same applies for RNNs with attention mechanisms, but they usually run over the output of each time step, and therefore the time variable is present at the attention mechanism as well.

Other of the main advantages Transformers has over RNNs is that they do not limit the number of elements that a sequence can have. RNNs with gating mechanisms like LSTMs use their memory cell to process sequential inputs, but, as the number of the sequence starts to grow beyond certain limit, the performance of the model starts to decrease, given the fact that they need to share this memory along the entire sequence. Transformers, on the other hand, do not have this problem; the limit on the sequence they can process depends more on the hardware rather than on an inherent feature of their architecture.

Given the fact that Transformers process sequences as a whole rather than input by input, their architectures can be more easily parallelized than RNNs. Nevertheless, as it will become clearer in the next sections, there are advantages in some cases as processing sequences in a more “natural” way with an autoregressive approach rather than using mechanisms like teacher forcing that are easier to parallelize.

3.2 ST-Transformer

Now that the Transformer architecture has been explained, we will go through the architecture we propose as the main focus of this work. The idea of the proposed model is to tackle the Trajectory Forecasting problem from two dimensions: the *spatial* dimension and the *time* dimension. The time dimension comes with no surprise, given the fact that we are stating the problem as a seq2seq problem. On the other hand, the spatial dimension does come as a natural way to frame the problem if you consider how agents move through a physical space, taking into account how the actions of the *other* agents affect their own actions. For example, consider figure 2.4. Imagine the vehicle represented by instance 2 moving to the right lane. This now enables the vehicle represented by instance 1 to be able to go faster and even occupy the physical space that vehicle 2 left. The idea to model the problem from a spatial dimension comes from this intuition, and we hoped that this would help the model in performing a better prediction.

3.2.1 Input Sequence Format

Before we dive into the model architecture, a view of the input sequence shape might help to ease the understanding of the model. Imagine that we have an agent denoted by the index a. The features used as input for this agent are the relative position (x_a, y_a) and the relative angle θ_a. When we say relative, we mean relative to a fixed time step for the ego-vehicle (i.e., the vehicle from which the sensory data were acquired), so that the position and the angle of the ego-vehicle at that time step are treated as the origin and the axis rotation respectively (see figure 3.2). This fixed time step is considered as time step zero and marks the boundary from the past and the future (the one that will be predicted). The time steps from the past are represented by a negative integer, and the time steps from the future are represented by a positive integer. Hence, basically, the input is conformed by all the negative time steps up to the zero time step, and the future time steps are the ones that the model must predict to be compared with the ground truth time steps (positive time steps) during training.

All the agents features in each time step are centered and rotated to the new origin and axis rotation. The idea to represent each input in this format is to have space invariant features rather than using the global coordinates defined by each dataset. To consider the spatial relations between the neighbors of a scene at each step, we build matrices that contain each neighbor's features along a fixed row for each agent. Then, each time step is represented by each of those matrices. Figure 3.3 should make it easier to understand. It is important to understand that each neighbor remains in the same row along the whole sequence.

Figure 3.4 shows how the masking of the inputs works. Masking in this model is really important because some neighbors could have information missing in some time steps due to being blocked by other objects or simply because they were out of the detection area of the sensors. The rows of the mask matrix represent the neighbors and the columns represent each time step for each neighbor. It is important to note that the mask in the figure has the first element of the sequence in the most right column and the last element in the sequence in the first column. It was depicted this way to make it easy for the reader to imagine to rotate the mask to match the input along the depth dimension and see how each value of the matrix masks the corresponding row of features in the input.

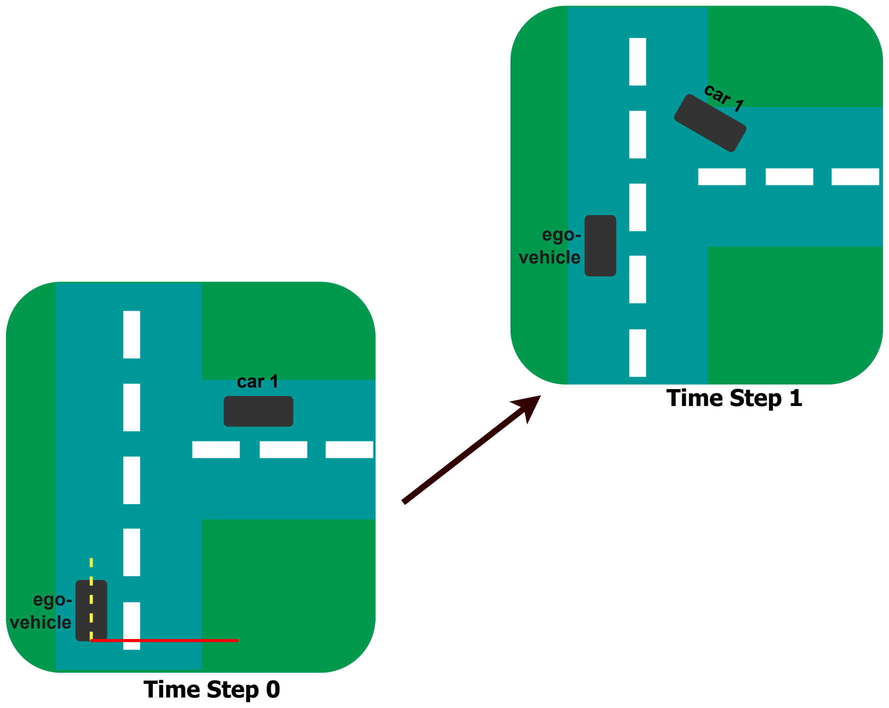

Figure 3.2: Relative Transformations. Time Step 0 is selected as the origin time step, so the position of the ego vehicle at that time step, namely $(x_0, y_0)_{ego}$, becomes the origin. The red line indicates the current axis with respect to which rotations are measured. The yellow dashed line indicates the axis that will be chosen as the new axis. Car 1 position and rotation are then transformed in all the time steps to the new origin and the new axis. Note that this applies to negative time steps as well (the past) and to the time steps of the ego-vehicle. Hence, after the transformation, the position of the ego-vehicle at time step 0 is always $(0, 0)$ and it has a rotation of 0 degrees.

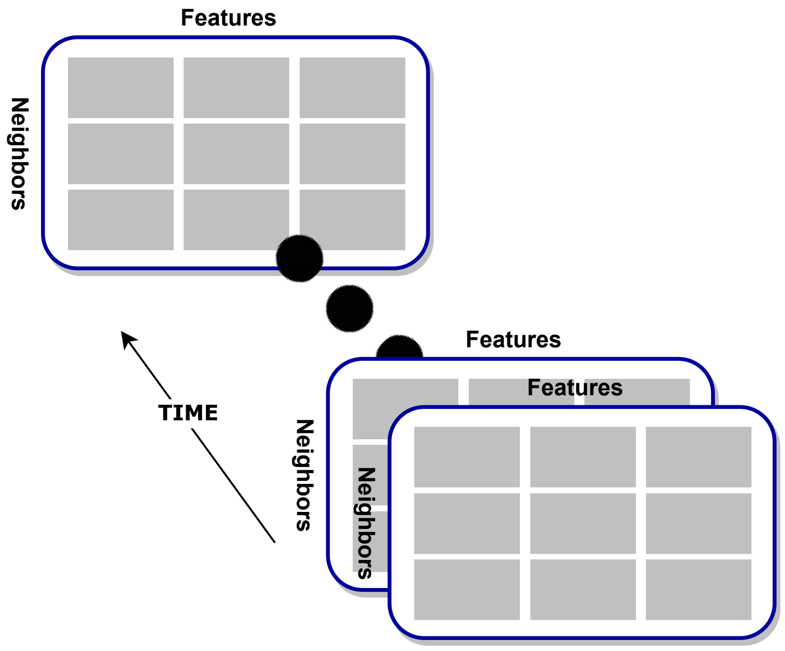

Figure 3.3: Input Shape. The neighbors dimension is along the rows dimension and the time dimension is along the depth dimension.

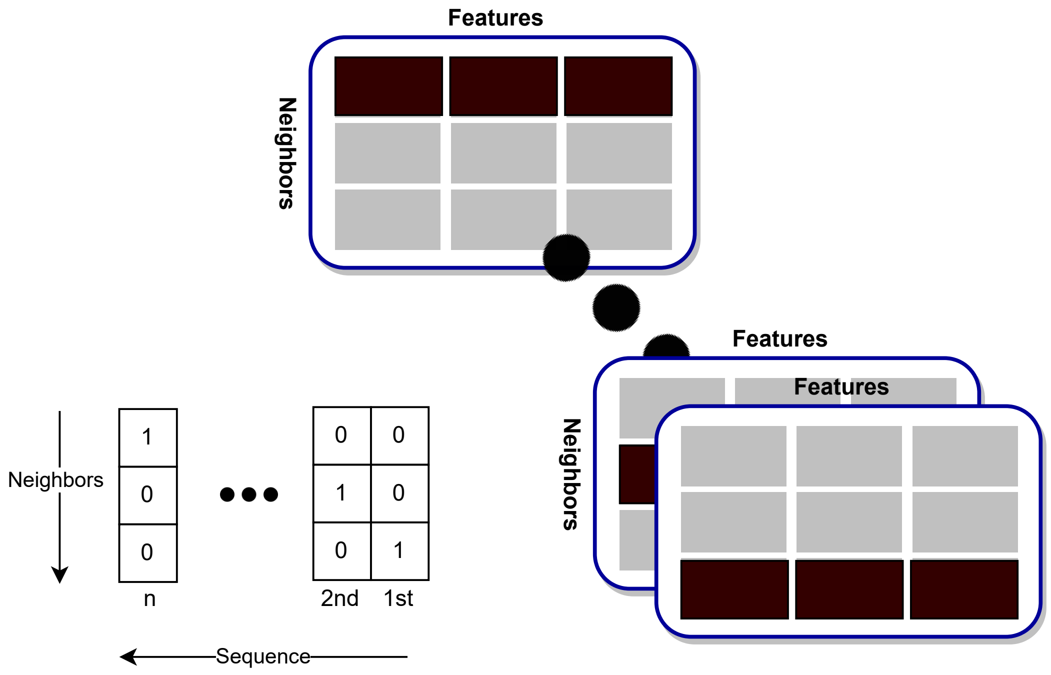

Figure 3.4: Input Masking. The mask is represented by the table at the bottom corner of the image. The black rectangles in the input indicate that the values are masked.

