Socially-Aware Human Trajectory Forecasting with Spatial Constraints

Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

Computer Science Master's Thesis
Year 2023-2024

Graduand Giacomo Rosin (875724)

Supervisor Prof. Sebastiano Vascon

Assistant Supervisor Rahman Muhammad Rameez Ur

Abstract

Accurate human trajectory forecasting is crucial for various applications, including autonomous vehicles, social robots, and augmented reality systems. However, predicting pedestrian motion is a challenging task due to the complexities of human behavior, including social interactions, scene context, and the multimodal nature of pedestrian trajectories. This thesis focuses on the problem of human trajectory forecasting in crowded scenes using deep learning techniques. The goal is to predict socially and physically plausible future paths for multiple interacting agents in a scene, considering their past trajectories and the scene context. Furthermore, we investigate the effectiveness of a contrastive learning approach to enhance the model's spatial reasoning capabilities to avoid collisions with environmental constraints. Our approach is evaluated through both qualitative and quantitative analysis on established publicly available bird-eye view datasets (e.g., ETH/UCY), as well as an internal first-person view dataset, which is essential for our ultimate goal of integrating the trajectory forecasting model on a robot. To this end, we also describe how to apply models trained on bird's-eye view data to work in first-person view settings, which is essential for integrating the trajectory forecasting model into robotic systems.

Keywords Human Trajectory Forecasting, Future Path Prediction, Pedestrian Motion Prediction, Multi-Agent Trajectory Forecasting, First-Person View Trajectory Forecasting, Contrastive Learning

Acknowledgments

I would like to thank my supervisor, Prof. Sebastiano Vascon, for his help, support, and encouragement throughout my thesis. His knowledge and expertise have been incredibly valuable, and I am grateful for the chance to work with him.
I also want to thank my assistant supervisor, Dr. Rahman Muhammad Rameez Ur, for his support and guidance. His insights and feedback have been essential in developing my work.
My gratitude extends to all the professors who have taught me throughout my academic career. Your dedication to teaching and the knowledge you shared have greatly enriched my academic journey and provided a strong foundation for this research.
To my friends, thank you for your constant encouragement and support. Your friendship has been a source of strength and motivation during this challenging process.
To my family, thank you for your unconditional love, patience, and understanding. Your support has been my anchor throughout this journey.
Finally, I would like to thank everyone else who has supported me, directly or indirectly, during my academic journey. Your contributions have been invaluable.

Thank you.

Contents

1 Introduction **7**
- 1.1 Human Trajectory Forecasting . . . 7
- 1.2 Contributions . . . 8
- 1.3 Outline . . . 9

2 Background **10**
- 2.1 Contrastive Learning . . . 10
- 2.2 Foundational Architectures . . . 11
  - 2.2.1 Recurrent Neural Networks . . . 11
  - 2.2.2 Transformer Encoders . . . 14
  - 2.2.3 Convolutional Neural Networks (CNNs) . . . 16
  - 2.2.4 Autoencoders . . . 17
- 2.3 First-Person View . . . 18
  - 2.3.1 Rationale for Bird's-Eye View in First-Person View Applications . . . 18
  - 2.3.2 Forecasting Pipeline . . . 18
- 2.4 Related Work . . . 25
  - 2.4.1 Multimodal Trajectory Prediction . . . 26
  - 2.4.2 Social context . . . 27
  - 2.4.3 Scene context . . . 28
  - 2.4.4 First Person View . . . 28

3 Methodology **30**
- 3.1 Problem Formulation . . . 30
- 3.2 Model Architecture . . . 31
  - 3.2.1 Trajectory Encoder . . . 32
  - 3.2.2 Social Interaction Module . . . 32
  - 3.2.3 Map Patch Encoder . . . 34
  - 3.2.4 Trajectory Decoder . . . 35
  - 3.2.5 Social-NCE Module . . . 36
  - 3.2.6 Map-NCE Module . . . 37
- 3.3 Loss . . . 39
  - 3.3.1 MSE Loss . . . 39
  - 3.3.2 Environment Collision Loss . . . 40
  - 3.3.3 Social-NCE and Map-NCE Losses . . . 40
- 3.4 Synthetic Dataset . . . 41
- 3.5 Training . . . 42

4 Experiments **46**
4.1 Datasets . . . 46
4.1.1 ETH/UCY . . . 46
4.1.2 Internal Dataset . . . 47
4.2 Evaluation Metrics . . . 48
4.2.1 Average Displacement Error (ADE) . . . 48
4.2.2 Final Displacement Error (FDE) . . . 49
4.2.3 Pedestrian Collisions (COL-PRED, COL-GT) . . . 49
4.2.4 Environment Collisions (ENV-COL) . . . 50
4.3 Experiments . . . 51
4.3.1 Social Interaction Module Ablation . . . 52
4.3.2 Social-NCE Module Ablation . . . 53
4.3.3 Map-NCE Module Ablation . . . 54
4.3.4 Environment Collision Loss Ablation . . . 56
4.3.5 Map Patch Offset Ablation . . . 57
4.3.6 Synthetic Dataset . . . 58
4.3.7 Inference Speed . . . 61

5 Results **63**
5.1 Quantitative Results . . . 63
5.2 Qualitative Results . . . 66
5.2.1 ETH/UCY Datasets . . . 66
5.2.2 Synthetic Dataset . . . 66
5.2.3 Internal First-Person View Dataset . . . 67
5.2.4 Latent Space Exploration . . . 67

6 Conclusions **69**
6.1 Summary . . . 69
6.2 Limitations and Future Work . . . 69

A Hyperparameters **80**

List of Figures

1.1 Example of human trajectory forecasting with social interactions and spatial constraints. There are three pedestrians in the scene, each represented by a different color. The past trajectories of the pedestrians are shown as thin dotted lines, and the future trajectories are shown in a solid line, with an arrow indicating the direction of movement. When walking the pedestrians take into account each other's presence, here represented as the thicker orange dotted lines. Moreover, the pedestrians avoid the obstacles in the scene, here represented as the gray areas. 7

2.1 LSTM scheme [1]. 12

2.2 GRU scheme [1]. 13

2.3 Transformer architecture [2]. 14

2.4 Example of Convolutional Neural Network [3]. 17

2.5 Example of Convolutional Autoencoder [4]. 17

2.6 YOLOv8 detection and tracking latency (s). The plot shows the inference latency of the YOLOv8 model (using BoT-SORT for tracking) at different model sizes and input image resolutions. Larger models and higher resolutions result in longer inference times. Tested on an Intel Core i5-8400 CPU and a NVIDIA GeForce GTX 1050 Ti GPU. Best viewed in color. 19

2.7 Example of YOLOv8 human detection. 20

2.8 Example of human tracking. 21

2.9 Example projection to world coordinates (BEV). The image in the right is the bird's-eye view of the scene, computed by warping the first-person view image using the homography matrix. Superimposed on the BEV image there are the detected human tracks. . . . 22

2.10 Example of multiple trajectory forecasting. The lines in front of each person represent the predicted future trajectories. By zooming in the image, a thin line over the past trajectory of each person is visible, representing the smoothing applied to the trajectories. . . . 24

2.11 Example of multiple occupancy heatmap. The heatmap shows the most likely locations of people in the scene in the next 4.8 seconds. 25

3.1 Architecture of the proposed model. The model consists of a Trajectory Encoder LSTM to capture motion history, a Social Interaction Transformer to model interactions between pedestrians, a Patch Encoder to process the environment map, and a Trajectory Decoder LSTM to predict future trajectories. Additionally, the Social-NCE and Map-NCE modules are used to compute auxiliary contrastive losses during training. The output of the Social Interaction Module, the Map Patch Encoder, and the noise vector are concatenated and used as initial hidden state for the Trajectory Decoder. The two figures on the left represent the input information for the model: the one on the top emphasizes the environment map and the relative patch extraction process around each pedestrian, and the one on the bottom shows just the trajectories of the pedestrians. Both figures show the positive (green) and negative (red for Map-NCE, orange for Social-NCE) samples used in the contrastive learning, that are then processed by the respective NCE modules, as can be seen by the colored arrows. The NCE modules also take as input a query vector; both queries are obtained from the output of the Social Interaction Module, allowing the modules to "see" the motion history, but the Map-NCE query is also concatenated with the output of the Map Patch Encoder, to allow the module to "see" the environment map. The figure also emphasizes the components of the Patch Encoder, which consists of a CNN pretrained through an autoencoder and frozen during the actual training, and a MLP that is instead trained only during the main model training. The Patch Encoder processes all the map patches independently. The colors of the arrows are used to disambiguate the different paths of the information flow in the model. Best viewed in color. 31

3.2 Social Interaction Module. Example of input and output of the Social Interaction Module for a scene with 4 pedestrians. The input to the module is a set of latent representations E of the trajectories of the individuals in the scene. The module performs 4 independent (parallel) forward passes of the transformer, one for each pedestrian, and outputs a set of social encodings S that capture the interactions between the individuals. The vertical lines with dots at the extremes represent the relative positions of each pedestrian with respect to the main pedestrian. The main pedestrian for each forward pass is highlighted with its own color. Best viewed in color. 33

3.3 Example of scenes from the synthetic dataset. Each scene has a size of 25m x 15m, and contains obstacles (black) and pedestrians (red). 42

3.4 Example of learning rate schedule of some training runs. 43

3.5 Example of what the validation losses of some training runs with different performance look like during training. The model reaches a good performance already after a few epochs. Then it slowly improves until it definitively plateaus. In this example, the different performances are due to different model sizes. 45

4.1 Example frames from the ETH/UCY dataset. 46

4.2 Histogram of pedestrian speeds in "eth" (red) vs the rest of the datasets (blue). The "eth" dataset has a higher speed than the other datasets, about 2 times faster. Velocity is expressed as meters every 0.4 seconds. 47
4.3 Example frames from the internal dataset. 48
4.4 Boxplot showing the inference time of the two models. Model1, as expected, clearly outperforms model2 in terms of inference speed, in fact apart from a few outliers, the inference time of model1 is better than all the measured inference times of model2. Moreover, the first and third quartiles (which contain 50% of the data) of both models are quite close, indicating that the inference latency of the two models is quite stable. The same interpretation can be made for the whiskers of the boxplot. 61
4.5 Plot showing the relation between the number of pedestrians in the scene and the inference time of the two models. The latency of model1 (blue) is quite stable across the different scene sizes, while the latency of model2 (orange) grows linearly with the number of pedestrians in the scene. Note that the outliers detected in the previous boxplot 4.4 are filtered out for better visualization. The bands around the lines represent the 95% confidence intervals. Best viewed in color. 62

5.1 Example of generated trajectories on the ETH/UCY zara2 dataset. On the left, the trajectories predicted by model1, on the right, the trajectories predicted by model2. The past trajectories are shown in red, while the future trajectories are shown in green. The segmented environment map is shown in the background, with the obstacles in black. Model1 predicts many trajectories that collide with the obstacles, while model2 predicts trajectories that avoid the obstacles, for example by slowing down. 66
5.2 Example of generated trajectories on the synthetic dataset. On the left, the trajectories predicted by model1, on the right, the trajectories predicted by model2. The past trajectories are shown in red, while the future trajectories are shown in green. The segmented environment map is shown in the background, with the obstacles in black. Model1 predicts many trajectories that collide with the obstacles, while model2 predicts trajectories that avoid the obstacles, for example by slowing down. 67
5.3 Example of generated trajectories on the internal first-person view dataset. On the left, the first-person view image, on the right, the bird's-eye view of the scene. Five predicted trajectories are shown for each person. 67
5.4 Example of predictable scene found by exploring the latent space of the model. The past trajectories are shown in blue, while the future trajectories are shown in red. We can see that this scene may represent the meeting of two people, where one person changes direction to continue walking with the other person. 68

List of Tables

2.1 Model characteristics table. Social: uses social information. Scene: uses scene context information. Multi: predicts multimodal trajectories. FPV: uses first-person view data. 29

4.1 Social Interaction module ablation study. COL-PRED/COL-GT. Lower is better. Comparison between different variants of the Social Interaction module on model1. No: no social interaction module. Attn: attention-based social interaction module. T1, T2, T3: different variants of the transformer-based social interaction module, with 1, 2, 3 layers. 53
4.2 Social-NCE module [5] ablation study. Cell format: upper row Best-of-20 ADE/FDE in meters, lower row COL-PRED/COL-GT. Lower is better. Comparison between model1 without and with the Social-NCE module. 54
4.3 Map-NCE module (3.2.6) ablation study. Cell format: upper row Best-of-20 ADE/FDE in meters, middle row COL-PRED/COL-GT, lower row ENV-COL. Lower is better. Comparison between "No": no scene context, "Map": with scene context, and "M-NCE": with scene context and Map-NCE module. 55
4.4 Environment Collision Loss ablation study. Cell format: upper row Best-of-20 ADE/FDE in meters, middle row COL-PRED/COL-GT, lower row ENV-COL. Lower is better. Comparison between "No": without Environment Collision Loss, and "Yes": with Environment Collision Loss. 57
4.5 Map patch offset ablation. Cell format: upper row Best-of-20 ADE/FDE in meters, middle row COL-PRED/COL-GT, lower row ENV-COL. Lower is better. Comparison between "No": without offset in the scene map patch extraction, and "Yes": with offset in the scene map patch extraction. 58
4.6 Synthetic dataset experiment, with evaluation on "ETH/UCY 2" (sped up) dataset. Cell format: upper row Best-of-20 ADE/FDE in meters, middle row COL-PRED/COL-GT, lower row ENV-COL. Lower is better. Comparison between "No": model2 trained only on the ETH/UCY datasets, and "Yes": model2 pretrained on the synthetic dataset and fine-tuned on the ETH/UCY datasets. . . . 60
4.7 Synthetic dataset experiment, with evaluation on "ETH/UCY" (original) dataset. Cell format: upper row Best-of-20 ADE/FDE in meters, middle row COL-PRED/COL-GT, lower row ENV-COL. Lower is better. Comparison between "No": model2 trained only on the ETH/UCY datasets, and "Yes": model2 pretrained on the synthetic dataset and fine-tuned on the ETH/UCY datasets. . . . 60

4.8 Stronger regularization experiment. Cell format: upper row Best-of-20 ADE/FDE in meters, middle row COL-PRED/COL-GT, lower row ENV-COL. Lower is better. Comparison between "M2": model2 pretrained on synthetic dataset and fine-tuned on the ETH/UCY datasets, and "M2-reg": model2 with stronger regularization (still pretrained on synthetic dataset and fine-tuned on the ETH/UCY datasets). 61

5.1 Model comparison table. The first part of the table shows the Best-of-20 ADE/FDE (meters) results of the previous works, comprising also the state-of-the-art methods. Since the previous works do not report on which version of the dataset they evaluated, comparisons should be taken with caution. The second part of the table shows the results of our models on both versions of the ETH/UCY datasets: "ETH/UCY 2" (the sped up version) and "ETH/UCY" (the original version). Our models report not only the Best-of-20 ADE/FDE (first row), but also the COL-PRED/COL-GT (second row) and the ENV-COL (third row) metrics. The results are reported for 12 future timesteps, given the previous 8. **Bold**: overall best results. Underlined: our best results on "ETH/UCY 2". Overlined: our best results on "ETH/UCY". 65

A.1 Hyperparameters used for the different models. 80

Chapter 1

Introduction

1.1 Human Trajectory Forecasting

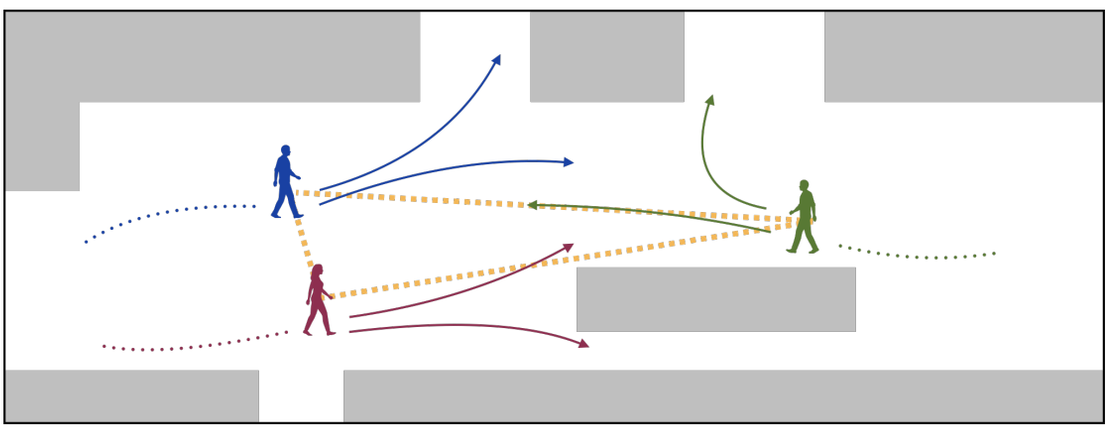

Figure 1.1: Example of human trajectory forecasting with social interactions and spatial constraints. There are three pedestrians in the scene, each represented by a different color. The past trajectories of the pedestrians are shown as thin dotted lines, and the future trajectories are shown in a solid line, with an arrow indicating the direction of movement. When walking the pedestrians take into account each other's presence, here represented as the thicker orange dotted lines. Moreover, the pedestrians avoid the obstacles in the scene, here represented as the gray areas.