3.2.2 Model Architecture

Now that the input format has been described, the model will be easier to understand. Figure 3.5 shows the architecture of the model proposed in this work. The model works with an encoder-decoder architecture. The **encoder** is used to process the input of the model, and the **decoder** is used to produce the target sequence in an autoregressive manner.

The encoder is conformed by all the blocks up to the Time Encoder. The inputs of this part of the model consist on the raw inputs in the format explained in the previous section, and bitmaps indicating the drivable area, lane division, and the position of the agents along their trajectory. The raw inputs go through a feature embeddings layer (basically a MLP) to get higher dimensional features. The bitmaps, on the other hand, are processed by a simple CNN module to produce feature maps. These feature maps are concatenated with the feature embeddings and then the first transformer comes in the scene. This module pays attention at a spatial level, because the rows contain the neighbors present at a scene for a fixed time step. As it was stated in the previous section, Transformers can yield attention mechanisms for problems that do not require a consideration for the time variable, by simply removing the positional encoding (subsection 3.1.4).

The output of the spatial encoder is transposed along the neighbors and sequence dimensions to be fed into a Transformer Encoder as the X_K (i.e., the keys) input of the attention mechanism in **all the encoder layers**. The inputs X for this module come from a representation of the inputs parameterized as "speeds" and with the time dimension along the rows as well. We refer to them as speeds because we represent each input as a difference vector for each agent a as

$$\Delta_a^{(t)} = \left(x_a^{(t+1)} - x_a^{(t)},\ y_a^{(t+1)} - y_a^{(t)}\right),$$

and, as the time between each input is constant, they can be seen as speeds. These values go through the positional encoding of the time Transformer Encoder and then they are used as X_Q and X_V for this second attention mechanism, using then the output of each layer as the input for the next layer. This module now pays attention at a time level, because the rows now contain each time step and the neighbors are positioned across the depth of the input.

Each transformer receives as inputs their respective masks that indicate if a neighbor information is present or not at a given time step. At this stage, the encoding phase has finished. The decoding phase involves just a time Transformer Decoder similar to the normal Transformer, using the same speed format described above. This means that the trajectory needs to be reconstructed by performing the sum of the outputs at each future time step to obtain the final trajectory. From figure 3.5 we can see that the decoder works in an autoregressive manner even during training, rather than with the teacher forcing approach. This was chosen that way because we saw that it hurt the performance of the model if teacher forcing was used, something that the work in [15] work also takes into account.

Finally, the loss function used to train the model is the Mean Squared Error (MSE) function

$$MSE(P_{pred}, P_{real}) = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{2}\Big[(x_{pred_t} - x_{real t})^2 + (y_{pred_t} - y_{real t})^2\Big], \tag{3.23}$$

where P_{pred} and P_{real} correspond to the sets of points that conform the predicted and real trajectory, respectively, and T corresponds to the number of points in the trajectory.

At this point we must take the time to clarify something that might not be as evident as we would like it to be. Based on figure 3.3 the reader might think that the Transformer implemented in this work looks really different from the original model in [25], but that is not the case. As a matter of fact, the attention mechanism comes from the same product of the matrices QK. The difference from the spatial and temporal attention basically depends on which dimension is along the rows (the neighbors dimension or the sequence dimension) and the remaining dimension could be seen as an extended batch dimension. Technically speaking, the implementation of this model looks like the implementation of the original Transformer, with differences in the masking mechanism, the use of the positional encoding, and in case of the Time Transformer the fact that it receives the output of the spatial encoder as X_K rather than the same input as the X_Q and X_V inputs (like in the original Transformer).

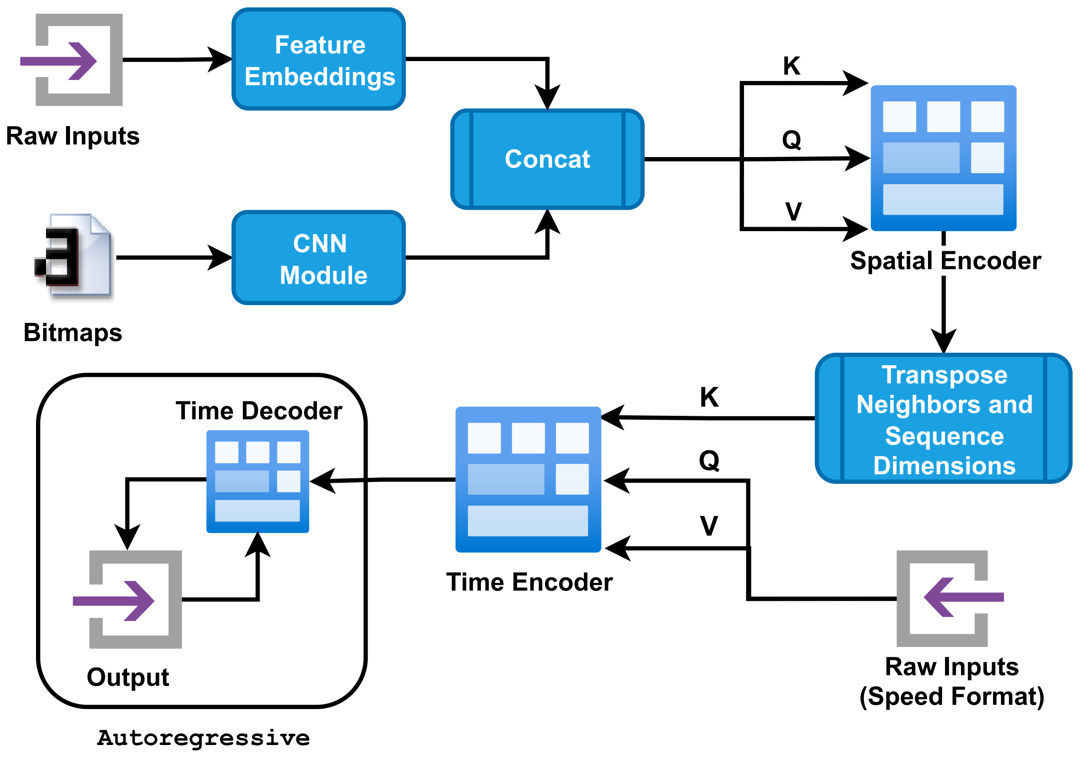

Figure 3.5: Model Architecture. The inputs for the model comprise the input sequence explained in the previous subsection and their respective masks, that indicate if a neighbor's information is present or not at a given time step. The bitmaps used as an extra input contain information of the drivable areas and their lane divisions. There is a bitmap for each neighbor in the scene because they also contain each neighbors past time steps stamped in it. The Encoders are almost the same as the Encoders proposed for the original Transformer, so the interpretation of spatial or temporal attention comes from the elements that are in the rows. If the neighbors are in the rows of the input, we call it Spatial Encoder. If the sequence elements are along the rows, then we call it Time Encoder. In the case of the Time decoder, it is also the same as the original Transformer. In this case we do not indicate the K, Q, V inputs because they work the same as the Transformer proposed in the NLP field.

Chapter 4

Experimentation and Results

In this chapter, we discuss the results obtained with the model proposed in this work. First, we start by specifying the hyper parameters used to train the model, as well as the optimizer chosen to perform the training. We also mention a couple of considerations taken into account to obtain better results. We specify in which environment the training was executed. Then the results that we obtained are discussed and compared through ablation studies carried out to analyze how much each part of the model contributes to the final result. Finally, we visualize some of the obtained results, altogether with the spatial attention computed by the spatial module to analyze in which parts is this module focusing its attention.

4.1 Training

In table 4.1, the reader can see the values chosen for the model hyper-parameters. The temporal and spatial dimensions d_k are fixed to a value of 256 as proposed in other papers like the AgentFormer [15]. We explored other values like 512, but they did not yield better results and made the training process slower. In the same way, we choose to stick to only 1 layer for the spatial encoder, because more layers did not improve the results and made the training phase slower and, in some cases, it even degraded the performance of the model. This could be due to the model having a too large number of parameters to adjust in comparison to the data available to train the model.

The convolutional module is a hand-crafted module with no greater thought, given the fact that the bitmaps given as inputs are relatively simple.

At this point, we must mention something important. The original Transformer model contains a feed-forward layer after each encoding and decoding layer that are supposed to push the transformer out of local minima. Nevertheless, the model proposed in this work has those layers removed and it is why the feed forward hidden dimension parameter is missing from table 4.1. The feed-forward module was removed because the results obtained were better that way. This could be due to the little data used to train the model in comparison to the data available to train models in the NLP field. It is left as a future work to reactivate the feed forward layers and see if better results can be obtained. It is hypothesized that it should be the case given that researchers in [17] explored the function of the feed forward layers in transformer-based language models and found that they operate as key-value memories.

The softmax function used for the attention computation in equation 3.11 is also substituted by a **sigmoid** function and, as a consequence, the attention modules do not normalize their attention. This was proposed in [24], arguing that there was no reason for why the normalized attention would help the model. This comes in opposition to the Transformer in the NLP context, in which the point of normalizing the attention is to weight how relevant a word is to another but considering the effect of other words at the same time. The work in [24] got to see through ablation studies that changing the softmax function by a sigmoid function, yielded better results.