Human trajectory forecasting is about predicting where people will walk in the future based on their past movements and interactions. This is important for applications like self-driving cars, social robots, and augmented reality systems. Predicting where people will go helps these systems interact better with humans. For example, self-driving cars need to know where people will walk to avoid hitting them and keep passengers safe [6]. Social robots need to understand human paths to move around people in a way that feels natural [7], and augmented reality systems can enhance the user experience by anticipating human actions and adjusting what they display accordingly [8].

However, forecasting pedestrian paths is difficult because human behavior is complex. Many factors influence how people move, including social interactions, the environment, and individual intent. Additionally, pedestrian trajectories are multimodal, meaning that there can be multiple plausible future paths that a person might take. Social interactions affect how people move, for example in public places, people avoid bumping into each other, walk in groups, and change their speed based on who is around them. The environment also plays a crucial role in how people move. Factors such as obstacles, the width of pathways, and road signs

heavily influence pedestrian paths. Personal goals, like where someone is going and how fast he needs to get there, are important too. For example, someone in a rush will take a direct route even if it means walking through a crowded area, while someone that is not in a hurry might take a longer path. Predicting paths means understanding these goals, which adds complexity. Finally, pedestrian paths are multimodal. For example, to avoid a collision, a person might step left, step right, slow down, or even speed up. A good prediction model needs to consider all these possible paths and generate diverse predictions.

In this thesis, we address the problem of human trajectory forecasting in crowded scenes using deep learning techniques. Our primary objective is to predict socially and physically plausible future paths for multiple interacting agents, taking into consideration their past trajectories and the scene context. To achieve this, we develop an encoder-decoder architecture that captures both social interactions and environmental constraints to generate plausible trajectories. We test our approach using both qualitative and quantitative methods on the ETH/UCY [9, 10] benchmark dataset, which provides bird's-eye view data of pedestrian interactions. We also test our model on an internal first-person view dataset to show how it works in real-world situations, aiming to integrate the model into a robotic system. Moving from bird's-eye view to first-person view is important for real-world applications, especially for robotic navigation in crowds.

1.2 Contributions

Map-NCE A significant contribution of this work is the implementation and evaluation of a contrastive learning approach, which enhances the model's spatial reasoning capabilities to better avoid collisions with environmental constraints, inspired by the Social-NCE method proposed by Liu et al. [5]. This approach introduces an auxiliary contrastive loss that encourages the model to learn where the pedestrian should not go (obstacles), in addition to where they should go (that is already modeled by the standard loss function).

Synthetic dataset with obstacles We analyze the effectiveness of pretraining our model on a synthetic dataset that includes obstacles to improve the model's ability to avoid collisions with environmental constraints.

Environment collision loss In order to facilitate the obstacle avoidance learning, we introduce a variation of the variety loss [11] to penalize the model for generating trajectories that collide with obstacles. This loss encourages the model to generate collision-free paths.

Environment collision metric To validate our approach, we introduce a metric that measures the effectiveness of the model in avoiding collisions with obstacles. This metric provides a quantitative evaluation of the model's obstacle avoidance capabilities, and allows us to compare different models based on their collision avoidance performance.

1.3 Outline

The chapters of this thesis are organized as follows: chapter 2 provides background information for understanding human trajectory forecasting, foundational deep learning architectures used in this work, and a description of the pipeline for integrating this model in a first-person view setting. It also reviews related work, highlighting key techniques with their strengths and limitations. Chapter 3 formalizes the problem of human trajectory forecasting and details the proposed deep learning architecture and contrastive learning approach. Chapter 4 discusses the experimental setup, including the datasets used, the evaluation metrics, and the experimental results. Chapter 5 presents the results of our models, comparing them with the state-of-the-art methods. Finally, chapter 6 concludes the thesis and outlines potential limitations and future works.

Chapter 2

Background

This chapter provides some background information required to understand the rest of the thesis. We start by introducing the concept of contrastive learning, then we present some foundational architectures that are used in the rest of the thesis as building blocks for the proposed model. We also discuss the first-person view setting and the related forecasting pipeline. Finally, we present some related works in the field of human trajectory forecasting.

2.1 Contrastive Learning

Contrastive learning, introduced by Hadsell et al. [12], is a machine learning technique that aims to learn a representation space of the data by contrasting similar and dissimilar pairs of examples. The goal is to bring similar examples closer in the representation space, while pushing dissimilar examples apart. This is achieved by minimizing a contrastive loss, that is a function of the similarity between the pairs.

Contrastive learning is often used in the pretraining stage, using a self-supervised approach, that is, without the need of manually labeled data. The pretraining step allows the model to learn a good representation of the data, that can be used in downstream tasks, sometimes with good results even without fine-tuning, in a zero-shot setting. When the model is fine-tuned for a specific task, the weights of the model start from a better initialization point, which often leads to better results. However, contrastive learning can also be used as an auxiliary loss in the training of a model, to improve the quality of the learned representation. For example, we may know, from domain knowledge, that some properties of the data are important for the task, and we can use a contrastive loss to enforce the model to learn these properties. In general, this can be done by the classical supervised learning approach, too, by adding a loss term that takes into account the desired properties. However, contrastive learning has the ability of learning also from the negative examples.

For contrastive learning to work well, first of all the contrastive task must be related to the downstream task, otherwise it might bring no benefit, or even harm the performance of the model. The contrastive task is related to the downstream task if the learned representation is useful for the downstream task. This is where domain knowledge matters. A bit more information on this can be found at the end of this section. Moreover, the contrastive task must be challenging enough, that is, distinguishing positive from negative examples must be hard. This requires

the model to learn to recognize the fine-grained differences between the examples, and not just rely on simple, easily recognizable features that might not generalize to unseen data. The sampling of good negative examples is called hard negative mining, and its goal is to sample informative negative examples that are close to the positive examples. Several works have shown that the quality of the negative examples is crucial for the success of contrastive learning [13, 14, 15, 16].
In the context of this thesis, a positive pair of examples is composed of a query and a positive key, and a negative pair is composed of a query and a negative key. The query is the example that we want to encode in the representation space, and that will play as an anchor for contrasting the positive and negative keys. The positive key is an example that is positively related to the query, while the negative key is an example that is negatively related to the query.
The most popular contrastive loss is the InfoNCE loss, introduced by Oord et al. [17], that is defined as:

$$\mathcal{L}_{\mathbf{NCE}} = -\log \frac{\exp(\text{sim}(q, k^+)/\tau)}{\sum_{n=0}^{N} \exp(\text{sim}(q, k_n)/\tau)}, \tag{2.1}$$

where q is the embedding of the query, $k_0 = k^+$ is the embedding of the positive key, k_n where $n \in [1, N]$ are the embeddings of the negative keys, $\text{sim}(q, k)$ is a similarity function between the query and the key, and τ is a temperature parameter that controls the smoothness of the distribution. The similarity function is usually the cosine similarity, defined as:

$$\text{sim}(q, k) = \frac{q \cdot k}{\|q\|\|k\|}. \tag{2.2}$$

The InfoNCE loss is basically a binary cross entropy loss for the classification problem of distinguishing the positive key from the negative keys, given the query. Contrastive learning has provable theoretical guarantees, as shown by Oord et al. [17] and Arora et al. [18], where they give an information-theoretic interpretation of the InfoNCE loss, and show that it provides guarantees on the downstream task performance.

$$\mathcal{L}_{\text{down}} \leq \mathcal{L}_{\text{cont}} + \text{BOUND} \tag{2.3}$$

Essentially, the downstream task loss $\mathcal{L}_{\text{down}}$ is upper bounded by the contrastive loss $\mathcal{L}_{\text{cont}}$ plus a term BOUND. This term is complex to characterize, but it depends on the relationship between the contrastive task and the downstream task.
It has been successfully used in several domains, such as computer vision [19, 20, 21], natural language processing [22, 23], and vision-language tasks [24].

2.2 Foundational Architectures

In this section we introduce some foundational architectures that are used in the rest of the thesis as building blocks for the proposed model.

2.2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed for processing sequences of data of arbitrary length. They differ from traditional feedforward neural networks in that they have feedback connections, that

is, connections that loop back on themselves. RNNs maintain a hidden state that depends on the sequence processed so far, that acts as a form of memory. The weights of the network are shared across time steps, allowing the network to be applied to sequences of different lengths.
The hidden state h_t is updated according to the following equation:

$$h_t = f(W_{hh} * h_{t-1} + W_{hi} * x_t + b_h) \tag{2.4}$$

where W_{hh} and W_{hi} are weight matrices, b_h is the bias term, and f is a nonlinear activation function, often the $tanh$ function. The output y_t is produced by applying a linear transformation to the hidden state:

$$y_t = W_{ho} * h_t + b_o \tag{2.5}$$

where W_{ho} is a weight matrix, and b_o is the bias term.
The network is trained using backpropagation through time, which is a variant of the backpropagation algorithm that takes into account the temporal dependencies in the data. The gradients are computed by unrolling the network over time and applying the chain rule to compute the derivatives of the loss with respect to the weights.
One of the main challenges in training RNNs is the problem of vanishing and exploding gradients. This occurs when the gradients become very small or very large as they are propagated back through time, making it difficult to update the weights, in particular for the early time steps in the sequence. This problem can be mitigated by careful initialization of the weights, using gradient clipping, which limits the norm of the gradients, or using more advanced architectures such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) networks, which are designed to better capture long-term dependencies in the data.

2.2.1.1 Long Short-Term Memory Networks (LSTMs)

Long Short-Term Memory (LSTM) networks from Hochreiter and Schmidhuber [25] are a type of RNN designed to deal with the vanishing gradient problem, thanks to the introduction of an additional memory vector, called the cell state, that functions as an information highway that runs through the entire sequence, providing a way for the gradients to flow without vanishing. LSTMs have a more complex architecture than standard RNNs, with three gates that control the flow of information: the input gate, the forget gate, and the output gate.

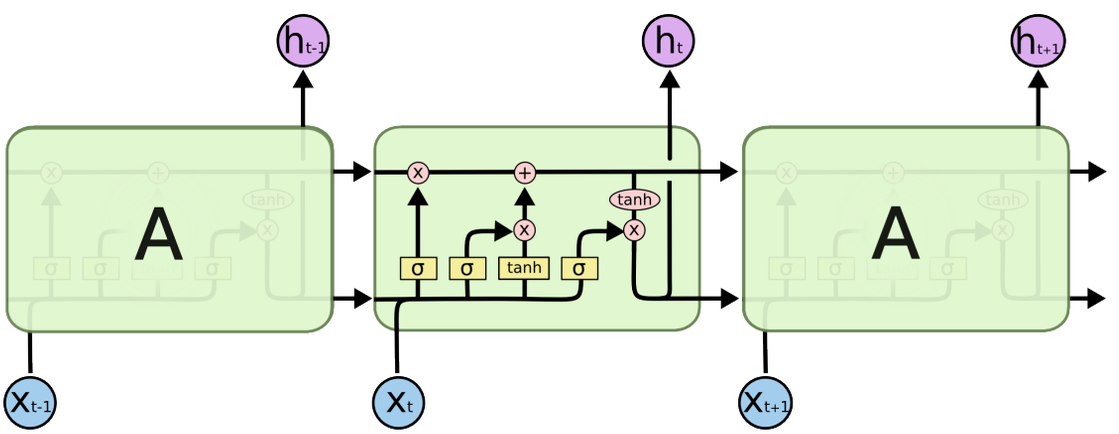

Figure 2.1: LSTM scheme [1].

LSTMs are mathematically defined as follows.

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
\tag{2.6}
$$

where i_t, f_t, o_t are the input, forget, and output gates, c_t is the cell state, h_t is the hidden state, x_t is the input at time step t, W are the weight matrices, b are the bias terms, σ is the sigmoid activation function, $\odot$ is the element-wise multiplication, and tanh is the hyperbolic tangent activation function.
The input gate i_t controls the flow of information into the cell state, the forget gate f_t controls the flow of information out of the cell state, and the output gate o_t controls the flow of information from the cell state to the hidden state. The cell state c_t is updated by adding the input gate times the candidate value, that is the new information that we want to add to the cell state, and by adding the forget gate times the old cell state, that is the information that we want to keep in the cell state.

2.2.1.2 Gated Recurrent Unit Networks (GRUs)

Gated Recurrent Unit (GRU) networks from Cho et al. [26] are a simplification of LSTMs. They merge the input and forget gates into a single update gate, and they merge the cell state and hidden state into a single state. There is also a reset gate that controls how much of the previous hidden state to forget when computing the new hidden state. This makes GRUs use less parameters than LSTMs, and depending on the task complexity, they can perform better or worse than LSTMs.

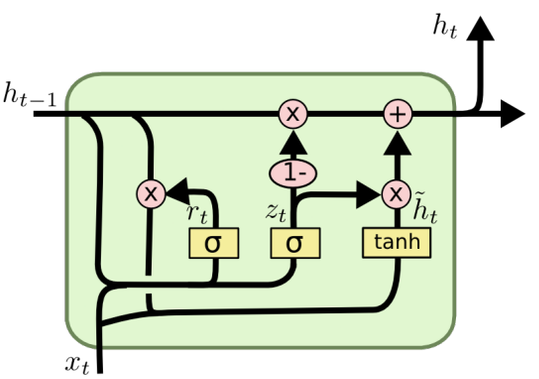

Figure 2.2: GRU scheme [1].

GRUs are mathematically defined as follows.

$$
\begin{aligned}
z_t &= \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\
r_t &= \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\
\tilde{h}_t &= \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{aligned}
\tag{2.7}
$$

where z_t is the update gate, r_t is the reset gate, $\tilde{h}_t$ is the candidate hidden state, h_t is the hidden state, x_t is the input at time step t, W are the weight matrices,

b are the bias terms, σ is the sigmoid activation function, $\odot$ is the element-wise multiplication, and tanh is the hyperbolic tangent activation function.

2.2.2 Transformer Encoders

Transformers are a type of neural network architecture introduced by Vaswani et al. [2] that have been shown to be very effective in various natural language processing tasks, such as language modeling [27], machine translation [2], and also computer vision tasks, such as image classification [28] and image generation [29]. Transformers are composed of two main components: the encoder and the decoder. Since our proposed architecture is based on the transformer encoder, we will focus just on it. The transformer encoder is composed of a stack of N identical layers. Each layer is composed of two sublayers: a multi-head self-attention mechanism and a feedforward neural network. The output of each sublayer is passed through a residual connection and a layer normalization operation, before being passed to the next layer. It is important to note that the transformer encoder outputs as many vectors as the input sequence length, and the order of the vectors is not important, as the self-attention mechanism works on sets of vectors.

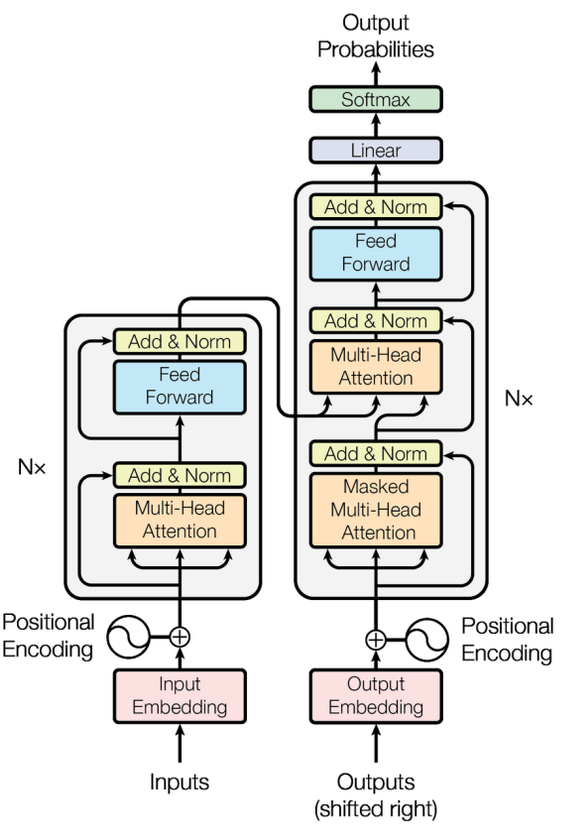

Figure 2.3: Transformer architecture [2].

Self-Attention Mechanism The self-attention mechanism is the core of the transformer architecture. It allows the model to weigh the importance of each element in the input sequence when computing the output representation. For each element in the sequence, the self-attention mechanism computes three vectors: the query vector, the key vector, and the value vector. The query and key vectors are used to compute a compatibility score between each pair of elements in the sequence, and the value vectors are used to compute the output representation.

The compatibility score is computed by taking the scaled dot product between the query and key vectors, and then applying a softmax function to obtain the attention weights. The output representation is then computed as a weighted sum of the value vectors, where the weights are the attention weights.
Mathematically, the self-attention mechanism is defined as follows. Given an input sequence $X = \{x_1, x_2, \ldots, x_n\}$, the query, key, and value vectors are computed as follows:

$$\begin{aligned} Q &= XW_Q \\ K &= XW_K \\ V &= XW_V \end{aligned} \tag{2.8}$$

where W_Q, W_K, and W_V are learnable weight matrices. The output representation is then computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.9}$$

where d_k is the dimension of the key vectors.
The self-attention mechanism can be extended to multi-head attention, where the query, key, and value vectors are projected into multiple subspaces, and the attention mechanism is applied independently to each subspace. The outputs of the different heads are then concatenated and projected into the original space.

Feedforward Neural Network The feedforward neural network is a simple two-layer neural network that applies two linear transformations with a ReLU activation function in between. The feedforward neural network is applied independently to each element in the sequence. The motivation for using a feedforward neural network is to allow the model to perform additional nonlinear computations on the output of the self-attention mechanism.
Mathematically, the feedforward neural network is defined as follows. Given an input sequence $X = \{x_1, x_2, \ldots, x_n\}$, the output of the feedforward neural network is computed as:

$$\text{FFN}(X) = \text{ReLU}(XW_1 + b_1)W_2 + b_2 \tag{2.10}$$

where W_1, W_2 are learnable weight matrices, and b_1, b_2 are bias terms.

Layer Normalization Layer normalization, introduced by Ba et al. [30], is a normalization technique that is applied to the output of each sublayer in the transformer encoder. It normalizes the output of the sublayer across the feature dimension, so that the mean and variance of each feature are close to zero and one, respectively. Layer normalization helps to stabilize the training of the model, by reducing the internal covariate shift, that is the change in the distribution of the input to a layer during training.
Mathematically, layer normalization is defined as follows.

$$\mu^{(l)} = \frac{1}{m}\sum_{i=1}^{m} x_i^{(l)} \tag{2.11}$$

$$\sigma^{(l)} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_i^{(l)} - \mu^{(l)})^2} \tag{2.12}$$

$$\text{LayerNorm}(x^{(l)}) = \gamma^{(l)} \frac{x^{(l)} - \mu^{(l)}}{\sigma^{(l)} + \epsilon} + \beta^{(l)} \tag{2.13}$$

where $x^{(l)}$ is the output of the sublayer, $\mu^{(l)}$ and $\sigma^{(l)}$ are the mean and variance of the output, $\gamma^{(l)}$ and $\beta^{(l)}$ are learnable scale and shift parameters, and ϵ is a small constant to avoid division by zero. The shift and scale parameters are learned during training, and allow the model to undo the normalization if needed.

Residual Connections Residual connections, introduced by He et al. [31], are connections that bypass one or more layers in a neural network. They allow the model to learn the residual, or the difference between the input and the output of the layer, instead of learning the output directly. Residual connections are particularly useful in deep neural networks, where they help to mitigate the vanishing gradient problem, by providing a shortcut for the gradients to flow through the network. The intuition is that a model with more layers should be at least as good as a model with fewer layers, and the residual connections help by providing a way for the layers to learn the identity function, if needed.

Positional Encoding One of the characteristics of the transformer architecture is that it is permutation equivariant, that is, it does not take into account the order of the elements in the sequence. To allow the model to make use of the order of the elements, it is necessary to inject some information about the position of the elements in the sequence. This is done by adding positional encodings to the input embeddings. The positional encodings can be learned during training, or can be fixed. The most common positional encoding is the sinusoidal positional encoding, introduced by Vaswani et al. [2].

2.2.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a type of neural network architecture, introduced by LeCun et al. [32], and popularized by the success in image classification tasks, such as the ImageNet challenge [3, 33, 31]. CNNs are designed to process input data where the spatial relationships between the elements are important, such as images and audio signals. They are composed of a series of convolutional layers, and optionally pooling layers and fully connected layers. The convolutional layers apply a set of filters to the input data that slide over the input and compute the dot product between the filter and the input at each position (convolution operation). The output of a convolutional layer is a feature map that captures the presence of certain patterns in the input, which are learned during training. Each convolutional layer is composed of a set of filters. Going deeper in the network, the filters capture more abstract patterns, and generally the number of filters increases, while the spatial dimensions of the feature maps decrease. This is achieved by using a stride greater than one in the convolution operation, or by using pooling layers that downsample the feature maps by taking the maximum or the average of a set of values. Finally, depending on the task, the output of the convolutional layers can be passed to a sequence of fully connected layers that perform the final computations.

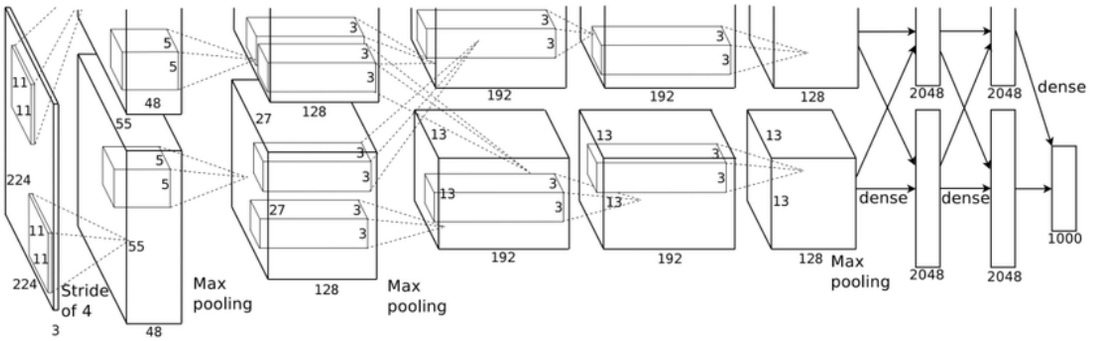

Figure 2.4: Example of Convolutional Neural Network [3].

The power of CNNs comes from the fact that they are able to exploit the local structure of the input data, by sharing weights across the input. This allows the model to learn the same pattern at different positions in the input (translation invariance). Moreover, they learn hierarchical representations of the input data, by composing simple patterns into more abstract patterns.

2.2.4 Autoencoders

Autoencoders are a type of neural network architecture, introduced by Ranzato et al. [34], that are designed to learn a compressed representation of the input data in a self-supervised way. The architecture is composed of an encoder and a decoder. The encoder maps the input data to a lower-dimensional latent space, and the decoder maps the latent representation back to the original input space. The goal of the autoencoder is to learn a representation of the input data that captures the most important features, while discarding the noise. The autoencoder is trained by minimizing the reconstruction error between the input and the output. The key component of the autoencoder is the bottleneck layer, that is the layer in the middle of the network that has a lower dimensionality than the input and output layers. The bottleneck layer forces the model to learn a compressed representation of the input data, that can be used for downstream tasks, such as classification or generation.

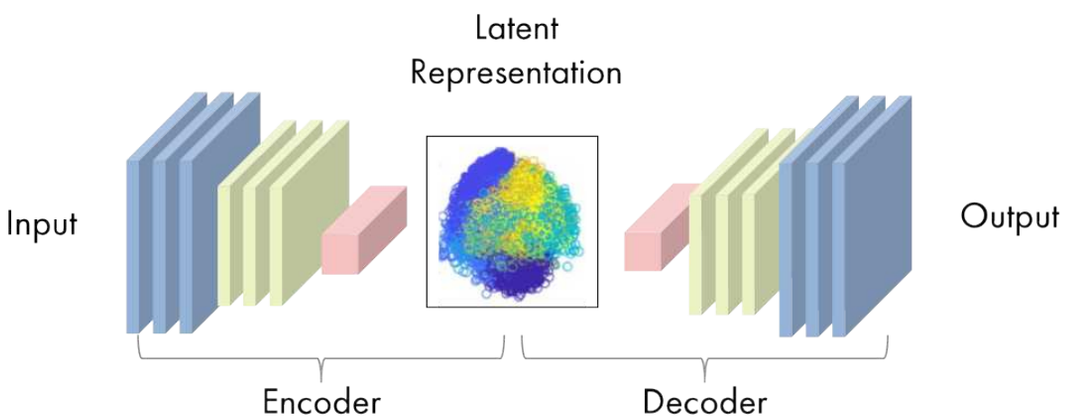

Figure 2.5: Example of Convolutional Autoencoder [4].

Other variants of autoencoders are the variational autoencoders (VAEs) [35], that force the latent representation to follow a prior distribution, usually a Gaussian

distribution, and the conditional variational autoencoders (CVAEs) [36], that condition the latent representation on some additional information, that is forwarded to both the encoder and the decoder.

2.3 First-Person View

2.3.1 Rationale for Bird's-Eye View in First-Person View Applications

While designing our model, we chose to predict trajectories using bird's-eye view (BEV) coordinates, rather than image-plane coordinates, despite the problem being set in a first-person view (FPV) context. This decision was driven by several critical factors.

Availability of datasets There is a significantly larger number of datasets available for training models for human trajectory prediction in bird's-eye view settings compared to first-person view settings.

Benchmarks and comparison Using bird's-eye view coordinates allows us to use existing benchmarks, facilitating direct comparisons with other models. Moreover, having the error in meters gives a better idea of the real world performance of the model, making the results more interpretable. Finally, there are many more models that have been developed and evaluated on BEV data. In contrast, there are fewer models that have been proposed for FPV data.

Scene context utilization Incorporating scene context is crucial for accurate predictions. However, extracting accurate scene information from a single FPV image is challenging. And then, exploiting this information to predict future trajectories in the image plane is even more challenging. On the other hand, if we assume the scene information is available in BEV coordinates, predicting future trajectories becomes more straightforward due to the clearer alignment between the scene information and the trajectories. Of course, this assumption is not entirely realistic, but we should be able to extract the scene map from several FPV frames, using simultaneous localization and mapping (SLAM) techniques, a well-established method that has extensive research support and has been shown to work well in practice. A survey of SLAM techniques can be found in [37, 38].

World coordinates requirement Ultimately, our goal is to deploy the trajectory forecasting module in autonomous agents that operate in the real world. In this setting, the agents need to predict trajectories in world coordinates, with themselves positioned at the origin. So, predicting trajectories directly in world coordinates is more natural and requires fewer transformations.

2.3.2 Forecasting Pipeline

This section describes the pipeline for forecasting human trajectories starting from a first-person view (FPV) video feed, and ending with the prediction of future trajectories in bird's-eye view (BEV) world coordinates.

2.3.2.1 Human Detection and Localization

The first step in the pipeline is to detect and localize humans in the FPV video feed. This is done using a human detection model, which takes as input the video frames and outputs the bounding boxes of the humans in the scene. For this task, we use the YOLOv8 model [39] which is a state-of-the-art real-time object detection model. It can also detect the keypoints of the human body, which will be useful for the next steps in the pipeline. The YOLOv8 model comes in different versions, starting from the smallest version, YOLOv8-n (nano) to the largest version, YOLOv8-x (extra), passing through YOLOv8-s (small), YOLOv8-m (medium), and YOLOv8-l (large). The size of the model, together with the input resolution, affects the inference time and the accuracy of the detection. In our experiments, we use the YOLOv8-m model, at a resolution of 1280x720, which provides a good trade-off between accuracy and inference time. The main effect that we observed by reducing the resolution is that the detection of far-away pedestrians is less accurate, while the closer ones are detected with comparable accuracy. However, that must be decided at deployment time, depending on the requirements, and the hardware available. Figure 2.6 shows the inference time of the YOLOv8 model (including also the tracking time as explained in section 2.3.2.2) at different sizes and resolutions, tested on an Intel Core i5-8400 CPU [40] and a NVIDIA GeForce GTX 1050 Ti GPU [41].

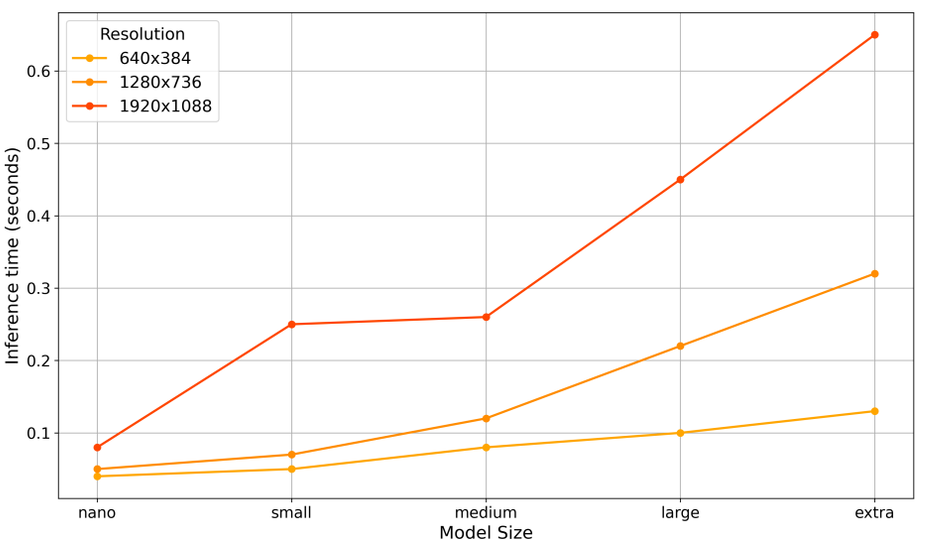

Figure 2.6: YOLOv8 detection and tracking latency (s). The plot shows the inference latency of the YOLOv8 model (using BoT-SORT for tracking) at different model sizes and input image resolutions. Larger models and higher resolutions result in longer inference times. Tested on an Intel Core i5-8400 CPU and a NVIDIA GeForce GTX 1050 Ti GPU. Best viewed in color.

The information about the inference time is important for us, because it tells us how much time we have for the forecasting module to make predictions. A reasonable time for real-time applications could be around 0.4 seconds, which is the standard interval in the literature, and is also the time that we use in our experiments.

YOLOv8 processes each FPV frame to detect humans and provides tight bounding boxes around detected individuals. Each bounding box is associated with a

confidence score indicating the likelihood that the detection is a human. Along with bounding boxes, YOLOv8 also detects keypoints on the human body. These keypoints, such as joints and feet, will be useful for the next steps in the pipeline.

Figure 2.7: Example of YOLOv8 human detection.

Once the bounding boxes are extracted, we need to localize the humans in the image, by extracting their position, that is, the point below their feet. This is done by identifying the point in the middle of the bottom side of the bounding box. This point is assumed to be the point that touches the ground, which is a reasonable assumption since humans are generally standing or walking, not jumping. We also employ some heuristics to filter out bad detections. First, we discard detections with low confidence scores. Then, we discard detections where the bounding box is not touching the ground, which can happen when the feet are occluded. Finally, we discard detections where the feet keypoints are not consistent with the bounding box position. For example, if the feet keypoints are outside the bounding box, or if they are too high with respect to the bounding box. This can happen due to errors in the YOLOv8 detection, or when a person is in a strange pose.

2.3.2.2 Human Tracking

Once people are detected in the video frames, the next step is to track them over time. Tracking means maintaining a unique ID for each detected person as the video advances. This is important because it allows us to follow the movement of each individual throughout the video and keep a history of their positions, which is needed for forecasting their future trajectories.