Model Parameter	**Value**
# Spatial Encoder Layers	1
# Temporal Encoder Layers	2
# Temporal Decoder Layers	2
Spatial d_k	256
Temporal d_k	256
# Attention Heads	8
Convolutional Module	
# Convolution Layers	4
Output Filters by Conv	[16, 16, 16, 1]
Kernel Sizes	[5, 5, 5, 7]
Strides	[2, 2, 2, 2]

Table 4.1: Model Hyper-Parameters

As it was mentioned in the previous chapter, the **loss function** that is being minimized is the **Mean Squared Error** (MSE)3.23 between the $(x_{pred},\ y_{pred})$ coordinates predicted by the model and the real $(x_{real},\ y_{real})$. Table 4.2 contains the parameters chosen for the optimizer. We are using the Adam optimizer with 1500 warm-up steps for the learning rate, after which the learning rate starts to decay. The learning rate equation we use is

$$LR(step) = d_k \cdot \min\left\{ \frac{1}{\sqrt{step}}\ ,\ step \cdot \text{warmup steps}^{-1.5} \right\}. \tag{4.1}$$

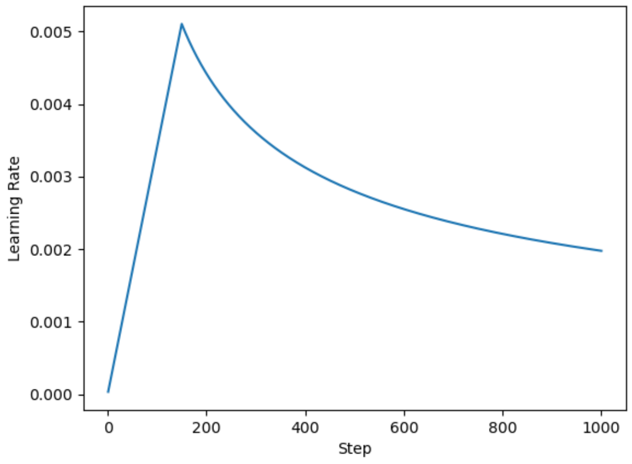

Figure 4.1: Learning rate vs. optimization step.

Figure 4.1 shows the plot of the learning rate vs. the step variable with 1500 warm-up steps and $d_k = 256$. We hope that this plot makes it easier to see the effect of the parameter warm-up steps.

The batch size was set to the small value of 64 because, as the model is being trained in an autoregressive manner, the chain rule needs to be applied multiple times to calculate the gradient, and that consumes a lot of memory resources. Hence, when the batch size becomes too big, the GPU runs out of memory.

Optimizer Parameter	**Value**
Optimizer	Adam
Learning Rate	Custom
Learning Rate Warm-up Steps	1500
β_1	0.99
β_2	0.9
ϵ	$1e-9$
batch size	64

Table 4.2: Optimizer parameters.

The training is carried out in the cluster of CIMAT using a NVIDIA Titan RTX GPU with 24 GB. Each node of the cluster has access to two of those graphic cards; the experiments make full use of them because the training phase is divided between both GPUs to save time. The training process is run during 130 epochs and each epoch of training takes approximately 1 hour to complete.

As it was mentioned at the beginning of the chapter, a few ablation studies are conducted

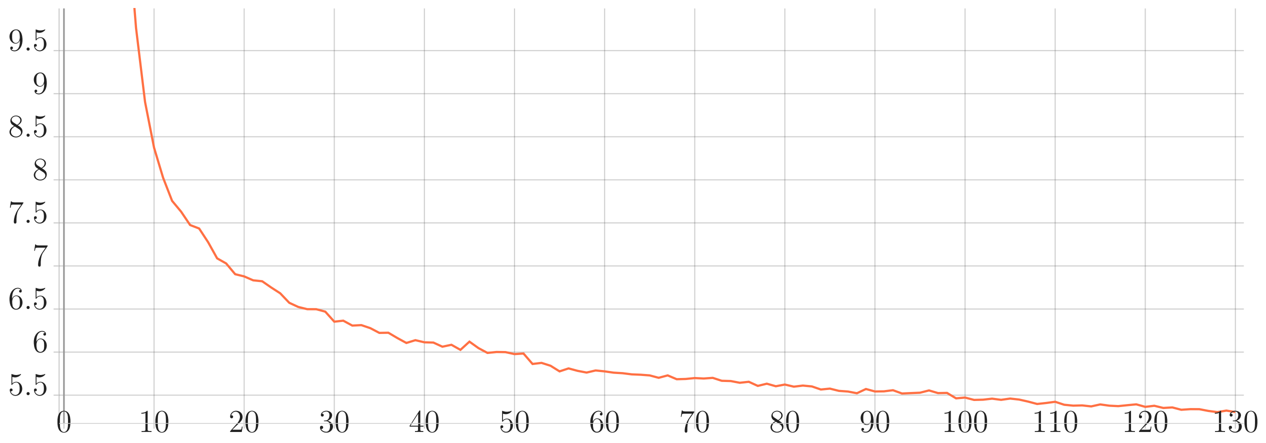

Figure 4.2: ST-Transformer: Train Loss.

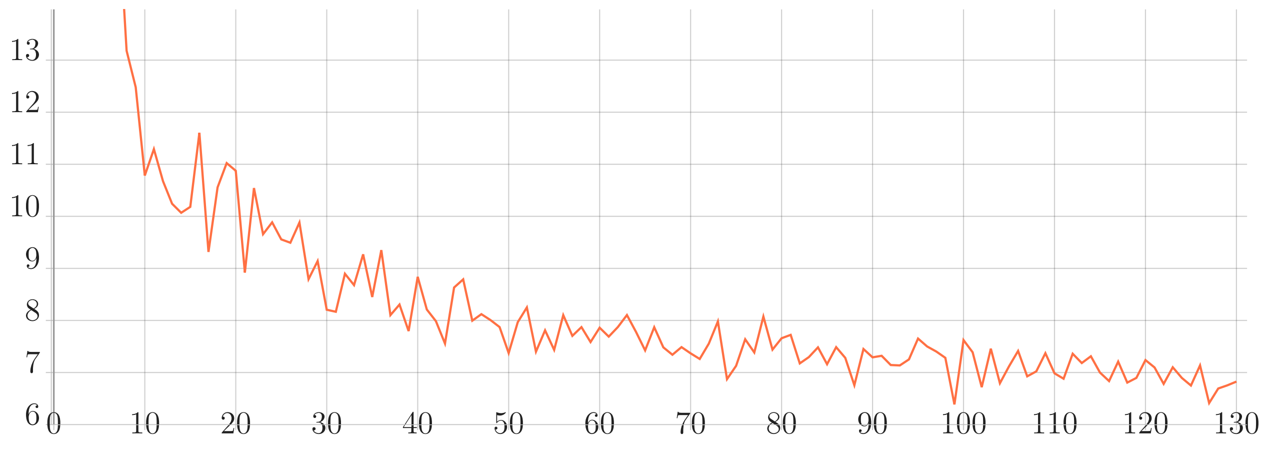

Figure 4.3: ST-Transformer: Eval Loss.

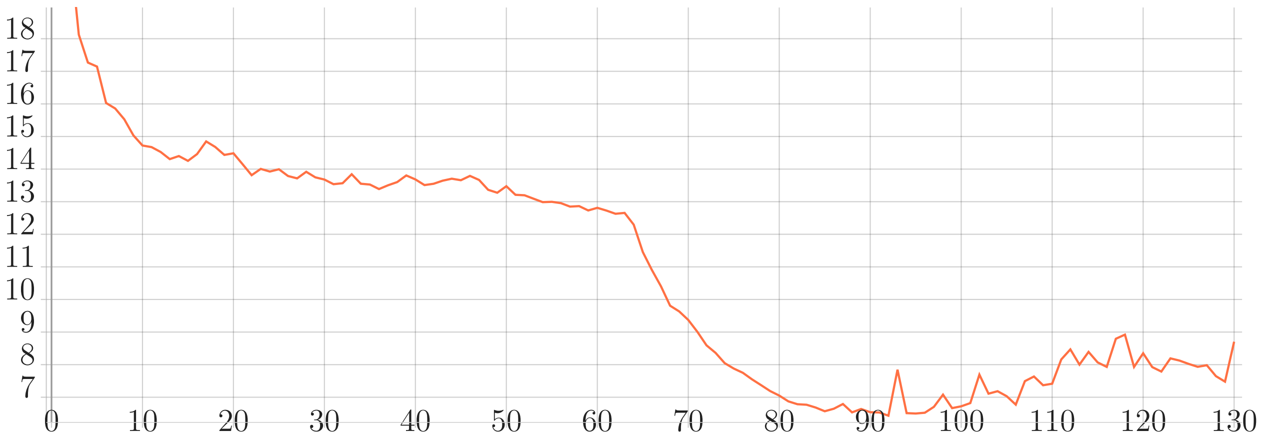

Figure 4.4: ST-Transformer without CNN: Train Loss.

to evaluate how much each part of the model contributes to the final result. We evaluate the performance of three variants of our model:

1. **ST-Transformer**. The main model proposed in this work, described in details in chapter 3.

2. **ST-Transformer without CNN module**. The same model as the ST-Transformer but with the convolutional module to process bitmap information removed, i.e. such that we use only trajectories information.

3. **Time Transformer**. This model is comprised by the time encoder and the time decoder of the ST-Transformer. Basically, it is the same as the model proposed in [13] but trained in an autoregressive manner.

All the models variants are trained with the same number of epochs mentioned above and in the same environment. Figures from 4.2 to 4.9 make reference to the training of the models with the **Shifts dataset** (see Chapter 2). This dataset has been chosen to carry the main training because it contains much more data and allows a better training of the model.

Figures 4.2 and 4.3 show the loss reported by the ST-Transformer model with the training and evaluation datasets, respectively. On the other hand, figures 4.4 to 4.7 show the loss reported by the ST-Transformer without the CNN Module and the Time Transformer alone, with the training and evaluation datasets. As we can see, the Time Transformer alone starts to overfit the data really quickly; from epoch 30 to around epoch 85 the general trend is that the evaluation loss starts to increase while the training loss keeps decreasing, a clear evidence of overfitting.

Figures 4.8 and 4.9 help to visualize the difference on how the Time Transformer alone is overfitting the data, in comparison with the ST-Transformer. This was expected, given the fact that the amount of parameters to adjust for the Time Transformer is much lower and therefore it does not require a great number of iterations.

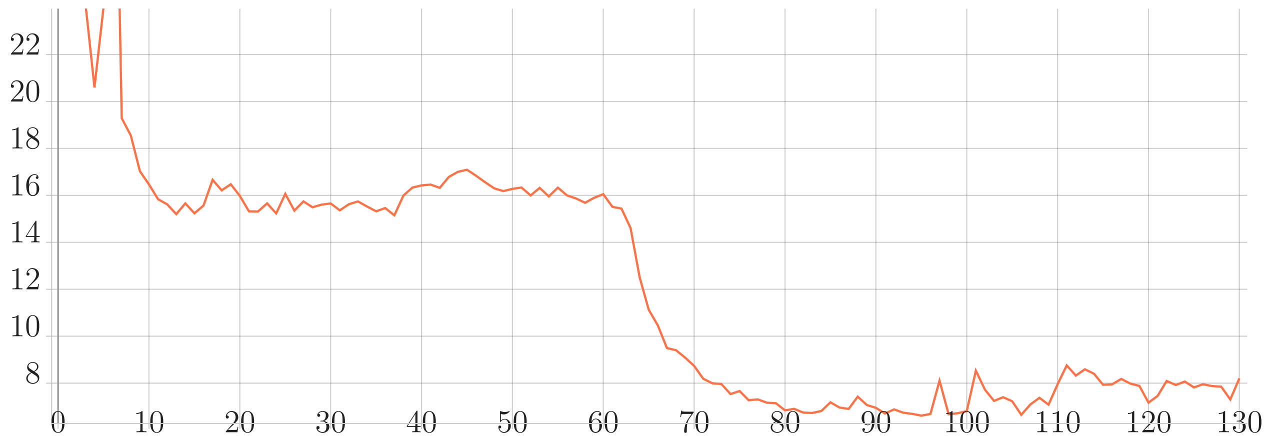

Figure 4.5: ST-Transformer without CNN: Evaluation Loss.

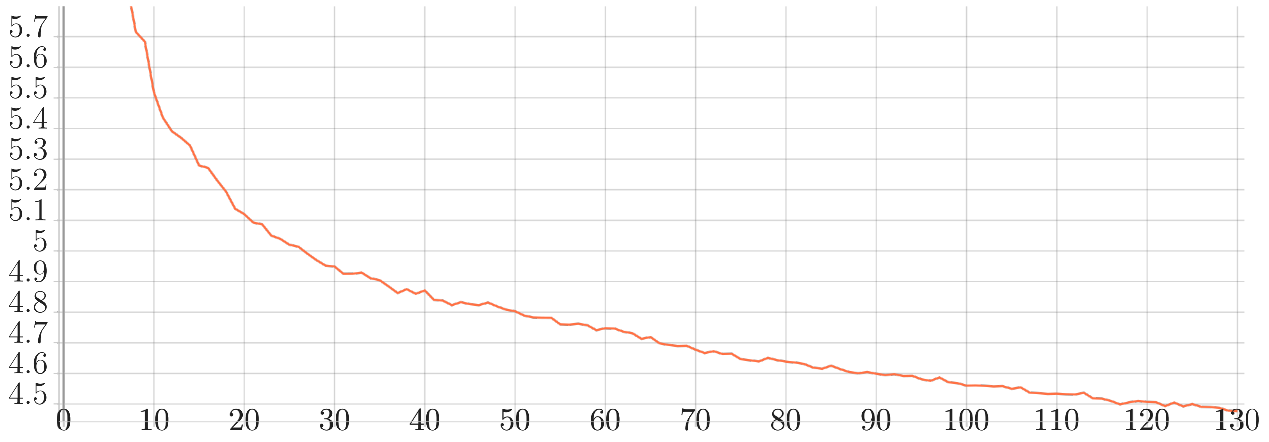

Figure 4.6: Time Transformer: Train Loss.

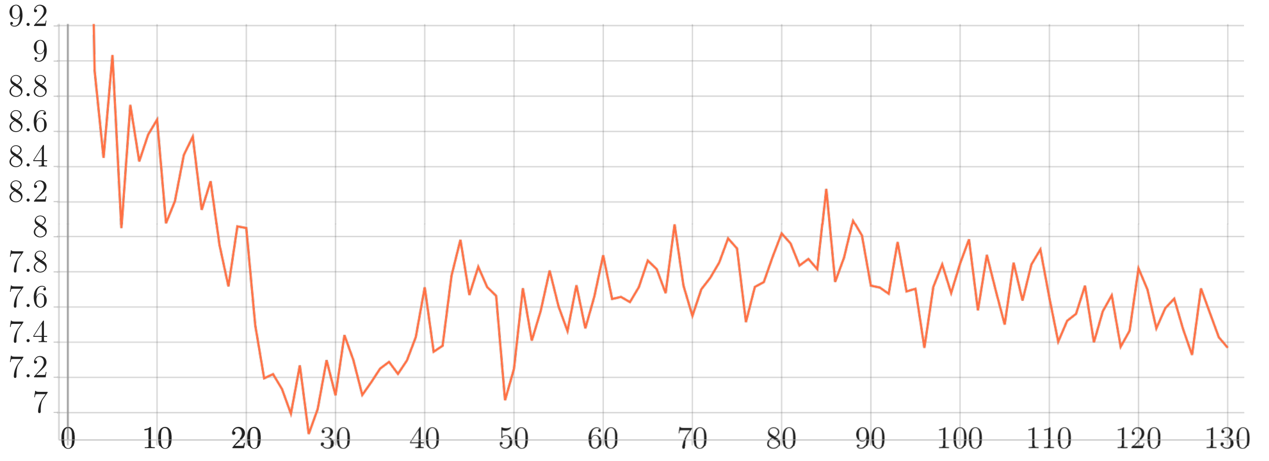

Figure 4.7: Time Transformer: Evaluation Loss.

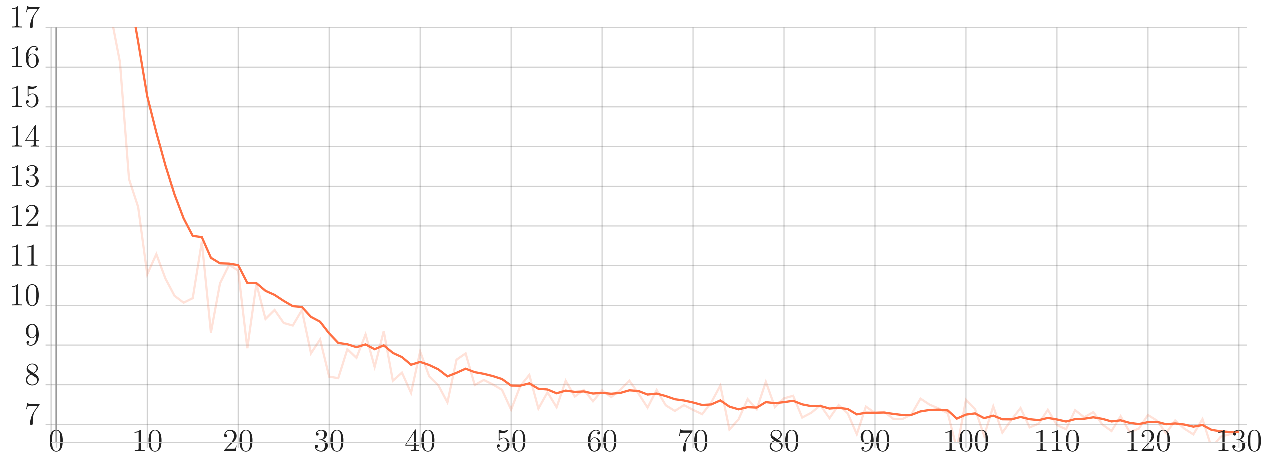

Figure 4.8: ST-Transformer: Smoothed Evaluation Loss.

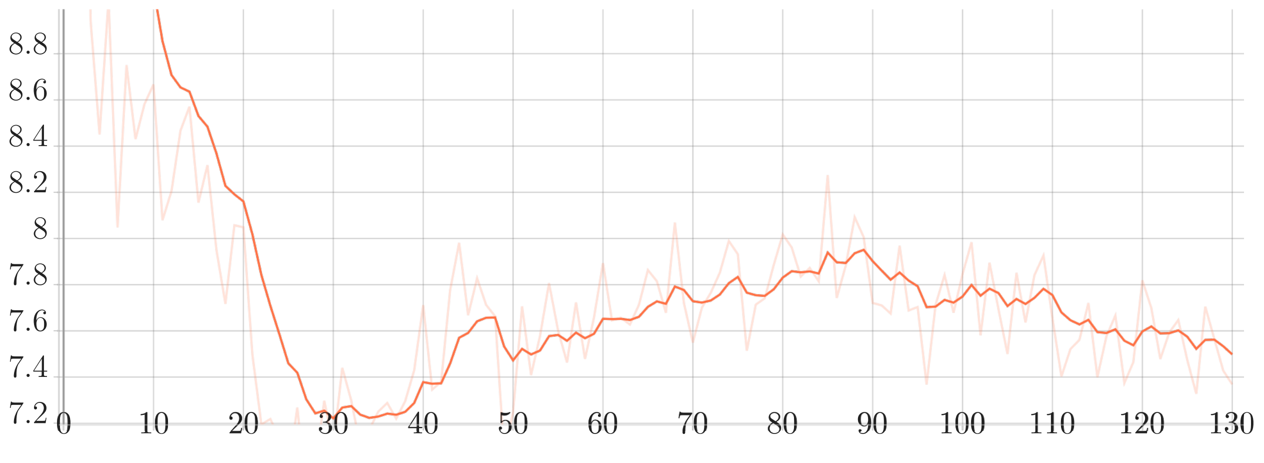

Figure 4.9: Time Transformer: Smoothed Evaluation Loss.