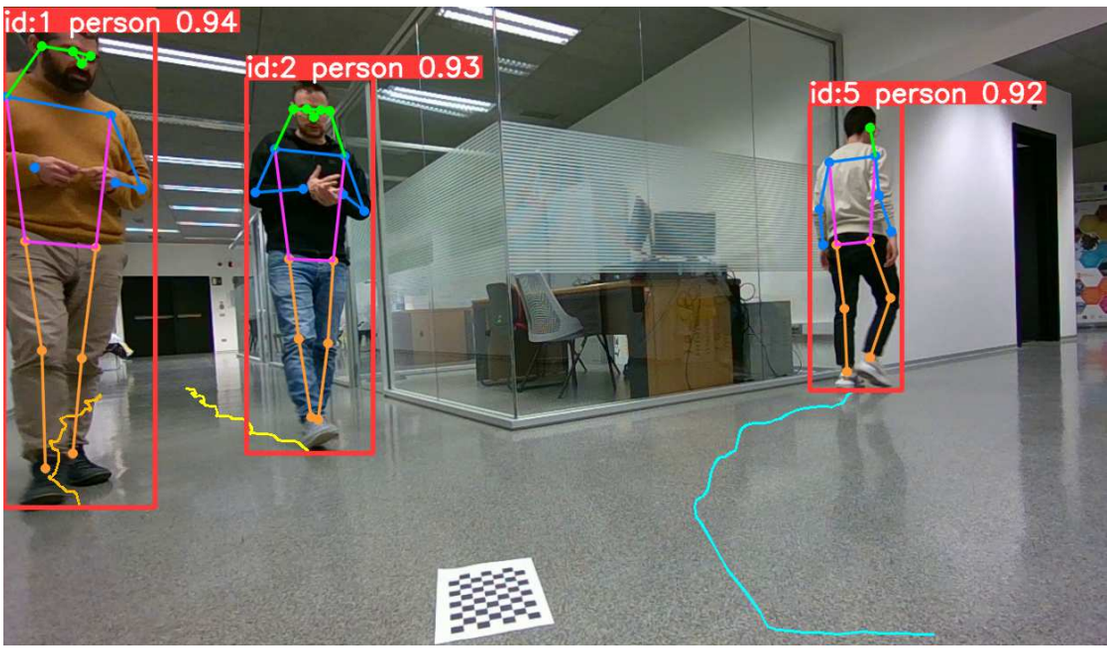

Figure 2.8: Example of human tracking.

For tracking, we use the BoT-SORT algorithm, introduced by Aharon et al. [42], which is a state-of-the-art tracking algorithm. BoT-SORT stands for "Better On-the-fly Tracking using Simple Online and Realtime Tracking". It is designed to work well with real-time applications, making it a good fit for our pipeline. BoT-SORT goal is to associate the new detections provided by YOLOv8 with the existing tracks, maintaining the IDs of the tracked humans. The algorithm uses a motion model to predict the future position of each detected person based on their past positions. This prediction is combined with appearance features extracted from the detected bounding boxes to distinguish between different individuals, even if they are close to each other. BoT-SORT uses the Kalman filter [43] for motion prediction and the Hungarian algorithm [44] for data association. The Kalman filter predicts the next position of each tracked human based on their previous positions and velocities. The Hungarian algorithm matches the predicted positions with the new detections, ensuring that each detection is correctly assigned to an existing track or marked as a new track if it does not match any existing ones. If a person is no longer detected (e.g., because they moved out of the frame), BoT-SORT will eventually remove their ID after a few frames of non-detection.

2.3.2.3 Projection to World Coordinates

After tracking the humans in the video frames, we need to project their positions to world coordinates, also known as bird's-eye view (BEV). To achieve this, we use the homography matrix, which maps points from a plane to another plane. In our case, we map points from the projection of the ground plane in the camera image to the actual ground plane in the world coordinates. The homography matrix is a 3x3 matrix with 8 degrees of freedom, which means it is determined up to scale (i.e., we can multiply all elements by a constant factor and still get the same transformation).

Figure 2.9: Example projection to world coordinates (BEV). The image in the right is the bird's-eye view of the scene, computed by warping the first-person view image using the homography matrix. Superimposed on the BEV image there are the detected human tracks.

The estimation of this matrix requires knowing the correspondence between points in the image and their real-world coordinates. At least four points are needed to compute the homography matrix. A simple way to obtain these points is to position the camera in a fixed location and take a picture of a scene where there are four points with known coordinates on the ground. Some examples could be the corners of a paper sheet or the corners of a football field. To compute the homography matrix, we use the OpenCV library [45], which provides a function to estimate the matrix given the point correspondences. It works by solving a least-squares problem, which minimizes the back-projection error between the real-world points and the image points.

Using the homography matrix H, we can transform any point (x_i, y_i) in the image to world coordinates (x_w, y_w), provided that the point in the image belongs to the ground plane. This transformation is done by first converting the point to homogeneous coordinates (by concatenating a 1 to the vector) and then multiplying it by the homography matrix:

$$\begin{bmatrix} \tilde{x_w} \\ \tilde{y_w} \\ z \end{bmatrix} = H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{2.14}$$

After projecting the points to world coordinates with the homography matrix, the resulting point $(\tilde{x_w}, \tilde{y_w}, z)$ is in homogeneous coordinates, so we need to divide the first two components by the third component to obtain the cartesian coordinates (x_w, y_w).

$$\begin{bmatrix} x_w \\ y_w \end{bmatrix} = \begin{bmatrix} \tilde{x_w}/z \\ \tilde{y_w}/z \end{bmatrix} \tag{2.15}$$

So, given the position of a human in the image, we can use the homography matrix to project it to world coordinates.

In our setup, the camera orientation with respect to the ground is assumed to be fixed. This means that the camera's tilt, pan, and roll angles and its height remain constant, even if the camera itself moves (around the scene). Additionally, we assume the ground is flat. These assumptions allow us to use a precomputed homography matrix for all frames in the video, without the need to recompute it for each frame.

An important point to note is that, before estimating the homography matrix, and also before projecting the humans to world coordinates, we need to calibrate the camera, which involves finding the distortion coefficients, and use them to undistort the image before processing it. This is important to ensure that the homography matrix estimation and the projection is accurate. The distortion is caused by the lens of the camera, which can introduce some warping in the image. The calibration process is done by taking pictures of a calibration pattern (e.g., a chessboard) from different angles and distances, and then using these images to estimate the distortion coefficients. The OpenCV library provides functions to perform camera calibration and undistortion.

2.3.2.4 Data Preprocessing

Before feeding the data to the forecasting model, we need to perform some preprocessing steps to clean the data and make it more suitable for the model. This involves filling missing detections and smoothing the trajectories of the tracked humans to remove noise.

Filling missing detections With a first-person viewpoint, the visibility of the individuals in the scene is limited. People can be occluded by obstacles or other people, or the detection model can fail, for example, due to poor lighting conditions. When a person is not detected in a few frames, the tracking algorithm is generally able to maintain the track, but the position of the person in those frames is missing. To fill these missing detections, we use linear interpolation between the last known position and the next known position. This is a simple method that works well considering that often the gaps are small and that people generally move smoothly. Linear interpolation works by creating a straight line between the two known positions and filling the missing positions along this line.

Smoothing trajectories The trajectories of the tracked humans are often noisy, in particular for far away people. This noise is generated by the detection model, either because the bounding boxes are imprecise or because the person while walking raises their feet, which can cause the bounding box to move up and down. This produces a jittery trajectory that is not representative of the actual movement of the person. Small changes in the position of the bounding box can cause large changes in the position of the projected point in the world coordinates, particularly for far away people. This is due to the perspective projection, which amplifies this error. To mitigate this effect, we smooth the trajectories using a Savitzky-Golay filter [46]. This filter smooths the trajectory by fitting a polynomial to a sliding window of data points. The result is a smoother trajectory that still follows the overall path of the person but without the zigzagging introduced by the noise. The Savitzky-Golay filter works by taking a window of data points and fitting a polynomial of a certain degree to these points. It then replaces the central point in the window with the value of the polynomial at that point. This process is repeated for all points in the trajectory. The degree of the polynomial and the size of the window are parameters that can be adjusted to get the desired level of smoothing. An example of the Savitzky-Golay filter applied to noisy trajectories can be seen in Figure 2.10.
To deal with imprecise detections, we also train the forecasting model with a bit of noise in the data (see data augmentation paragraph in section 3.5), for example, by

perturbing the positions of the detected humans by a small amount. This makes the model more robust to noise in the real data, which should help improve the accuracy of the forecasts.

2.3.2.5 Forecasting

The next step in the pipeline is the actual trajectory forecasting. Given the history of the trajectory of the people in the scene, we want to predict their future positions. To do this, we use our proposed model (see chapter 3). The model takes as input the past trajectories of all the people in the scene, sampled at regular intervals of 0.4 seconds. It accepts up to 8 history points for each person, but we trained it to start predicting with just 2 points, to reduce the latency from the first detection to the forecast. This allows it to start predicting the future trajectory of a person after only 0.4 seconds from the first detection. The model has a prediction horizon of 12 points in the future, which corresponds to 4.8 seconds. We predict 20 samples for each person, which allows us to capture the uncertainty in the forecast. An example of the predictions made by the model can be seen in Figure 2.10, where only 5 out of 20 samples are shown for each person.

The model is suitable for real-time applications, as it can make predictions in about 5-15 milliseconds on a NVIDIA GeForce GTX 1050 Ti GPU [41] (with an Intel Core i5-8400 CPU [40]) for scenes of about 10-30 people. Considering that the detection and tracking steps take about 0.2 seconds, we can easily keep up with the target of 0.4 seconds prediction interval on this hardware. More details about the inference latency of the model can be found in section 4.3.7.

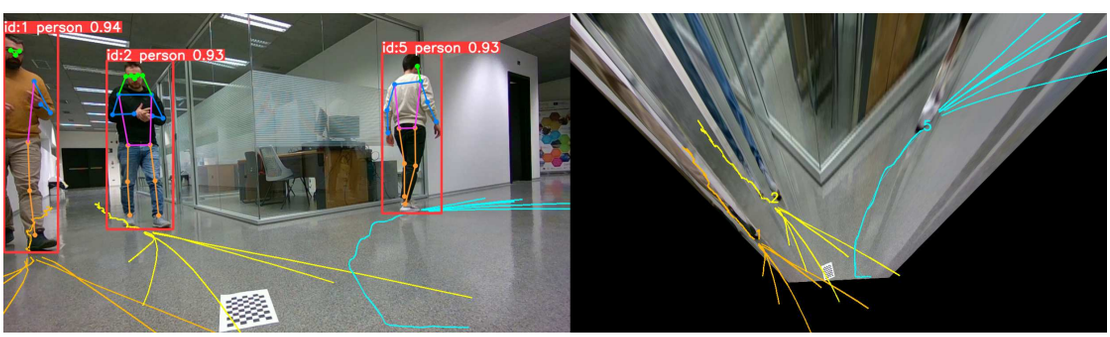

Figure 2.10: Example of multiple trajectory forecasting. The lines in front of each person represent the predicted future trajectories. By zooming in the image, a thin line over the past trajectory of each person is visible, representing the smoothing applied to the trajectories.

2.3.2.6 Occupancy Heatmap

The final step in the pipeline is to generate an occupancy heatmap of the future scene. This map helps in understanding how the space will be used in the near future. This heatmap combines all the predictions of all the people in the scene. Each predicted position contributes to the heatmap, showing where people are likely to be in the next 4.8 seconds ($12 \times 0.4s = 4.8s$). This represents a non-parametric distribution of the occupancy of the scene, giving us a visual and programmatic representation of where humans are expected to be.

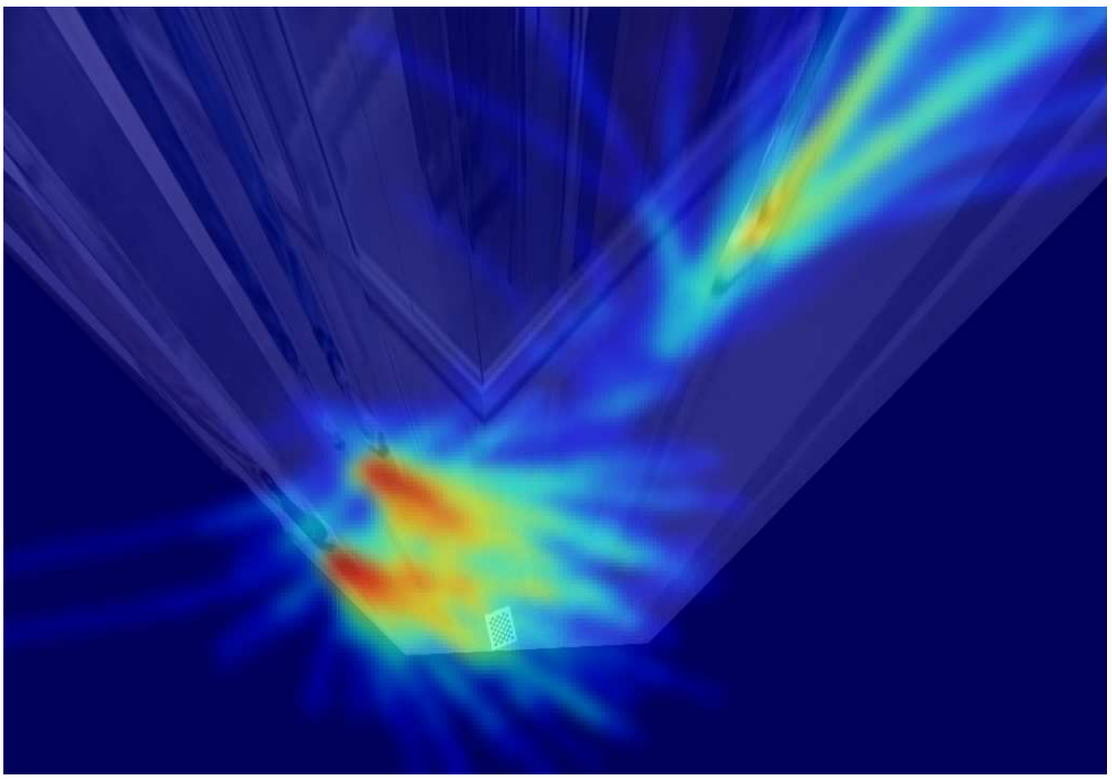

Figure 2.11: Example of multiple occupancy heatmap. The heatmap shows the most likely locations of people in the scene in the next 4.8 seconds.

The occupancy heatmap serves as a layer in between the raw trajectory predictions and final tasks, such as path planning and collision avoidance. For example, a robot can use this map to plan a path that avoids areas where many people are predicted to be, reducing the risk of collisions. We compute the occupancy heatmap by using a Gaussian smoothing kernel. This kernel smooths the predicted positions, making the heatmap more continuous and realistic. Additionally, we account for previous predictions by summing them using a decay factor. This factor reduces the weight of past predictions as we move further into the future, ensuring that more recent predictions have a greater influence on the occupancy map.

2.4 Related Work

Early frameworks for trajectory prediction were based on hand-crafted rules, such as attraction to goals, repulsion from obstacles and other agents. One of the most popular of these is the social force model from Helbing and Molnar [47]. Several works have proposed improvements to this model [48, 49, 50, 51, 52, 53, 54, 55, 56]. Another parallel line of work for simulating pedestrian motion is the reciprocal velocity obstacles (RVO) model from Van den Berg et al. [57], which provides safe motion planning with no oscillation, provided that the agents use the same collision avoidance strategy. Based on the same idea, Van Den Berg et al. [58] proposed an improved and more efficient version, named ORCA. As shown by Kuderer et al. [59], these hand-crafted models perform reasonably on interaction modeling, but their ability to predict the future is poor, due to the limited generalization capabilities of rules hand-crafted by domain experts.
Deep learning is now the standard for trajectory prediction, and several different architectures have been proposed to tackle the problem. Since trajectories are se-

quences, recurrent neural networks (RNNs) are a natural choice, and most works use LSTMs or GRUs as encoders and decoders, predicting the future in an autoregressive way [60, 11, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72]. Notably, Alahi et al. [60] pioneered the use of LSTMs for trajectory prediction, and proposed Social-LSTM, which models the interactions between agents using a social pooling layer that aggregates the hidden states of neighboring agents. Similarly, with the success of Transformers in modeling sequential data, particularly in natural language processing, several works have proposed to use them for trajectory prediction [73, 74]. Franco et al. [73] analyze in detail the use of transformers for trajectory prediction, and propose some models that account only for the past of each pedestrian independently, without considering neither social nor scene context. Liu et al. [74] propose a stack of transformer's cross attention layers that process the history, scene context and social interactions in a sequential way. Yuan et al. [75] propose a model that simultaneously integrates temporal and social information using a novel agent-aware attention. With the recent success of diffusion models [76, 77] in image generation, Gu et al. [78] have proposed to use them for trajectory prediction. These models predict the whole future trajectory at once, instead of autoregressively, and have shown to be pretty accurate. However, they are computationally expensive, due to the sequential denoising process. For this reason Mao et al. [79] have proposed a more efficient version that performs a smart initialization of the noise, allowing it to skip most of the denoising steps. It also improves the diversity of the generated samples, and performs really good, reaching state-of-the-art levels of accuracy. A powerful approach to model trajectory prediction is to use physics-informed models. Salzmann et al. [65] studied integrating dynamics into the prediction model. Similarly Yue et al. [80] proposed NSP-SFM, that introduces a strong inductive bias by combining model-based and model-free methods into a novel Neural Differential Equation framework, that reaches state-of-the-art performance.

2.4.1 Multimodal Trajectory Prediction

Following [66, 11], most works started focusing on the multimodal nature of the problem, that is, there are multiple valid future trajectories for each agent. It is particularly important to predict a distribution of future trajectories, instead of a single one, because it allows downstream tasks, such as deploying autonomous robots, to account for the uncertainty in the prediction. Lee et al. [66] propose DESIRE a framework to predict multiple future paths for each agent, and rank them afterwards. Social GAN, from Gupta et al. [11] uses a GAN [81] to generate multiple realistic future paths, and propose a novel variety loss to encourage diversity in the generated trajectories. Several other works [61, 62, 82, 83, 72] have proposed to use GANs, given their ability to generate realistic samples, by exploiting the adversarial training to push the model to generate samples belonging to the manifold of real trajectories. Kosaraju et al. [62] push this further by utilizing a bicycle GAN [84], that uses a latent encoder to encourage the model to learn a bijection between the latent space and the generated trajectories. Particularly interesting is the work from Dendorfer et al. [83], that proposes to directly predict a probability distribution of goals in an end-to-end trainable way. A subsequent work from the same authors [72] proposes to use a mixture of generators to predict multimodal trajectories, avoiding out-of-distribution samples, and improving the quality of the generated trajectories. Another line of work is to use conditional

variational autoencoders (CVAEs) [36, 35, 34] to predict multimodal trajectories [66, 85, 86, 65, 80]. Trajectron++, by Salzmann et al. [65], uses the CVAE framework by introducing a discrete categorical latent variable that represents the abstract mode of a trajectory. Another approach to multimodal trajectory prediction is to predict a heatmap representing the probability of occupancy of each cell in the scene through a non-parametric distribution. This approach has been chosen by Gilles et al. in [87, 88], where they propose to use a CNN to output the heatmap. Similarly Y-net, from Mangalam et al. [89], predicts a heatmap by using a CNN based U-net architecture [90], conditioned on the past trajectory of the agent and on a sampled goal. Diffusion based models [78, 79] are stochastic by design and can predict a set of future trajectories, by sampling from the noise distribution at each step of the denoising process. However, [79] have shown that the diversity of the generated trajectories can be improved by sampling the noise in a correlated way, instead of sampling it independently. A different framework to achieve multimodality, introduced by [73], is to quantize the future velocity space, and predict the probability of each quantized velocity. In our work, we condition the decoder on a latent Gaussian noise, and train it to predict a diverse set of future trajectories by exploiting the variety loss introduced in [11].

2.4.2 Social context

Processing each pedestrian independently is not enough to predict trajectories that are socially acceptable. In order to account for social interactions, a model must have some way of encoding the social context, that is, sharing information between agents. The strategies that have been proposed to handle social interactions can be divided depending on whether they consider only the neighboring or all agents in the scene, and whether they consider the interactions in a static or dynamic way. The simplest way to account for social interactions is to consider only the neighboring agents. This is the approach taken by most previous works [60, 66, 63, 68, 61, 82, 64, 91]. The most prominent approaches in this category are to either aggregate information from the neighbors in a fixed radius, as in [60, 66, 68, 63, 82, 64], or to consider only the k nearest neighbors, as in [61, 91] and adopting a padding strategy to handle the case when the number of neighbors is less than k. Another way to account for social interactions is to consider all agents in the scene. This is the approach taken by [11, 67, 70, 62, 92]. To handle a variable number of agents, these works use permutation invariant functions that work on sets, such as the social pooling layer from Gupta et al. [11] that aggregates the hidden states of all agents using max pooling, or the social attention mechanism from Kosaraju et al. [62]. The use of attention mechanisms allows it to model the interactions in a dynamic way, by learning the importance of each agent in the scene, depending on the context, like the past motion pattern of the agent. While other approaches, such as the max pooling, model the interactions in a uniform way, by giving the same importance to all agents. Some works [60, 91] prefer to model the social interactions at each time step, allowing to capture in an easier way the evolution of the interactions. However, this approach is computationally expensive, because it requires to compute the interactions at each time step, and it is not always necessary, because the interactions are often stable over time. For this reason, some works [11, 62] prefer to model them only once, at the last observed time step. A great analysis of the possible ways to model social interactions is presented by Kothari et al. [91]. An innovative way to improve the social interactions modeling

and reduce the number of collisions, without adding any inference time overhead, is Social-NCE, a model agnostic framework, proposed by Liu et al. [5], that uses an auxiliary contrastive loss to encourage the model to encode in its hidden state information useful to avoid future collisions with other agents. In our model we choose to use a two layer transformer encoder inserted between the history encoder and the future decoder, that processes the hidden states of all agents in the scene, and learns the importance of each agent in a dynamic way. Moreover, we integrate the Social-NCE framework in our model, as in our experiments we have observed that it reduces the number of collisions.

2.4.3 Scene context

In order to condition the trajectory prediction on the scene context, most previous works [66, 63, 93, 61, 62, 94, 72] have used a pretrained or custom CNN to encode the scene image into a vector. Notably, [61, 62, 94] use the attention mechanism to focus on the relevant parts of the scene. However, compressing the scene image into a one-dimensional space does not yield satisfactory results, as it loses the spatial information and the alignment between the encoded trajectories and the image. This problem is pointed out in [95], which achieved better results without using the images. A solution to this problem is proposed in Y-net by Mangalam et al. [89], which represents the trajectory directly on the same two-dimensional space as the scene image, providing a solution to the alignment issue, and achieving state-of-the-art results. In this work, we propose a different approach that exploits an auxiliary contrastive loss, inspired by Social-NCE [5], combined with other architectural and training changes.
When it comes to providing the network with scene context, most prior works feed the whole scene image [62, 63, 61, 68, 65], while some feed only a cropped region around the agent [66, 82, 96, 72]. Lee et al. [66] proposes to extract a patch from the CNN feature map, in the corresponding position, instead of forwarding just the cropped region. Often the image is also rotated to align the scene with the agent's trajectory, as in [65]. Exploiting our domain knowledge, we prefer to feed the network with a patch of the scene image that is oriented, but also offset, to include the agent's most likely future path. Moreover we train from scratch the CNN that encodes the scene image, using an autoencoder.

2.4.4 First Person View

Some works have focused on forecasting in first person view settings. Yagi et al. [97] use a CNN based model to predict the future trajectory of the people directly in the image plane, taking into account also the ego-motion of the camera. Bi et al. [98] predict the future trajectory in world coordinates, by encoding the past trajectory with a LSTM, and the first-person view images with a CNN. Then combines the two representations and predicts the future trajectory with another LSTM. However, it assumes that each agent has a camera, and the camera stream is available to all agents in real time. Stoler et al. [99] propose a method to build a first person view dataset from third person view data through 3D rendering, and an input correction model, to impute the missing information. Particularly useful, considering that missing information appears frequently in the first person view setting. Qiu et al. [100] predict the future trajectory of the camera wearer, using a graph based approach.

Model	Social	Scene	Multi	FPV
Social Force [47]	✓	X	X	X
Social-LSTM [60]	✓	X	X	X
DESIRE [66]	✓	✓	✓	X
Social-GAN [11]	✓	X	✓	X
SoPhie [61]	✓	✓	✓	X
Social-BiGAT [62]	✓	✓	✓	X
MID [78]	✓	X	✓	X
Leapfrog [79]	✓	X	✓	X
Transformer Networks [73]	X	X	✓	X
Trajectron++ [65]	✓	✓	✓	X
SS-LSTM [63]	✓	✓	X	X
Tensor Fusion [82]	✓	✓	✓	X
Agentformer [75]	✓	✓	✓	X
Stacked Transformers [92]	✓	X	✓	X
STAR [92]	✓	X	✓	X
MG-GAN [72]	✓	✓	✓	X
Y-net [89]	X	✓	✓	X
SMEMO [71]	✓	X	✓	X
NSP-SFM [80]	✓	✓	✓	X
FPL-FPV [97]	X	X	X	✓
FvTraj [98]	✓	✓	✓	✓

Table 2.1: Model characteristics table. Social: uses social information. Scene: uses scene context information. Multi: predicts multimodal trajectories. FPV: uses first-person view data.

Chapter 3

Methodology

3.1 Problem Formulation

The objective of human trajectory forecasting is to predict the future positions of all the pedestrians in a scene. The input to the model is composed of the sequence of all the past positions (2D coordinates) of all the pedestrians in the scene, that we denote as:

$$\mathbf{X} = \{X_1, X_2, \dots, X_N\} \tag{3.1}$$

where N is the total number of pedestrians in the scene, and X_i is the sequence of 2D coordinates of pedestrian i.

$$X_i = [x_{i,t} \in \mathbb{R}^2 \mid t = 1, 2, \dots, T_{\text{obs}}] \tag{3.2}$$

where T_{obs} is the size of the observation window.
Optionally, the model can also take additional information, such as the sequence of bird's eye view images of the scene at each timestep:

$$\mathbf{I} = [I_t \mid t = 1, 2, \dots, T_{\text{obs}}] \tag{3.3}$$

where I_t is the bird's eye view image of the scene at timestep t. In our case, we assume to have access to a segmentation mask of the scene that encodes the walkable area and the obstacles in the scene. Moreover, we use only the last image of the sequence, $I_{T_{\text{obs}}}$.
The task of the model is to forecast the corresponding sequence of future positions of all the pedestrians in the scene:

$$\mathbf{Y} = \{Y_1, Y_2, \dots, Y_N\} \tag{3.4}$$

where Y_i is the future sequence of 2D coordinates of pedestrian i.

$$Y_i = [y_{i,t} \in \mathbb{R}^2 \mid t = T_{\text{obs}} + 1, T_{\text{obs}} + 2, \dots, T_{\text{obs}} + T_{\text{pred}}] \tag{3.5}$$

where T_{pred} is the size of the prediction window.
We denote the model output as:

$$\hat{\mathbf{Y}} = \{\hat{Y}_1, \hat{Y}_2, \dots, \hat{Y}_N\} \tag{3.6}$$

$$\hat{Y}_i = [\hat{y}_{i,t} \in \mathbb{R}^2 \mid t = T_{\text{obs}} + 1, T_{\text{obs}} + 2, \dots, T_{\text{obs}} + T_{\text{pred}}] \tag{3.7}$$

The model output can be extended to predict K future trajectories for each pedestrian, in which case the model output is a set of K sets of N future trajectories:

$$\hat{\mathbf{Y}} = \{\hat{Y}^1, \hat{Y}^2, \dots, \hat{Y}^K\} \tag{3.8}$$

$$\hat{Y}^k = \{\hat{Y}_1^k, \hat{Y}_2^k, \dots, \hat{Y}_N^k\} \tag{3.9}$$

$$\hat{Y}_i^k = [\hat{y}_{i,t}^k \in \mathbb{R}^2 \mid t = T_{\text{obs}} + 1, T_{\text{obs}} + 2, \dots, T_{\text{obs}} + T_{\text{pred}}] \tag{3.10}$$

3.2 Model Architecture

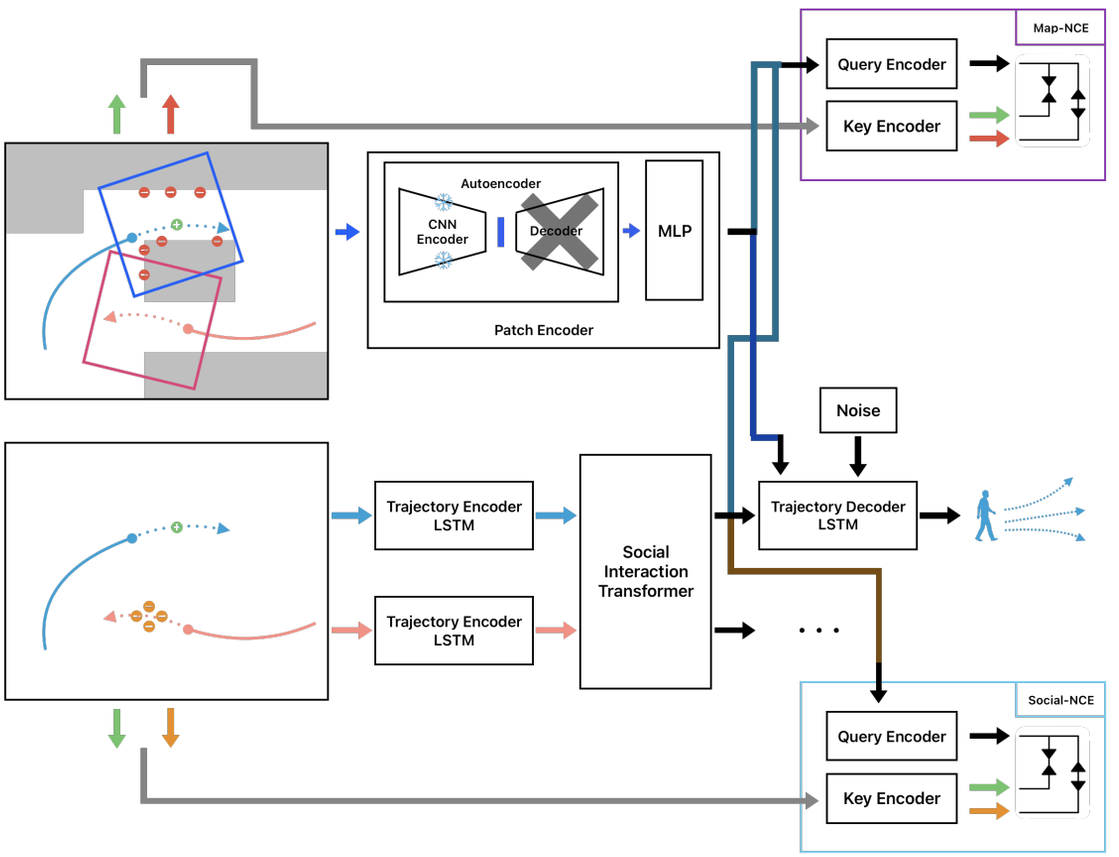

Figure 3.1: Architecture of the proposed model. The model consists of a Trajectory Encoder LSTM to capture motion history, a Social Interaction Transformer to model interactions between pedestrians, a Patch Encoder to process the environment map, and a Trajectory Decoder LSTM to predict future trajectories. Additionally, the Social-NCE and Map-NCE modules are used to compute auxiliary contrastive losses during training. The output of the Social Interaction Module, the Map Patch Encoder, and the noise vector are concatenated and used as initial hidden state for the Trajectory Decoder. The two figures on the left represent the input information for the model: the one on the top emphasizes the environment map and the relative patch extraction process around each pedestrian, and the one on the bottom shows just the trajectories of the pedestrians. Both figures show the positive (green) and negative (red for Map-NCE, orange for Social-NCE) samples used in the contrastive learning, that are then processed by the respective NCE modules, as can be seen by the colored arrows. The NCE modules also take as input a query vector; both queries are obtained from the output of the Social Interaction Module, allowing the modules to "see" the motion history, but the Map-NCE query is also concatenated with the output of the Map Patch Encoder, to allow the module to "see" the environment map. The figure also emphasizes the components of the Patch Encoder, which consists of a CNN pretrained through an autoencoder and frozen during the actual training, and a MLP that is instead trained only during the main model training. The Patch Encoder processes all the map patches independently. The colors of the arrows are used to disambiguate the different paths of the information flow in the model. Best viewed in color.

We propose two models for human trajectory forecasting. The first model, referred to as *Model1*, is designed to predict the future trajectories of all the individuals

in a scene based on their past movements, considering the interactions between them. The model takes as input the observed trajectories of the individuals up to the current time step and generates predictions for their future paths. The output of the model is a set of predicted trajectories for each individual, indicating the likely paths they will take in the upcoming time steps. The second model, referred to as *Model2*, builds upon the architecture of Model1 by incorporating additional contextual information about the scene, in the form of an environment map. This map provides information about the spatial layout of the scene, including obstacles and boundaries, as a binary mask indicating walkable areas. By integrating this map data into the model, Model2 can consider the environmental constraints that may influence human movement. Both models are trained using relative displacements, both as input and output, meaning that the model never sees the absolute positions of the individuals in the scene, but only the relative displacements between consecutive time steps. This allows the model to be agnostic to the absolute position of the individuals in the scene, and focus on capturing the underlying dynamics of their movements (more details in section 3.5). The models are built on top of several modules, some of which are used only at training time to help the model learn the underlying patterns in the data. In the following sections, we describe the modules one by one, and later we will explain how the training process works, including the loss functions used.