4.2 Quantitative Results

As we have seen in the previous section, the ablation studies we have conducted involve 3 models, one is the ST-Transformer described in Chapter 3, the second model is the same with the CNN module removed, and the third module is just the temporal Transformer. All these models have been trained in an autoregressive manner. In the following, we use the mean average displacement error (mean ADE) and mean final displacement error (mean FDE) as metrics.

Table 4.3 show the results obtained for the Shifts dataset (see Chapter 2). We train the models by taking into account at most **5 neighbors** (the closest ones). It is important to highlight which agents we are considering as neighbors. For this work, we are using only vehicles as neighbors and the vehicles that are not candidates for the trajectory prediction task (the dataset creators decide which is and which is not) are ignored. This was done because it helped to simplify the preprocessing phase, but we are aware that other type of agents (pedestrians, cyclists. . .) or even the ones that are not candidates could also provide useful information.

As the Shifts dataset intended, we use 25 observation points as the past and make the prediction of 25 points into the future. In Table 4.3, we can see that the model proposed in this work performs the best among the three. We can also see that the ST-Transformer without the convolutional module outperforms the Time Transformer.

Model	**Mean ADE**	**Mean FDE**
ST-Transformer	**1.82**	**4.49**
ST-Transformer no CNN	1.93	4.77
Time Transformer	2.12	4.97

Table 4.3: Results reported in meters by each model on Shifts dataset [18].

To be able to get some reference on how the model is performing, table 4.4 shows the results publicly available from the 3 best models on the Shifts challenge according to their website. We can see that there are two new metrics in it, the WADE and the WFDE. WADE stands for Weighted Average Displacement Error and measures the output of the model in a weighted manner. It is important to keep in mind that the shifts challenge has the objective of measuring the uncertainty of a model's output, which should be a distribution rather than a single trajectory. So the weighted part tries to capture the output of the model taking into consideration the probability that the model assigns to a certain trajectory. Likewise, the WFDE is the weighted version of the Final Displacement Error.

Rank	Team	WADE	MIN ADE	WFDE	MIN FDE
1	SBteam	1.85	0.526	4.43	1.02
2	SBteam	1.84	0.526	4.41	1.02
3	Alexey & Dmitry	1.33	0.495	3.16	0.94

Table 4.4: Results reported in Shifts challenge. The results can be checked in the challenge official website `https://research.yandex.com/shifts/vehicle-motion-prediction`

To train the models with the NuScenes dataset (see Chapter 2), we apply transfer learning because the data available in NuScenes for these models is too little. Table 4.5 shows the results obtained for the NuScenes dataset. We have also used at most **5 neighbors**, using 10 observed points from the past, and predicting 12 points into the future. The first row serves as a reference with the state of the art **Interaction Transformer** model proposed in [24]. We can see that, with this dataset, the model that performs the best is the Time Transformer. Despite doing transfer learning, we can see that the model's performance on this dataset is considerably worst. We think that there might be two factors that have an impact on the model. The first factor is the frequency at which data is sampled from both datasets. Let us recall that Shifts samples data at a frequency of 5Hz, in comparison with the 2Hz frequency of NuScenes. Hence, when doing the initial training of the model with the 378,000 inputs from Shifts, we might obtain a model that has learned really specific characteristics from the sampling frequency from Shifts. Some images obtained from the qualitative evaluation also support that this could be the case, because from those images you could see that the last prediction of the model for each trajectory of the NuScenes dataset tended to be much more far away from the ground truth, therefore it might be the case that the model is taking much more little steps for each prediction (because it learned that from the Shifts dataset based on the greater frequency) and the final point predicted for each trajectory is more far away.

The other factor that we think that it might impact on the model, is the fact that NuScenes contains several scenes from Singapore and vehicles drive on the left side of the road. And once again, given that the model was heavily pretrained on Shifts dataset, it could have learned those patterns from the data.

Model	Mean ADE	Mean FDE
Contextual Perception Transformer	**0.76**	**1.25**
ST-Transformer	4.00	8.41
ST-Transformer no CNN	4.81	10.08
Time Transformer	3.84	8.11

Table 4.5: Results reported in meters by each model on NuScenes dataset.

4.2.1 Data Augmentation

Data augmentation is also applied on the NuScenes dataset. Data augmentation in these datasets is relatively easy because one can perform random rotations of the trajectories to get new data. We have performed 4 rotations to each input in the NuScenes dataset in the range of $[\frac{\pi}{4}, \pi]$, so we end up with four times the amount of the original data. Nevertheless, the other models could not be tested with this data augmentation technique because the Time Transformer alone receives velocities rather than positions, and velocities are invariant to rotations. Hence, the data augmentation process would yield the same inputs. Table 4.6 show the metrics obtained with the data augmentation approach.

Model	**Mean ADE**	**Mean FDE**
ST-Transformer	4.00	8.41
ST-Transformer data augmentation	**3.60**	**8.30**

Table 4.6: Results reported in meters on NuScenes dataset with data augmentation.

4.3 Qualitative Results

Quantitative results allow us to evaluate the model in a rigorous way, but a qualitative analysis might give some intuition on what is going under the hood, specially when working with really complex models. In this specific scenario, qualitative results are easy to obtain given the fact that we can plot the predicted trajectory vs. the real trajectory.

Figures 4.10 and 4.11 show the predicted trajectories in red, and the ground truth trajectories, combined with the observed past, in blue. Something that is not easy to see through the quantitative evaluation is that the model might benefit from a multimodality behaviour. With multimodality, we refer to the ability of the model to output not a single trajectory, but a set of trajectories or a distribution of them. It would be expected that this set of trajectories would probably capture the real trajectories on the left middle image and right up corner image in figure 4.10. It is necessary to highlight that the fact that some lines are dashed and other lines are continuous is because the predicted points are closer and therefore they create a “continuous” line. This artifact in the visualization helps to understand that the model is in fact capturing information about the speed at which the agent is moving; an agent that is moving at a greater speed has its sampled points further away from each other, and the model replicates that behaviour. As it was stated at the beginning, this is something we can deduce more directly by observing the trajectories rather than just evaluating the metrics.

Figure 4.11 shows two interesting things. The left top corner shows an agent that is going through the not drivable area. This could be due to that the driver is parking the vehicle. Nevertheless, the model performs a prediction as if the agent goes straight forward, instead of going over the ‘not drivable area’. But in the same figure, we can see, at the bottom images where the model is predicting, that the agents go over the not ‘not drivable area’. This makes us think that the model can still be improved in relation to the bitmaps convolutional layers. Some possible improvement could be to extract more information through the bitmaps.

4.3.1 Spatial Attention

One of the objectives this work was set up to achieve was the exploration and analysis of how Transformers could perform with spatial features as inputs. The visualization of the spatial attention computed by the trained model could help us understand where the model is focusing its attention. Figure 4.12 shows a visualization of the spatial attention. The white rectangle represents the ego-vehicle that is being analyzed at a specific time step and whose spatial attention is being visualized. The blue rectangles represent the neighbors surrounding the ego-vehicle. The dashed green line represents the trajectory of the ego-vehicle. Finally, the red circles surrounding the neighbors represent where the model is focusing its attention at that given time step.

The left top corner image shows something interesting. Despite having other neighbors being closer to the ego-vehicle, they are parked (or that is what the data suggests because they are not moving). But the neighbor that captures the model's attention is the furthest one, but one that is slightly moving and that could interfere directly with the ego-vehicles path. This is something similar to what the work in [1] says about the importance of neighbors to an agent and that the attention should not always be in function of the proximity of those neighbors.

The right top and bottom corner images show how the model is deciding to not pay attention to neighbors that are too far. In the right bottom corner image and the left middle image, it even seems like the model is ignoring the neighbors that do not even share a possible path in common. Let us recall that the bitmaps feature maps are given as an input to the spatial transformer, so the model could have learned in some way to distinguish when two agents have a physical impossibility to cross their trajectories. This needs to be studied more in depth as future work because, in the previous image, we could see a couple of cases in which the model mistakenly predicted trajectories that were physical impossible, or at least such that it would be really difficult that they occur in real world data, given the fact that they did not even resemble to the trajectory a car takes when it is parking.

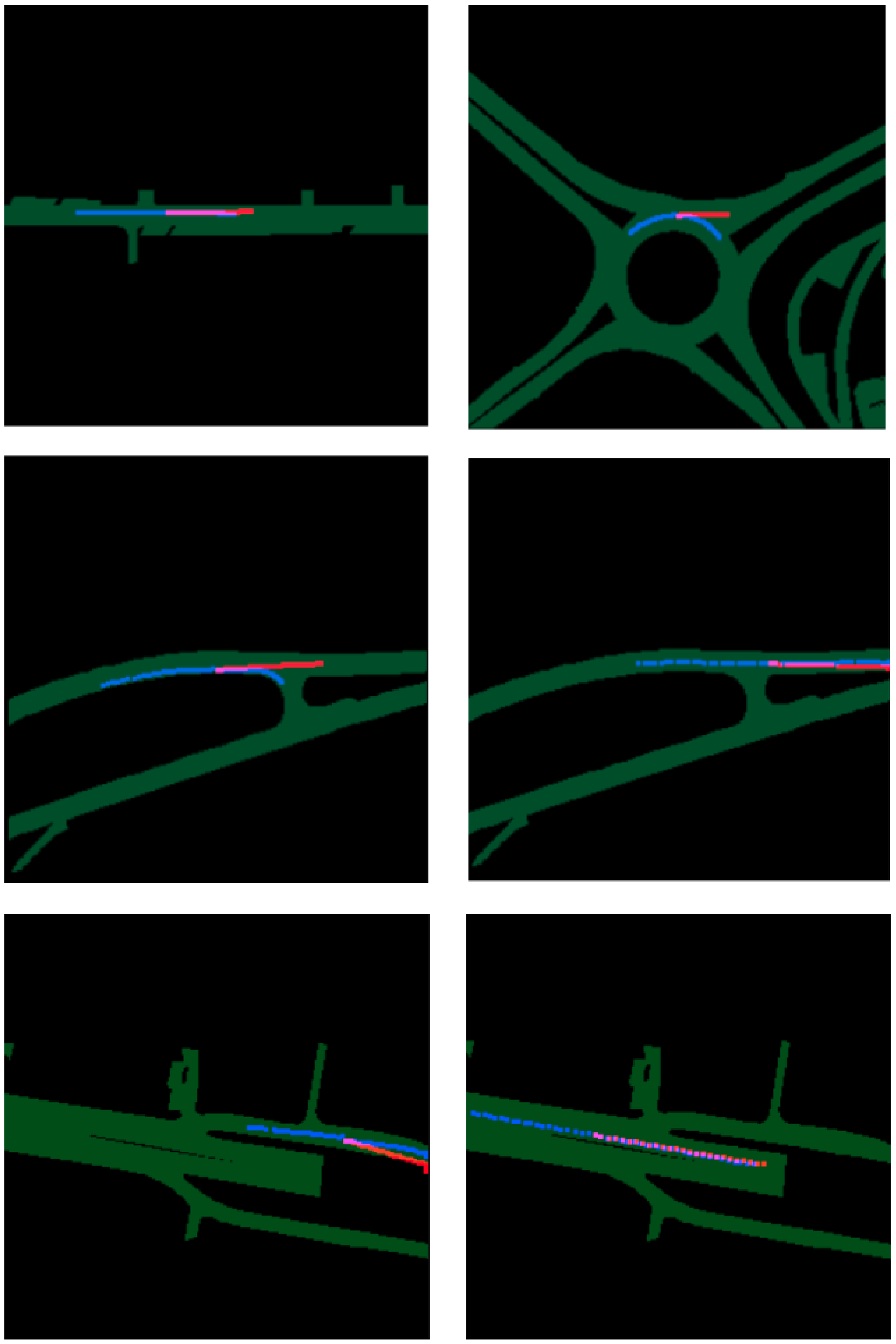

Figure 4.10: Qualitative Results. Multimodal models would be expected to output a trajectory closer to the ground truth trajectories for the left middle image and right up corner image.

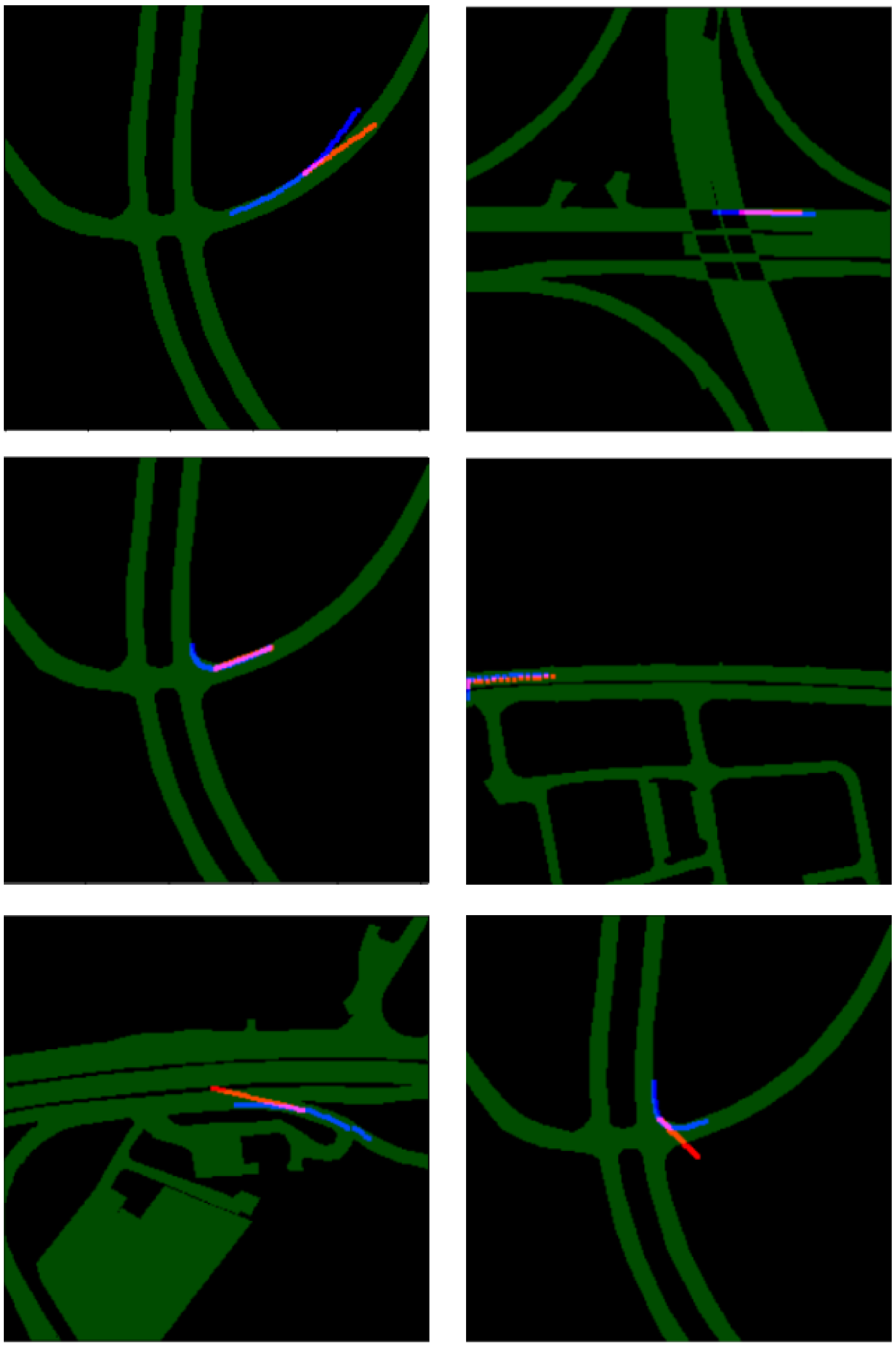
Figure 4.11: Qualitative Results. The bottom images let us see that the bitmaps impact on the overall result can still be improved so the model is able to detect on those cases that the car should 'not go over the not drivable area'.

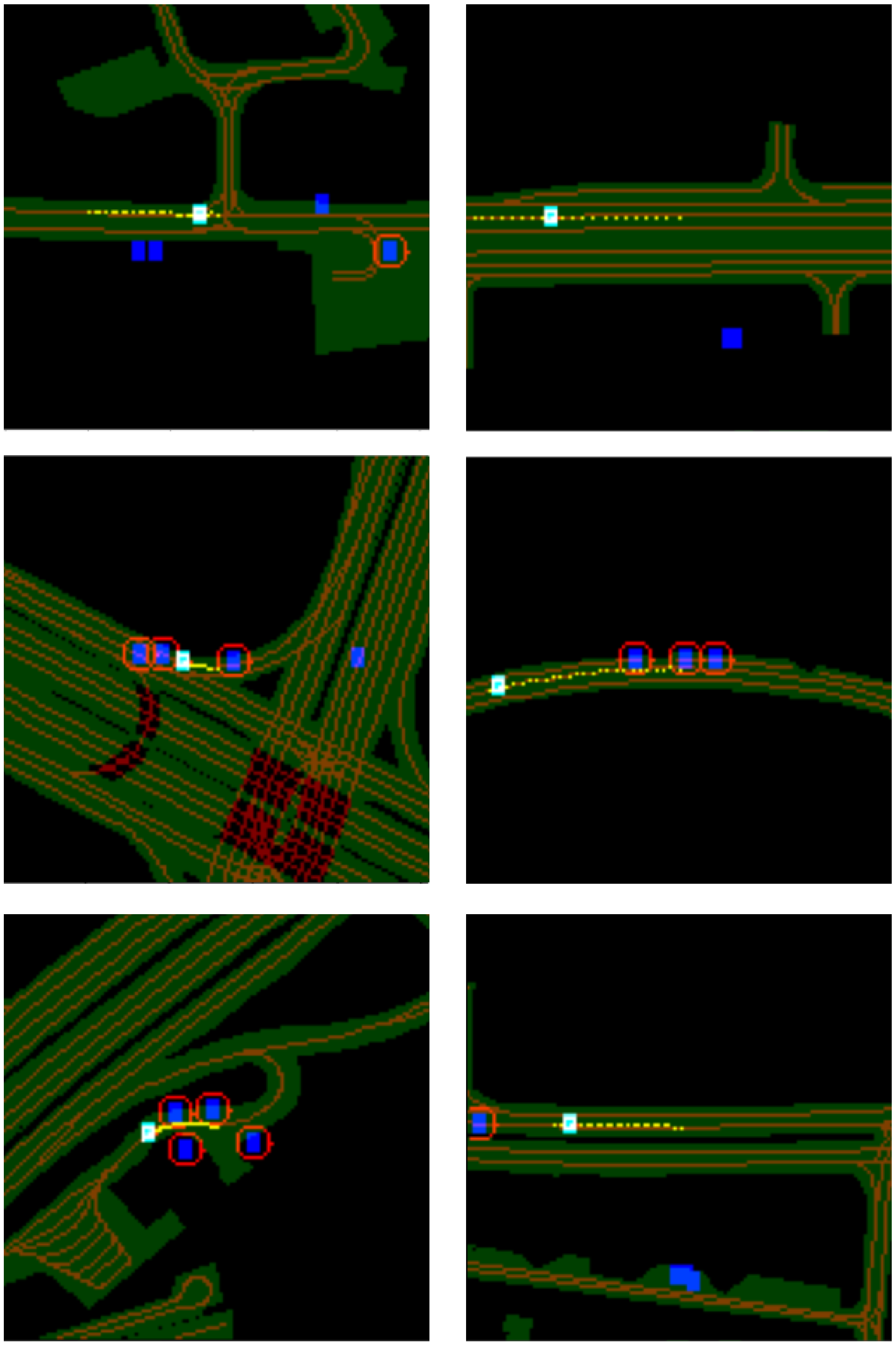

Figure 4.12: Spatial Attention Visualization. The white rectangle represents the ego-vehicle at a specific time step. The dashed green line represents the trajectory of the ego-vehicle. The blue rectangles represent the neighbors surrounding the ego-vehicle and the red circles surrounding the neighbors represent in which neighbors is the model focusing its attention.