3.2.1 Trajectory Encoder

The Trajectory Encoder module processes the observed trajectories of the individuals in the scene, encoding them into a latent representation that captures the relevant information for predicting future movements. The module consists of a LSTM recurrent neural network that processes the trajectory data over time, capturing the temporal dependencies. Each pedestrian's trajectory is encoded independently by a separate instance of the same LSTM model, sharing the same weights. The output of the Trajectory Encoder is a latent representation for each individual (the final hidden state of the LSTM), which is then used as input to the next module. Formally, the Trajectory Encoder module produces the encoding e_i for each individual i in the scene, by processing its observed trajectory X_i with the LSTM model:

$$e_i = \text{LSTM}_{\text{enc}}(X_i; W_{\text{enc}}) \tag{3.11}$$

where W_{enc} are the parameters of the LSTM model. So, the output of the Trajectory Encoder module is a set of latent representations of the trajectories of the individuals in the scene:

$$E = \{e_1, e_2, \ldots, e_N\} \tag{3.12}$$

3.2.2 Social Interaction Module

The Social Interaction Module takes as input the latent representations e_i of the individuals' trajectories produced by the Trajectory Encoder, and processes them to capture the interactions between the individuals in the scene. The module consists of a two-layer transformer encoder that processes the latent representations

of the individuals, attending to the information from other pedestrians to model the social interactions.
The input trajectory embeddings E alone do not contain information about the spatial arrangement of pedestrians in the scene, since the trajectory encoder only sees relative displacements with respect to the previous time step (of a single pedestrian). To incorporate this spatial context, we add to the latent representation of the trajectories E, an embedding of the relative positions of the pedestrians with respect to each other, at the last observed time step T_{obs}. This is, in some way, similar to the positional encoding used in the transformer architecture (when used to encode sequences), but specific to the relative positions of the pedestrians in the scene. This allows the transformer input to contain information about the spatial positions of the individuals.

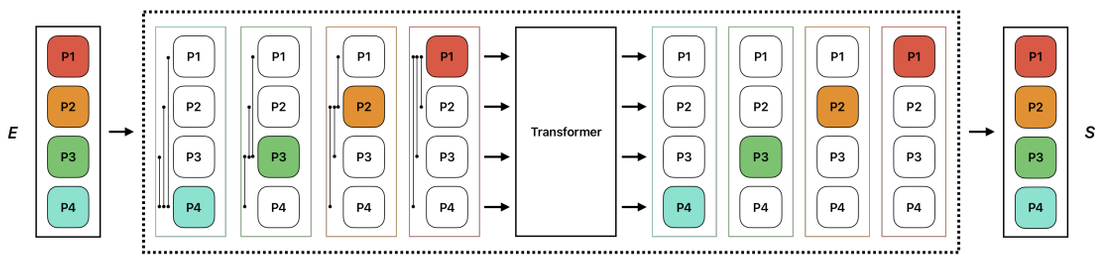

Figure 3.2: Social Interaction Module. Example of input and output of the Social Interaction Module for a scene with 4 pedestrians. The input to the module is a set of latent representations E of the trajectories of the individuals in the scene. The module performs 4 independent (parallel) forward passes of the transformer, one for each pedestrian, and outputs a set of social encodings S that capture the interactions between the individuals. The vertical lines with dots at the extremes represent the relative positions of each pedestrian with respect to the main pedestrian. The main pedestrian for each forward pass is highlighted with its own color. Best viewed in color.

Each pedestrian has its own relative position with respect to all the others, that is, we need to perform a forward pass of the transformer for each pedestrian in the scene, and each time add to the latent representation of the trajectories the embedding of the relative positions of that pedestrian with respect to the others (that changes for each pedestrian). For each forward pass, we take the output of the transformer corresponding to the main pedestrian, and use it as input to the next module. An illustration of the input and output of the Social Interaction Module for a scene with 4 pedestrians is shown in figure 3.2.
Formally, to compute the social encoding s_i for the individual i, we first compute the relative positions R_i of the individual with respect to all the N people in the scene, at the last observed time step T_{obs}:

$$R_i = \{r_{i,0}, r_{i,1}, \ldots, r_{i,N}\} \tag{3.13}$$

$$r_{i,j} = x_{i,T_{\text{obs}}} - x_{j,T_{\text{obs}}} \tag{3.14}$$

where $x_{i,T_{\text{obs}}} \in \mathbb{R}^2$ is the position of pedestrian i at the last observed time step T_{obs}, and $j \in \{1, 2, \ldots, N\}$ is an index that ranges over all the pedestrians in the scene. Note that, when $i = j$, the relative position is $\overrightarrow{0}$.
Then, we compute the relative position embeddings R'_i for the individual i:

$$R'_i = \{r'_{i,0}, r'_{i,1}, \ldots, r'_{i,N}\} \tag{3.15}$$

$$r'_i = \text{MLP}_{\text{rel\_pos}}(r_i; W_{\text{rel\_pos}}) \tag{3.16}$$

Then, we mix the trajectory embeddings E and the relative position embeddings R'_i to obtain the input E'_i to the transformer encoder for the individual i:

$$E'_i = \{e'_{i,0}, e'_{i,1}, \ldots, e'_{i,N}\} \tag{3.17}$$

$$e'_{i,j} = e_j + r'_{i,j} \tag{3.18}$$

Finally, we compute the social encoding s_i for the individual i by processing the input E'_i with the transformer encoder, and taking the output corresponding to the current pedestrian i:

$$S_i = \text{Transformer}_{\text{soc}}(E'_i; W_{\text{soc}}) \tag{3.19}$$

$$s_i = S_i[i] \tag{3.20}$$

where W_{soc} are the parameters of the transformer encoder. The output of the Social Interaction Module is a set of social encodings s_i for each individual in the scene:

$$S = \{s_1, s_2, \ldots, s_N\} \tag{3.21}$$

The key advantages of using a transformer architecture in the Social Interaction Module are that it allows for an arbitrary number of neighboring pedestrians to be considered in the interactions, and also the attention mechanism allows the model to dynamically adjust the importance of each pedestrian's information based on the context, making the model more flexible and capable of capturing complex social dynamics. The downsides of this approach are the quadratic complexity in the number of pedestrians and the potential overfitting to spurious correlations between distant pedestrians.

3.2.3 Map Patch Encoder

The Map Patch Encoder module gets as input a patch of the environment map centered around each pedestrian, and encodes it into a latent representation that captures the spatial structure of the scene around the individual. The extracted patches have a fixed size of 100x100 pixels, that corresponds to a physical area of 10x10 meters in the real world, so each pixel corresponds to a 10x10 cm area. More precisely, each patch is oriented according to the pedestrian's heading direction, and is not centered around the pedestrian, but is offset in order to provide a larger forward view of the scene (9 meters) and a smaller backward view (1 meter), with equal side views (5 meters). This way, the model can take into account the pedestrian's field of view and the spatial context in which the individual is moving. If the pedestrian is near the edge of the visible scene, the extracted patch will extend beyond the boundaries of the map, where the ground truth is unknown. In such cases, the unknown portion of the patch is assumed to be walkable area, which is the most reasonable assumption. The Map Patch Encoder module is composed of two submodules: a convolutional neural network (CNN) that processes the map patch and extracts the spatial features, and a two layer multi-layer perceptron (MLP) with ReLU activations that processes the output of the CNN and produces a latent representation useful for predicting the future trajectories. The CNN is trained in a self-supervised manner, using an autoencoder on a randomly generated

set of map patches, in a separate training phase from the main model training. The autoencoder is trained to embed the map patch into a fixed-size vector, and then reconstruct the original map patch from the compressed representation. This allows the CNN to learn to compress the spatial information of the map patch into a latent representation that preserves the spatial structure of the scene. The CNN encoder maps the input patch of size 100x100 pixels to a latent representation of size 64, and the decoder maps it back to the original size through a series of convolutional layers with pixel shuffle [101] upsampling layers in between. When training the main model, the decoder part of the autoencoder is discarded, and the encoder is frozen. So at training time, only the MLP part of the Map Patch Encoder is trained, while the CNN is fixed.
Formally, the Map Patch Encoder module produces the encoding m_i for the map patch p_i of each individual i in the scene:

$$m_i = \text{MLP}(\text{CNN}(p_i; W_{\text{cnn}}); W_{\text{mlp}}) \tag{3.22}$$

where W_{cnn} are the parameters of the CNN model, and W_{mlp} are the parameters of the MLP model. So, the output of the Map Patch Encoder module is a set of latent representations of the map patches of the individuals in the scene:

$$M = \{m_1, m_2, \ldots, m_N\} \tag{3.23}$$

The Map Patch Encoder module is used only in Model2, as Model1 does not consider the environment map in the prediction process.

3.2.4 Trajectory Decoder

The Trajectory Decoder module consists of a LSTM recurrent neural network that generates autoregressive predictions of the future trajectory of the individual, conditioned by initializing the hidden state with the output of the Social Interaction Module. The module generates the future trajectory of each individual independently, using a separate instance of the same LSTM model for each pedestrian. To account for the multimodal nature of human behavior, the model is trained to predict multiple possible future trajectories for each individual, capturing the uncertainty in the predictions. This is achieved by conditioning the module on a random noise vector sampled from a Gaussian distribution, and training the model using the variety loss proposed by Gupta et al. [11] to encourage diversity in the predicted trajectories.
Formally, the Trajectory Decoder module produces the predicted trajectory $\hat{Y}_i^k$ for each individual i in the scene and each sample k, by autoregressively generating the future trajectory based on the social encoding s_i, the map patch encoding m_i, and the random noise vector $\epsilon_k \sim \mathcal{N}(0, I)$ (that is sampled once for each sample k).

$$\hat{Y}_i^k = \text{LSTM}_{\text{dec}}([s_i, m_i, \epsilon_k]; W_{\text{dec}}) \tag{3.24}$$

where $[s_i, m_i, \epsilon_k]$ is the initial hidden state of the LSTM model, and W_{dec} are the parameters of the LSTM model. So, the final output of the Trajectory Decoder module are K sets of N predicted trajectories, denoted by $\hat{Y}$ as previously defined in equations 3.8, 3.9, and 3.10, repeated here for convenience:

$$\hat{\mathbf{Y}} = \{\hat{Y}^1, \hat{Y}^2, \ldots, \hat{Y}^K\} \tag{3.8}$$

$$\hat{Y}^k = \{\hat{Y}_1^k, \hat{Y}_2^k, \ldots, \hat{Y}_N^k\} \tag{3.9}$$

$$\hat{Y}_i^k = [\hat{y}_{i,t}^k \in \mathbb{R}^2 \mid t = T_{\text{obs}} + 1, T_{\text{obs}} + 2, \ldots, T_{\text{obs}} + T_{\text{pred}}] \tag{3.10}$$

The Trajectory Decoder module is used in both Model1 and Model2, but the map patch encoding m_i is used only in Model2.

3.2.5 Social-NCE Module

The Social-NCE module, introduced by Liu et al. [5], is a contrastive learning based module, that aims to ensure that the learned social representations encapsulate the necessary information for identifying scenarios that could lead to collisions with other pedestrians. The need for this module arises from the fact that the training dataset contains only positive examples of trajectories, that is, examples of trajectories that did not lead to collisions. To learn a robust representation capable of generalizing to unseen scenarios, the model must be capable of distinguishing between a pedestrian's true future trajectory and potential discomfort zones around other pedestrians' future paths. The Social-NCE module consists of two submodules: a query encoder and a key encoder. The query encoder embeds the output of the Social Interaction Module (section 3.2.2) through a linear projection head, producing a query vector for each pedestrian. The key encoder embeds the positive and negative samples through a MLP, producing a set of key vectors for each pedestrian. Queries and keys are embedded in a shared space, so that the similarity between them can be computed using the dot product. The goal is to maximize the similarity between the query and the key corresponding to the true future trajectory, while minimizing the similarity between the query and the keys corresponding to the negative samples.
The procedure to obtain the positive and negative samples for pedestrian i is as follows. Given a future timestep t, the positive sample is the true future position of the pedestrian i at timestep t, with a small Gaussian noise to prevent overfitting, denoted by:

$$\text{pos\_sample}_i = y_i^t + \epsilon \tag{3.25}$$

where $\epsilon \sim \mathcal{N}(0, c_\epsilon I)$, and c_ϵ is a small constant, set to 0.05 meters in our experiments. The negative samples are obtained starting from the true future positions of all the other pedestrians in the scene:

$$\{y_j^t \mid j \in \{1, 2, \ldots, N\} \wedge j \neq i\} \tag{3.26}$$

For each other pedestrian j, we generate a set denoted $\text{neg\_samples}_{i,j}$ of 8 negative samples around him, by adding a small displacement Δ_p to the true future position y_j^t, to simulate a discomfort zones around the pedestrian:

$$\text{neg\_samples}_{i,j} = y_j^t + \Delta_p + \epsilon \tag{3.27}$$

$$\Delta_p = (\rho \cos \theta_p, \rho \sin \theta_p) \tag{3.28}$$

$$\theta_p = \frac{\pi}{4} p \quad \forall p \in \{0, 1, \ldots, 7\} \tag{3.29}$$

where ρ is a small constant, set to 0.5 meters in our experiments, and p is an index that ranges from 0 to 7, corresponding to the 8 displacement directions around the pedestrian. Again, a small Gaussian noise ϵ is added to the negative samples to prevent overfitting.
Finally the set of negative samples for pedestrian i is the union of the negative samples around all the other pedestrians in the scene:

$$\text{neg\_samples}_i = \bigcup_{j \neq i} \text{neg\_samples}_{i,j} \tag{3.30}$$

In order to perform the contrastive learning, the Social-NCE module defines a query encoder that embeds the social encoding s_i of each individual i in the scene into a query vector q_i:

$$q_i = \text{MLP}_{\text{query}}(s_i; W_{\text{query}}) \tag{3.31}$$

And a key encoder that embeds the positive and negative samples into a set of key vectors K_i for each individual i in the scene:

$$K_i = \{k_{i,0}, k_{i,1}, \ldots, k_{i,(N-1)\times 8}\} \tag{3.32}$$

$$k_{i,0} = \text{MLP}_{\text{key}}(\text{pos\_sample}_i; W_{\text{key}}) \tag{3.33}$$

$$k_{i,j} = \text{MLP}_{\text{key}}(\text{neg\_samples}_i[j]; W_{\text{key}}) \quad \forall j \in \{1, 2, \ldots, (N-1) \times 8\} \tag{3.34}$$

where W_{query} and W_{key} are the parameters of the MLP models.
The goal of the Social-NCE module is to maximize the similarity between the query vector q_i and the key vector $k_{i,0}$ corresponding to the positive sample, while minimizing the similarity between the query vector q_i and the key vectors $k_{i,j}$ corresponding to the negative samples. This is achieved by optimizing the Social-NCE contrastive loss detailed in section 3.3.3. A more in depth explanation of Social-NCE can be found in the original paper by Liu et al. [5].