Chapter 5

Conclusions and Future Work

Through this work we have discussed how to tackle the problem of trajectory forecasting for autonomous driving systems. The model proposed in this project, the ST-Transformer, explores more in depth a new approach making use of the most innovative models in the field of deep learning. These models require big amounts of data to be trained. Along Chapter 2, we explored and described the two datasets used during the training of the ST-Transformer. We have processed each dataset depending on how the data was stored and we have developed a framework to unify this information and get the inputs for the model. This framework is easily extensible for other datasets, which enables the model to be tested in the future with data that could benefit from spatial attention. Even more, the convolutional module of the model can be deactivated so that it can be trained with datasets that do not contain bitmaps information.

Chapter 3 provides an overview of the ST-Transformer, and it also gives a more in depth description of the Transformer, the core of the model proposed in this work. We reviewed how Transformers attention's works, and how the temporal variable is incorporated into their architecture through the positional encoding. We provided a proof that shows that the generated positional encoding vectors are unique for each time step, which is a desired characteristic of these vectors. The ST-Transformer can be seen as a combination of two Transformer networks along the two dimensions of the problem: the spatial dimension and the temporal dimension. The spatial Transformer of our model is comprised by just the encoder. We hypothesized that the decoder of the model could benefit as well from a spatial attention layer, but that was not the case for this specific work. Probably, with the spatial decoder, there were too many parameters to be adjusted with the data available. It is left as a future work to incorporate the decoder of this module to see if it helps the model to achieve better results.

We carried out this project with an exploratory objective in mind. As it could be seen from the experiments in Chapter 4, the ST-Transformer obtained decent results and despite not competing with the state of art most recent models, we believe that they show that it is worth to invest more resources and time into digging more in depth with the model, given that there is still a lot of room for improvement from different angles. For instance, the optimizer used could have a parameters search to find out if a fine-tuning of them could lead to better results or the model to converge in less time. This is left as a future work, given the fact that the amount of time required to train a single model was about 5 days, and the ablation studies conducted to assess each module of the model were more important from our perspective.

Another thing that the experiments have revealed is that the model could benefit from a multimodality approach. Receiving a distribution or a set of trajectories from the model's output could make the model to get a trajectory that it is closer to the ground truth. Of course, when dealing with distributions, the probability that the model assigns to a certain trajectory is also something to take into consideration. This is a starting point for what could be done next to improve the model.

As a future work, it could also be a good idea to test the model with more neighbors and even with different type of neighbors. As we have stated in the results (see Chapter 4), we made use of only candidate vehicles as neighbors. Other type of neighbors and the ones that are also not candidates for the prediction task could provide useful information as which positions or which trajectories are impossible due to physical implications. The bitmaps fed to the model could also include more information as crosswalks marks, lane direction and related features that could help the convolutional module to provide more information to the spatial encoder. And finally, the feed forward layer of the model should be enabled and test with data augmentation solutions to see if it is able to help the model to improve its performance.

Transformer-based models are difficult to train given the complexity of their architecture. Despite using the most innovative optimizers and schedulers, their training in some cases tends to be unstable and under-performs [14]. Nevertheless, several tasks have shown how powerful they can be when they are correctly trained. Based on this work, we believe that it is worth to keep exploring with more configurations of the model and different training techniques to pursue better results and be able to determine how good are these models for performing at tasks that require spatial attention mechanisms.

Appendix A

Data Type

Scene		
Field	Type	Key
token	string	Primary Key
name	string	
description	string	
log_token	string	Foreign Key (Log table)
nbr_samples	int	
first_sample_token	string	Foreign Key (Sample table)
last_sample_token	string	Foreign Key (Sample table)

Table A.1: Scene Data Types

Instance		
Field	Type	Key
token	string	Primary Key
timestamp	int	
scene_token	string	Foreign Key (Scene table)
next	string	Foreign Key (Sample table)
prev	string	Foreign Key (Sample table)

Table A.2: Sample Data Types

Instance		
Field	Type	Key
token	string	Primary Key
category_token	string	Foreign Key (Category table)
nbr_samples	int	
first_annotation_token	string	Foreign Key (Sample_annotation table)
last_annotation_token	string	Foreign Key (Sample_annotation table)

Table A.3: Instance Data Types

Sample_annotation		
Field	Type	Key
token	string	Primary Key
sample_token	string	Foreign Key (Sample table)
instance_token	string	Foreign Key (Instance table)
attribute_token	string	Foreign Key (Attribute table)
visibility_token	string	Foreign Key (Visibility table)
translation	array[3] float	
size	array[3] float	
rotation	array[4] float	
num_lidar_pts	int	
num_radar_pts	int	
next	string	Foreign Key (Sample_annotation table)
prev	string	Foreign Key (Sample_annotation table)

Table A.4: Sample_annotation Data Types

Sample_data		
Field	Type	Key
token	string	Primary Key
sample_token	string	Foreign Key (Sample table)
ego_pose_token	string	Foreign Key (Ego_pose table)
calibrated_sensor_token	string	Foreign Key (Calibrated_sensor table)
filename	string	
format	string	
width	int	
height	int	
timestamp	int	
is_key_frame	bool	
next	string	Foreign Key (Sample_data table)
prev	string	Foreign Key (Sample_data table)

Table A.5: Sample_data Data Types

Ego_pose		
Field	Type	Key
token	string	Primary Key
translation	array[3] float	
rotation	array[4] float	
timestamp	int	

Table A.6: Ego_pose Data Types

Appendix B

More Attention Visualization

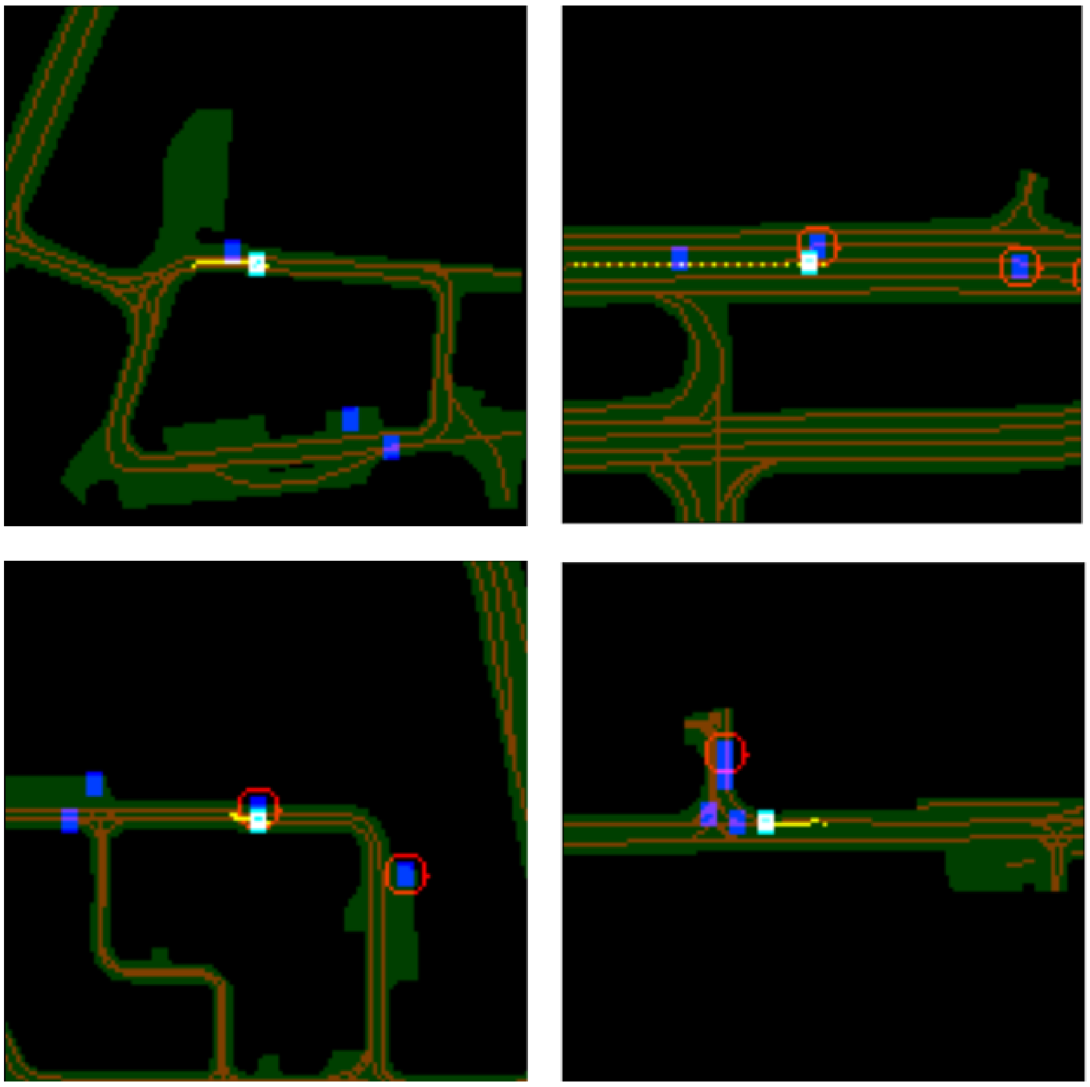

Figure B.1: Attention Visualization.

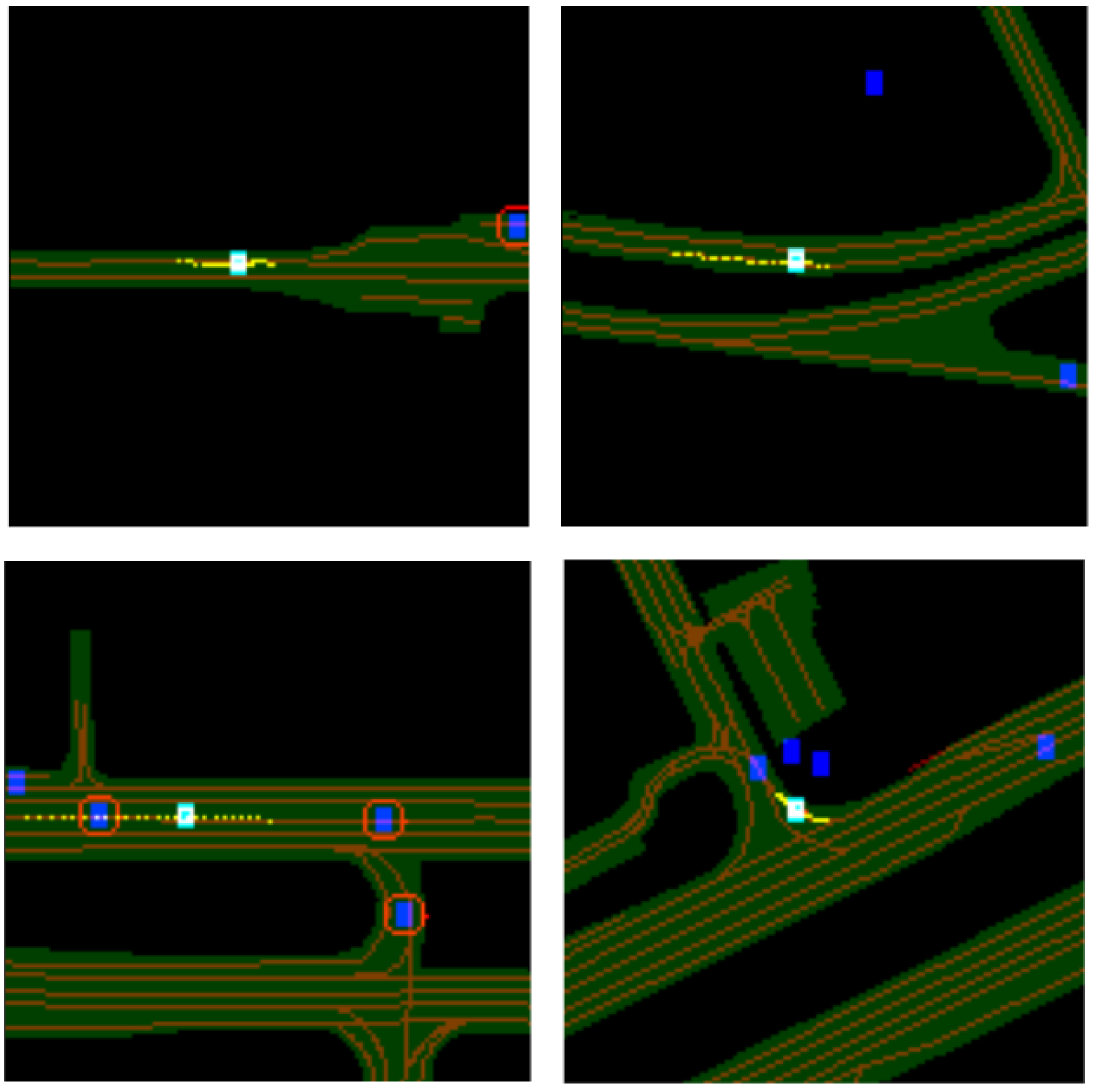

Figure B.2: Attention Visualization.

Bibliography

[1] A. Vemula , K. Muelling , and J. Oh. “Social Attention: Modeling Attention in Human Crowds.” In: *IEEE International Conference on Robotics and Automation (ICRA). arXiv:1710.04689* (2018).

[2] T. Phan-Minh , Elena C. Grigore , Freddy A. Boulton , O. Beijbom , and Eric M. Wolff. “CoverNet: Multimodal Behavior Prediction using Trajectory Sets.” In: *arXiv:1911.10298* (2020).

[3] A. Gupta , J. Johnson , L. Fei-Fei , S. Savarese , A. Alahi. “Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks.” In: *Conference on Computer Vision and Pattern recognition (CVPR). arXiv:1803.10892* (2018).

[4] Dzmitry Bahdanau, K. Cho, Y. Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate.” In: *ICLR. arXiv:1409.0473* (2015).

[5] Marco Pavone Boris Ivanovic. “The Trajectron: Probabilistic Multi-Agent Trajectory Modeling With Dynamic Spatiotemporal Graphs.” In: *IEEE/CVF International Conference on Computer Vision (ICCV). arXiv:1810.05993* (2019).

[6] Holger Caesar et al. “nuScenes: A multimodal dataset for autonomous driving.” In: *arXiv preprint arXiv:1903.11027* (2019).

[7] C Wang , Y. Wang , M. Xu , D. Crandall. “Stepwise Goal-Driven Networks for Trajectory Prediction.” In: *IEEE Robotics and Automation Letters. arXiv:2103.14107* (2022).

[8] T. Mikolov, K. Chen, G. Corrado, J. Dean. “Efficient Estimation of Word Representations in Vector Space.” In: *ICLR. arXiv:1301.3781* (2013).

[9] Y. Yao , E. Atkins , M. Johnson-Roberson , R. Vasudevan , X. Du. “BiTraP: Bi-directional Pedestrian Trajectory Prediction with Multi-modal Goal Estimation.” In: *IEEE Robotics and Automation Letters. arXiv:2007.14558* (2021).

[10] R. Elmasri. *Fundamentals Of Database Systems (6th Ed)*. London: Bantam, 2011.

[11] J. Liang , L. Jiang , Juan C. Niebles , A. Hauptmann , L. Fei-Fei. “Peeking into the Future: Predicting Future Person Activities and Locations in Videos.” In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).

[12] K. Mangalam , H. Girase , S. Agarwal , K. Lee , E. Adeli , J. Malik , A. Gaidon. “It Is Not the Journey but the Destination: Endpoint Conditioned Trajectory Prediction.” In: *European Conference on Computer Vision (ECCV). arXiv:2004.02025* (2020).

[13] F. Giuliari, I. Hasan, M. Cristani, F. Galasso. "Transformer Networks for Trajectory Forecasting." In: *ICPR. arXiv:2003.08111* (2020).

[14] L. Liu , X. Liu , J. Gao , W. Chen , J. Han. "Understanding the Difficulty of Training Transformers." In: *Empirical Methods in Natural Language Processing (EMNLP)* (2020).

[15] Y. Yuan , X. Weng , Y. Ou , K. Kitani. "AgentFormer: Agent-Aware Transformers for Socio-Temporal Multi-Agent Forecasting." In: *arXiv:2103.14023* (2021).

[16] I. Sutskever, O. Vinyals, Quoc V. Le. "Sequence to Sequence Learning with Neural Networks." In: *NIPS. arXiv:1409.3215* (2014).

[17] M. Geva , R. Schuster , J. Berant , O. Levy. "Transformer Feed-Forward Layers Are Key-Value Memories." In: *Empirical Methods in Natural Language Processing (EMNLP)* (2021).

[18] Andrey Malinin et al. "Shifts: A Dataset of Real Distributional Shift Across Multiple Large-Scale Tasks." In: *arXiv preprint arXiv:2107.07455* (2021).

[19] D. Helbing, P. Molnár. "Social force model for pedestrian dynamics." In: *Physical Review E 51, 4282-4286 (1995). arXiv:cond-mat/9805244* (1995).

[20] T. Salzmann , B. Ivanovic , P. Chakravarty , M. Pavone. "Trajectron++: Dynamically-Feasible Trajectory Forecasting With Heterogeneous Data." In: *arXiv:2001.03093* (2021).

[21] J. Amirian , J. Hayet , J. Pettré. "Social Ways: Learning Multi-Modal Distributions of Pedestrian Trajectories with GANs." In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). arXiv:1904.09507* (2019).

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

[23] A. Sadeghian , V. Kosaraju , A. Sadeghian , N. Hirose , S. Hamid Rezatofighi , S. Savarese. "SoPhie: An Attentive GAN for Predicting Paths Compliant to Social and Physical Constraints." In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). arXiv:1806.01482* (2018).

[24] L. Li , B. Yang , M. Liang , W. Zeng , M. Ren , S. Segal , R. Urtasun. "End-to-end Contextual Perception and Prediction with Interaction Transformer." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2021).

[25] A. Vaswani et al. "Attention Is All You Need." In: *Neural Information Processing Systems (NIPS). arXiv:1706.03762* (2017).