3.2.6 Map-NCE Module

The Map-NCE module is an adaptation of the Social-NCE [5], module, based on contrastive learning, that uses the environment map to generate negative samples for training the model. The module is used only in Model2, as Model1 does not consider the environment map in the prediction process. The usefulness of this module comes from the fact that the training dataset contains only positive examples of trajectories, that is, examples of well-behaved trajectories that did not lead to collisions with the environment obstacles. To learn a robust representation that can generalize to new environments, the model must be able to distinguish between a pedestrian's actual future trajectory and potential discomfort zones

around the obstacles in the scene. This module forces the learned map and trajectory representations to contain useful information about the scene in order to avoid collisions with the obstacles. The Map-NCE module is composed of two submodules: a query encoder and a key encoder. The query encoder embeds the output of the Social Interaction Module (section 3.2.2) through a linear projection head, producing a query vector for each pedestrian. The key encoder embeds the positive and negative samples through a MLP, producing a set of key vectors for each pedestrian. Queries and keys are embedded in a shared space, so that the similarity between them can be computed using the dot product. The goal is to maximize the similarity between the query and the key corresponding to the true future trajectory, while minimizing the similarity between the query and the keys corresponding to the negative samples.
The procedure to obtain the positive and negative samples for pedestrian i is as follows. Given a future timestep t, the positive sample is the true future position of the pedestrian i at timestep t, with a small Gaussian noise to prevent overfitting, denoted by:

$$\text{pos\_sample}_i = y_i^t + \epsilon \tag{3.35}$$

where $\epsilon \sim \mathcal{N}(0, c_\epsilon I)$, and c_ϵ is a small constant, set to 0.05 meters in our experiments. The negative samples are obtained starting from the contours of the obstacles in the environment map patch. The map is processed to extract the contours of the obstacles. The contours are a set of points that form the boundaries of the obstacles in the map. The set of contour points for pedestrian i is denoted by:

$$\text{contours}_i = \{c_{i,1}, c_{i,2}, \ldots, c_{i,M_i}\} \tag{3.36}$$

where M_i is the number of contour points for pedestrian i. For each pedestrian i, we sample a subset of $Z = 10$ contour points, denoted by $\text{neg\_sample\_seeds}_i$:

$$\text{neg\_sample\_seeds}_i = \{c_{i,1}, c_{i,2}, \ldots, c_{i,Z}\} \tag{3.37}$$

The $\text{neg\_sample\_seeds}_i$ are used to generate 8 negative samples for each contour point $c_{i,z} \in \text{neg\_sample\_seeds}_i$ by adding a small displacement Δ_p:

$$\text{neg\_samples}_{i,z} = c_{i,z} + \Delta_p + \epsilon \tag{3.38}$$

$$\Delta_p = (\rho \cos\theta_p, \rho \sin\theta_p) \tag{3.39}$$

$$\theta_p = \frac{\pi}{4} p \quad \forall p \in \{0, 1, \ldots, 7\} \tag{3.40}$$

where ρ is a small constant, set to 0.5 meters in our experiments, and p is an index that ranges from 0 to 7, corresponding to the 8 displacement directions around the contour point. Again, a small Gaussian noise ϵ is added to the negative samples to prevent overfitting.
Finally the set of negative samples for pedestrian i is the union of the negative samples around all the Z selected contour points in the scene:

$$\text{neg\_samples}_i = \bigcup_{z \in \{1,2,\ldots,Z\}} \text{neg\_samples}_{i,z} \tag{3.41}$$

In order to perform the contrastive learning, the Map-NCE module defines a query encoder that embeds the social encoding s_i of each individual i in the scene into a query vector q_i:

$$q_i = \text{MLP}_{\text{query}}(s_i; W_{\text{query}}) \tag{3.42}$$

And a key encoder that embeds the positive and negative samples into a set of key vectors K_i for each individual i in the scene:

$$K_i = \{k_{i,0}, k_{i,1}, \ldots, k_{i,Z\times 8}\} \tag{3.43}$$

$$k_{i,0} = \text{MLP}_{\text{key}}(\text{pos\_sample}_i; W_{\text{key}}) \tag{3.44}$$

$$k_{i,z} = \text{MLP}_{\text{key}}(\text{neg\_samples}_i[z]; W_{\text{key}}) \quad \forall z \in \{1, 2, \ldots, Z \times 8\} \tag{3.45}$$

where W_{query} and W_{key} are the parameters of the MLP models.
The goal of the Map-NCE module is to maximize the similarity between the query vector q_i and the key vector $k_{i,0}$ corresponding to the positive sample, while minimizing the similarity between the query vector q_i and the key vectors $k_{i,z}$ corresponding to the negative samples. This is achieved by optimizing the Map-NCE contrastive loss detailed in section 3.3.3.

3.3 Loss

The loss function used to train the model is a combination of several losses.

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + \lambda_{\text{Env-Col}}\mathcal{L}_{\text{Env-Col}} + \lambda_{\text{Social-NCE}}\mathcal{L}_{\text{Social-NCE}} + \lambda_{\text{Map-NCE}}\mathcal{L}_{\text{Map-NCE}} \tag{3.46}$$

where $\lambda_{\text{Env-Col}}$, $\lambda_{\text{Social-NCE}}$, and $\lambda_{\text{Map-NCE}}$ are hyperparameters that control the importance of each loss term.

3.3.1 MSE Loss

The main loss is the Mean Squared Error (MSE) between the predicted trajectories $\hat{Y}$ and the ground truth trajectories Y. For a scene with N pedestrians the MSE loss is defined as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}\left\|\hat{Y}_i - Y_i\right\|_2^2 \tag{3.47}$$

One thing to note is that, in the training procedure, the loss is computed once for each batch of scenes. But scenes can have different numbers of pedestrians, so the loss is computed for each pedestrian in the scene, and then averaged over all the pedestrians in the batch. As opposed to computing the loss for each scene and then averaging over all the scenes in the batch. This is done to prevent giving more importance to pedestrians in less crowded scenes. In this way all pedestrians have the same weight in the loss computation.

Variety Loss To train the model to generate more samples from the multimodal distribution of human trajectories, we use the variety loss proposed by Gupta et al. [11], instead of the pure MSE loss. This loss encourages the model to generate diverse trajectories, by backpropagating the gradient only for the best sample, and not for all the samples generated, preventing the model from collapsing to a single mode. The variety loss, for a single pedestrian i, is defined as:

$$\mathcal{L}^i_{\text{variety}} = \min_{k \in \{1,2,\dots,K\}} \left\| \hat{Y}_i^k - Y_i \right\|_2^2 \tag{3.48}$$

where K is the number of samples generated by the model.
For a scene with N pedestrians, the total variety loss is defined as:

$$\mathcal{L}_{\text{variety}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}^i_{\text{variety}} \tag{3.49}$$

3.3.2 Environment Collision Loss

To make the model learn to avoid obstacles, we introduce an environment collision loss, that penalizes the model when the sampled trajectories collide with the obstacles in the scene. The environment collision loss is a variation of the variety loss (explained in the previous section 3.3.1), where the mean squared error is computed not only for the best sample, but for all the samples that collide with the obstacles in the scene. In this way, the gradients are backpropagated also for the samples that collide with the obstacles, making the model learn to avoid them. Without this loss, the model has a tendency to ignore the obstacles, since the variety loss only backpropagates the gradients for the best sample, and the best sample never collides with the obstacles. Instead we want all the wrong samples to be penalized, so that the model can learn to generate all the samples in a consistent way with the environment.
For a pedestrian i, we denote the set of samples that collide with the obstacles in the scene as:

$$C_i = \{k \in \{1, 2, \dots, K\} \mid \text{collides}(\hat{Y}_i^k, I_i)\} \tag{3.50}$$

where $\text{collides}(\hat{Y}_i^k, I_i)$ is a function that returns true if the trajectory $\hat{Y}_i^k$ collides with the obstacles in the map patch I_i of the pedestrian i, and false otherwise.
Then, the environment collision loss for a single pedestrian i is defined as:

$$\mathcal{L}^i_{\text{Env-Col}} = \frac{1}{|C_i|} \sum_{k \in C_i} \left\| \hat{Y}_i^k - Y_i \right\|_2^2 \tag{3.51}$$

Finally, for a scene with N pedestrians, the total environment collision loss is defined as the average of the individual environment collision losses:

$$\mathcal{L}_{\text{Env-Col}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}^i_{\text{Env-Col}} \tag{3.52}$$

3.3.3 Social-NCE and Map-NCE Losses

Finally, we use two auxiliary contrastive losses to help the model learn where not to go. The first loss is the Social-NCE loss, introduced by Liu et al. [5],

that encourages the model's hidden representations to encode information about avoiding collisions with other agents. Inspired by this loss, we introduce a new loss, the Map-NCE loss, that encourages the model to encode information about the surrounding environment in the hidden representations. These contrastive losses are motivated by the fact that the datasets contain only positive examples, that is, examples of agents that are well behaving, but not negative examples, that is, examples of agents that collide with themselves or with obstacles. And so the model never learns what is wrong, but only what is right.
The general loss function for Social-NCE and Map-NCE, for a single pedestrian i, is defined as:

$$\mathcal{L}^i_{\mathbf{NCE}} = -\log \frac{\exp(\psi(q_i) \cdot \phi(k_{i,0})/\tau)}{\sum_{j=0}^{J} \exp(\psi(q_i) \cdot \phi(k_{i,j})/\tau)} \tag{3.53}$$

where ψ is the query encoder, ϕ is the key encoder, τ is the temperature parameter, and J is the number of negative samples. Recall also that $k_{i,0}$ is the positive sample, and $k_{i,j>0}$ are the negative samples. To compute the actual Social-NCE and Map-NCE losses, it is sufficient to pass to the equation the correct query, positive sample, and negative samples.
Both Social-NCE and Map-NCE losses are computed for each pedestrian in the scene, and then averaged over all the pedestrians in the scene. The total Social-NCE and Map-NCE losses are defined as:

$$\mathcal{L}_{\text{NCE}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}^i_{\text{NCE}} \tag{3.54}$$

where N is the number of pedestrians in the scene.

3.4 Synthetic Dataset

The ETH/UCY datasets [9, 10] that we use to train and evaluate the model have some limitations, such as the small number of scene environments, and number of pedestrians that interact with the obstacles. To overcome these limitations, we generate a synthetic dataset of pedestrian trajectories in environments with obstacles, that we use to pretrain the model before fine-tuning on the real-world datasets (more information about the training procedure can be found in section 3.5). In a later section we will show how the synthetic dataset affects the performance of the model on the real-world datasets (see 4.3.6).
The synthetic dataset is generated using a simple physics-based model of pedestrian movement, that takes into account the social interactions between pedestrians, and the interactions with the obstacles in the scene. The model is based on the RVO2 library [102, 103], which simulates the movement of pedestrians using the "Optimal Reciprocal Collision Avoidance" (ORCA) algorithm, by Van Den Berg et al. [58]. The different scene maps are generated by sampling a set of obstacles from a predefined set of shapes such as triangles, rectangles and circles, and placing them randomly in the scene, with random sizes and orientations. The pedestrians are generated by sampling a random initial position and a random goal position, both outside the obstacles, and then simulating their movement using the RVO2 library, that takes into account the social interactions between pedestrians, and

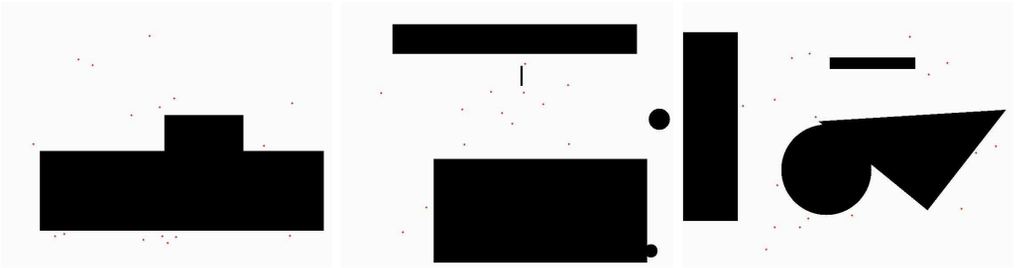

Figure 3.3: Example of scenes from the synthetic dataset. Each scene has a size of 25m x 15m, and contains obstacles (black) and pedestrians (red).

the interactions with the obstacles, using a simple physics-based model of pedestrian movement. The pedestrian trajectories are not always realistic, but at least, many of them interact with the obstacles in the scene, in a plausible way, giving the model the opportunity to learn to avoid them.
A similar synthetic dataset, containing only map patches, was used also to train the Map Patch Encoder 3.2.3 CNN, through an autoencoder architecture.

3.5 Training

A good training process is often crucial to the success of a deep learning model. In this section we describe the training procedure used to train the proposed models.

Dataset, data augmentation, and synthetic datasets The dataset used for training the model (ETH/UCY [9, 10]) (a more in depth description is provided in section 4.1) contains human trajectories described as a sequence of 2D coordinates sampled at a 0.4 second interval. Since the 2D coordinates are a characteristic of each particular scene, in the sense that they depend on how the authors of the dataset have defined the coordinate system, the model is trained to predict the relative displacement between consecutive frames instead of the absolute position of the human. In this way, the model can be applied to any scene, regardless of the coordinate system used, provided that the sampling interval is the same.
In order to prevent overfitting to dataset specific characteristics, such as more common directions of movement, the dataset is augmented by rotating and flipping the input data. Additionally, a small amount of noise is added to the input data to make the model more robust to small perturbations (e.g., the one caused by first-person view settings, as described in section 2.3.2.4).
For our purposes, the dataset has a few problems: most of the trajectories are linear, there are very few different scenes, and most of the agents rarely interact with the environment. These problems are quite important for the model to learn to predict realistic human trajectories in presence of obstacles. In fact, having very few different scenes can lead the model to overfit to the specific maps of the dataset, not being able to generalize to new unseen maps. And also if the agents rarely interact with the environment, the model will not learn to avoid obstacles. To solve these problems, we created a synthetic dataset with more complex scenes and more interactions with the environment (for more details see section 3.4). This dataset is used to pretrain the model before fine-tuning it on the real dataset. In a later section we will show how the synthetic dataset affects the performance of the model on the real-world datasets (see 4.3.6). A similar synthetic dataset,

containing only map patches, was used also to train the Map Patch Encoder 3.2.3 CNN, through an autoencoder architecture.
Since our model assumes that the binary scene maps, that segment the walkable areas from the obstacles, are available, we created them manually from the original scene images. In principle, these binary maps could be generated automatically using an image segmentation model, or the model could be modified to take as input the original scene images, but we chose to keep the model simple and use the binary maps as input.

Optimization algorithm and learning rate scheduler The model is trained using the Adam optimizer [104] with a learning rate of 3e-4, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The Adam optimizer is chosen because, in our testing, it has proven to be less sensitive to changes in the learning rate and other hyperparameters, and overall faster to converge, compared to SGD. We use a learning rate scheduler that halves the learning rate when the validation loss plateaus for 5 epochs (see figure 3.4). This learning rate schedule has been effective in our experiments, pushing the validation loss to lower values than just using a fixed learning rate. Often the scheduler reduces the learning rate to values close to 1e-5 before converging. The model is trained for a variable amount of epochs, usually between 50 and 80, stopping when the validation loss plateaus for 10 epochs. The best checkpoint is saved and used for evaluation.

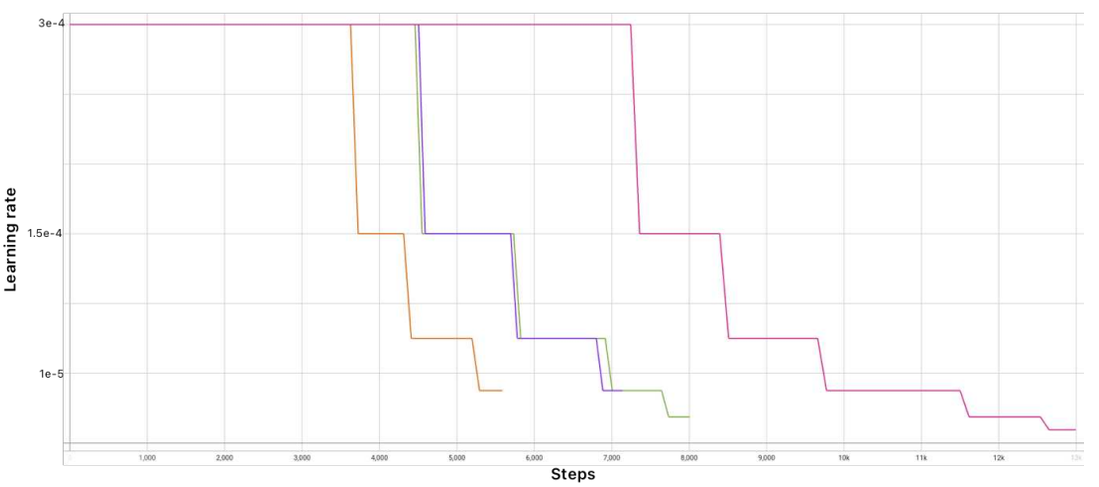

Figure 3.4: Example of learning rate schedule of some training runs.

Regularization To prevent overfitting, besides data augmentation, we use two techniques. First, we apply dropout [105] in the transformer that composes the social interaction module 3.2.2. Dropout randomly turns off some neurons during training, making the model less sensitive to specific neurons, and promoting the learning of more robust features. Second, we use weight decay, a regularization technique that shrinks the model's weights towards zero. This makes the weights fight to survive, and only if they provide a meaningful signal they will be kept. In this way, the model will be simpler and less prone to overfitting, by reducing the variability of the model.

Weights initialization The model's weights are initially set using the PyTorch's default initialization method, which samples from a uniform distribution

$\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{\text{fan\_in}}$. For the recurrent networks, we experimented with a more advanced initialization approach. Here, input-hidden weights were initialized using the Xavier initialization [106], while hidden-hidden weights were initialized orthogonally [107]. Bias terms were initialized to zero, except for the forget gate bias, set to 1 to encourage initial memorization. Despite these efforts, we observed no significant performance or convergence speed improvements compared to using the default initialization. Given the well-behaved gradients in both scenarios, we opted to stick with the default PyTorch initialization method.

Training process The training process has three main steps, however, the pretraining of the model with the synthetic pedestrian trajectories is optional. In the experiments section, we will explicitly specify when the model is pretrained on the synthetic dataset, and when it is not. The training process is as follows:

1. *Pretraining the Map Patch Encoder*: The Map Patch Encoder (section 3.2.3) CNN is trained in a separate step, before training the full model. It is trained using an autoencoder architecture on a synthetic dataset of map patches. The dataset is an infinite stream of map patches automatically generated on the fly, by randomly choosing a geometric shape, and its size, position, and orientation, and then rendering it on a blank canvas. The model is trained to compress the map patch into a latent representation, and then reconstruct them. This should make the model learn to encode the spatial information of the map in a low-dimensional representation. Since the dataset is infinite, the model can be trained for a large number of steps, without the need to worry about overfitting.

   In the successive training steps, the Map Patch Encoder CNN is frozen, and only the MLP on top of it is trained. This is done to prevent the model from forgetting the spatial information learned in the previous step.

2. *Pretraining the Model*: The model is pretrained on a synthetic dataset of human trajectories with obstacles (see section 3.4). The synthetic dataset is generated using a simple physics-based model of pedestrian movement, that takes into account the social interactions between pedestrians, and the interactions with the obstacles in the scene (see section 3.4). The pretraining optimization procedure is similar as the one explained above, where the model is trained until the validation loss plateaus.

   This step is optional, and in the experiments and results sections we will specify when the model is pretrained on the synthetic dataset. However, as we will show in the experiments section (section 4.3.6), pretraining the model can be beneficial to improve the obstacle avoidance capabilities of the model.

3. *Fine-tuning the Model*: Finally, the model is fine-tuned (or trained if not pretrained) on the real-world dataset of human trajectories. This helps the model to learn more realistic human behaviors, but starting from a better knowledge of how the environment shapes the trajectories.

Software and tools The model is implemented using PyTorch [108] and PyTorch Lightning [109], a high-level interface for PyTorch that simplifies the training process. We use TensorBoard [110] to monitor the training process and visualize

the results, as can be seen in figure 3.5. OpenCV [45] is used to perform some manipulations on the map images.

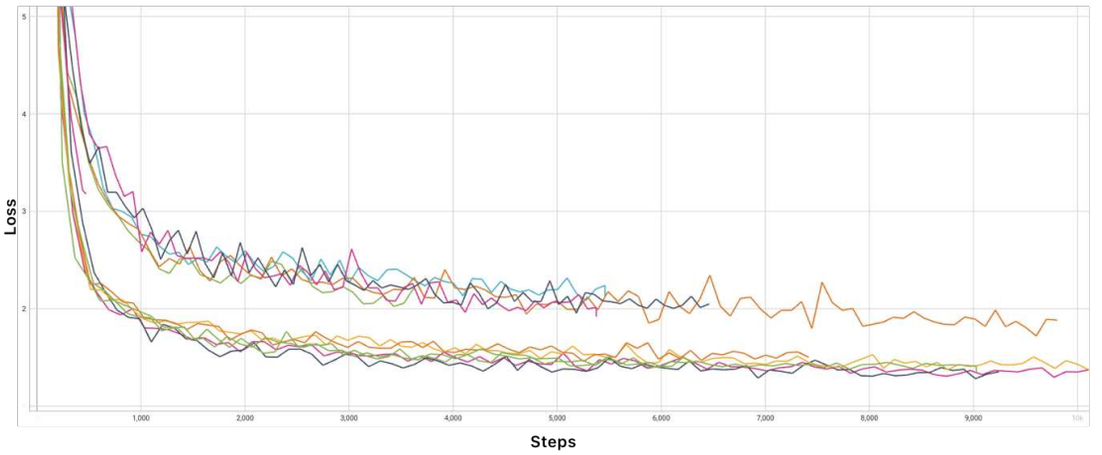

Figure 3.5: Example of what the validation losses of some training runs with different performance look like during training. The model reaches a good performance already after a few epochs. Then it slowly improves until it definitively plateaus. In this example, the different performances are due to different model sizes.

Chapter 4

Experiments

4.1 Datasets

4.1.1 ETH/UCY

The ETH/UCY dataset [9, 10] is a widely used publicly available benchmark for evaluating human trajectory prediction models. It consists of two datasets, ETH (also known as BIWI ETH), introduced by Pellegrini et al. [9], and UCY, introduced by Lerner et al. [10], which contain a total of five scenes captured from a bird's-eye view perspective. The ETH dataset includes two scenes: ETH main building ("eth") and ETH hotel ("hotel"), which are recorded respectively at a resolution of 640x480 pixels and 720x576 pixels. Both were recorded at 25 fps. Notably, the ETH dataset provides homography matrices, which are useful for converting between image coordinates and world coordinates, and vice versa. The UCY dataset includes three scenes: "univ", "zara1", and "zara2". All three scenes were recorded at a video resolution of 720x576 pixels and 25 fps. However, unlike ETH, the UCY dataset does not provide homography matrices, requiring to compute them separately. Both the datasets provide the annotations at 2.5 fps, that is 0.4s per time step. The positions are expressed in meters. The ETH/UCY dataset features more than 1500 pedestrians and provides manually annotated pedestrian trajectories, including a variety of challenging scenarios where pedestrians interact with each other and the environment, such as avoiding collisions and forming groups. On average, the datasets include between 3 to 10 people in the same frame, but the "univ" dataset includes more individuals, averaging over 30 people per frame. To ensure fair comparisons, researchers often use a leave-one-out approach, where the models are trained on four scenes and tested on the remaining one, with the results averaged over the five datasets. We use the same approach in our experiments.

(a) ETH (b) HOTEL (c) UNIV (d) ZARA1 (e) ZARA2

Figure 4.1: Example frames from the ETH/UCY dataset.

Two ETH dataset versions There exists another version of the ETH datasets where the ETH "main building" ("eth") is sped up by about 2 times[1]. This version became popular because one of the early works on trajectory forecasting with deep learning mistakenly created and used it, and then shared it on the model code repository. Many researchers have used this version since it provides the data already split into training, validation, and test sets, making it easier to use. Unfortunately, authors often do not mention which version of the dataset they used (likely because they are unaware of the issue), leading to confusion and incorrect comparisons. In this work, we report results on both versions of the dataset. We refer to the original version as "ETH/UCY" and the sped-up version as "ETH/UCY 2". Of course, the results on the sped-up version are worse than on the original version, not only when the sped-up version is used as test set, but also when it is used as training set. This is because in the first case the model is evaluated on trajectories that are out of the learned distribution, and in the second case the model is trained on trajectories that are not realistic, which makes it harder to learn the real distribution. In Figure 4.2, we show the histogram of pedestrian speeds in the "eth" dataset compared to the rest of the datasets.

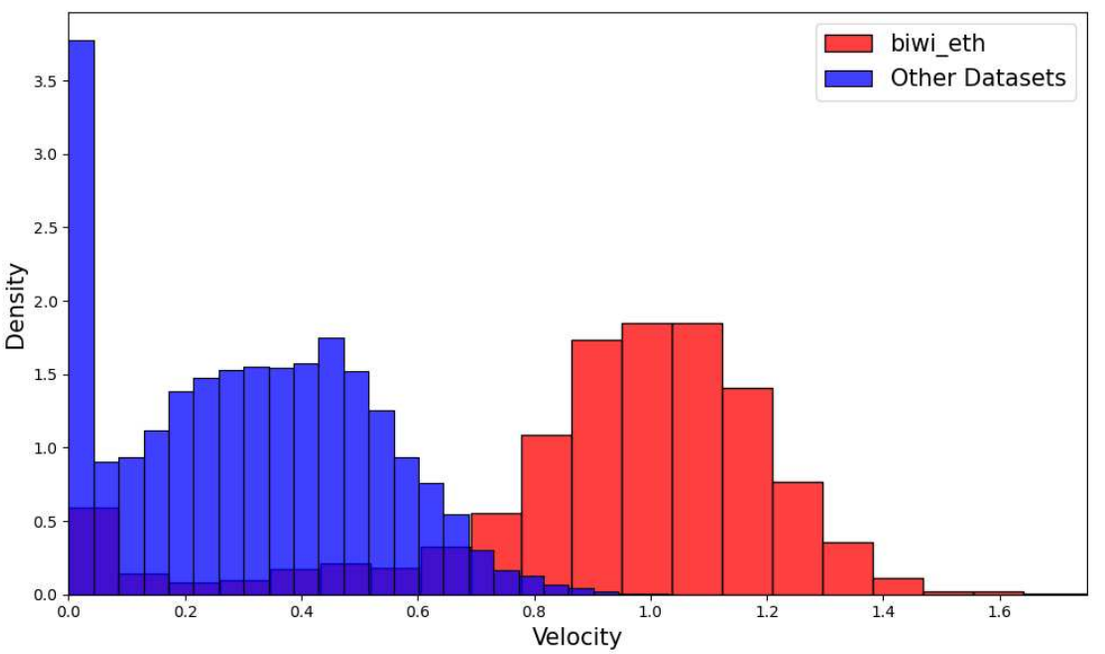

Figure 4.2: Histogram of pedestrian speeds in "eth" (red) vs the rest of the datasets (blue). The "eth" dataset has a higher speed than the other datasets, about 2 times faster. Velocity is expressed as meters every 0.4 seconds.

4.1.2 Internal Dataset

We also collected a small dataset of first-person videos of pedestrians walking, for example in a university campus, which we refer to as the "internal" dataset. The dataset consists of 3 videos, with a total of about 5 minutes of footage. The number of pedestrians in each video varies between 1 and 4. The videos were recorded at 1080p@30fps or 720p@25fps, depending on the camera used. The videos were recorded using a tripod, in a fixed position, meaning that there is no ego-motion. The dataset does not provide any annotations, it is only raw video footage, since the main goal is to qualitatively evaluate the model on real-world

[1] https://github.com/StanfordASL/Trajectron-plus-plus/issues/67

Figure 4.3: Example frames from the internal dataset.

noisy data. In the future, we plan to collect more videos to create a larger and more diverse dataset, moreover annotating the videos with pedestrian trajectories could be useful if one wants to train or quantitatively evaluate a model on this dataset. This dataset can be used by following the procedure described in section 2.3. The essential idea is to use a detection and tracking model to extract the pedestrian trajectories from the videos, then convert them to world coordinates using a homography matrix, and finally use them as input to the model.
The main reason for collecting this dataset was to have a first-person view dataset that is more closely related to the real-world scenarios that our model will be used in. In fact, the idea is to deploy the model on a robot that navigates in crowded environments. The model should be able to predict the trajectories of pedestrians in the robot's neighborhood, even in the presence of noise and occlusions, caused by the first-person viewpoint of the robot.

4.2 Evaluation Metrics

We evaluate the models on the ETH/UCY dataset using the classical metrics Average Displacement Error (ADE) and Final Displacement Error (FDE), which give a measure of the distance between the predicted and ground truth trajectories. We also evaluate the model with pedestrian collision metrics (COL-PRED, COL-GT), that measure the ability of the model to avoid collisions. Finally we introduce a novel metric, the *Environment Collision metric* (ENV-COL), that evaluates the ability of the model to avoid collisions with the obstacles in the scene.

4.2.1 Average Displacement Error (ADE)

The Average Displacement Error (ADE) measures the average Euclidean (L2) distance between the predicted trajectory and the ground truth trajectory at each future time step (lower is better). The ADE for the predicted trajectory of a single pedestrian i is defined as:

$$\text{ADE}_i = \frac{1}{T_{\text{pred}}} \sum_{t=T_{\text{obs}}+1}^{T_{\text{pred}}} \|\hat{y}_{i,t} - y_{i,t}\|_2 \tag{4.1}$$

where T_{obs} is the number of observed time steps, T_{pred} is the number of predicted time steps, $\hat{y}_{i,t}$ and $y_{i,t}$ are the predicted and ground truth positions of pedestrian i at time step t, respectively. The ADE for the entire dataset is then computed as the average of the ADEs of all pedestrians:

$$\text{ADE} = \frac{1}{N} \sum_{i=1}^{N} \text{ADE}_i \tag{4.2}$$

where N is the number of pedestrians in the dataset.
When evaluating a model that predicts multiple trajectories for each pedestrian, the ADE is computed for each predicted trajectory, and then the minimum ADE is selected (Best-of-K approach). It is defined as:

$$\text{ADE}_{\text{min}} = \frac{1}{N} \sum_{i=1}^{N} \min_{k} \text{ADE}_i^k \tag{4.3}$$

where ADE_i^k is the ADE of the k-th predicted trajectory of pedestrian i.

4.2.2 Final Displacement Error (FDE)

The Final Displacement Error (FDE) measures the Euclidean (L2) distance between the predicted final position and the ground truth final position of each pedestrian (lower is better). The FDE for the predicted trajectory of a single pedestrian i is defined as:

$$\text{FDE}_i = \left\| \hat{y}_{i,T_{\text{pred}}} - y_{i,T_{\text{pred}}} \right\|_2 \tag{4.4}$$

where $\hat{y}_{i,T_{\text{pred}}}$ and $y_{i,T_{\text{pred}}}$ are the predicted and ground truth final positions of pedestrian i, respectively. The FDE for the entire dataset is then computed as the average of the FDEs of all pedestrians:

$$\text{FDE} = \frac{1}{N} \sum_{i=1}^{N} \text{FDE}_i \tag{4.5}$$

Similarly to the ADE, when evaluating a model that predicts multiple trajectories for each pedestrian, the FDE is computed for each predicted trajectory, and then the minimum FDE is selected (Best-of-K approach). It is defined as:

$$\text{FDE}_{\text{min}} = \frac{1}{N} \sum_{i=1}^{N} \min_{k} \text{FDE}_i^k \tag{4.6}$$

where FDE_i^k is the FDE of the k-th predicted trajectory of pedestrian i.

4.2.3 Pedestrian Collisions (COL-PRED, COL-GT)

In crowded spaces, it is important for models to avoid collisions. To measure how well models can predict collision-free trajectories, we use two metrics: *Prediction Collision* (COL-PRED, COL-I) and *Ground Truth Collision* (COL-GT, COL-II), introduced by Kothari et al. [91]. The need for two metrics arises from the fact that the ADE and FDE metrics can be misleading, as they do not directly measure the ability of the model to avoid collisions. For example, a model may predict trajectories with low ADE and FDE values, but these trajectories may result in collisions, or vice versa, a model may predict trajectories with higher than average ADE and FDE scores, but these trajectories may be collision-free.

4.2.3.1 Prediction Collision (COL-PRED, COL-I)

The Prediction Collision metric (in short COL-PRED or COL-I) computes the percentage of predicted trajectories that result in collisions with the *predicted trajectories* of the other pedestrians in the scene. A lower COL-PRED value indicates that the model is better at producing collision-free future scenes.

4.2.3.2 Ground Truth Collision (COL-GT, COL-II)

The Ground Truth Collision metric (COL-GT) measures the percentage of predicted trajectories that collide with the actual positions of other pedestrians in the ground truth future scene. This metric (lower is better) helps to determine whether the model is able to understand the intentions of other pedestrians.

4.2.3.3 Implementation Details

To compute the collision metrics, we first augment the resolution of the trajectories by linearly interpolating consecutive positions. This is done to avoid, or at least reduce, false negatives (i.e., when the model predicts a collision, but the metric computation does not detect it). This can happen, for example, when two predicted trajectories are orthogonal to each other, and at least one of them is particularly fast. In this case, the predicted points may be farther apart than the threshold distance of pedestrian collision detection.
Formally, a trajectory i collides with a trajectory j if there exists a time step t such that the Euclidean distance between the two trajectories is less than a threshold τ:

$$\mathrm{COL}(i,j) = \min_{t \in T} \|y_{i,t} - y_{j,t}\|_2 < \tau \tag{4.7}$$

where

$$T = \{T_{\text{obs}} + 1, T_{\text{obs}} + 1.5, T_{\text{obs}} + 2, \ldots, T_{\text{pred}}\} \tag{4.8}$$

The fractional time steps represent the interpolated positions. The threshold τ is set to 0.2 meters, which is the same value used by Kothari et al. [91].

4.2.4 Environment Collisions (ENV-COL)

The obstacles in the scene constrain the possible trajectories that a pedestrian can take. Therefore, it is important for models to predict trajectories that avoid collisions with the obstacles. To measure how well models can predict collision-free trajectories with the environment, we introduce the *Environment Collision* metric (ENV-COL), which computes the percentage of predicted trajectories that collide with the obstacles in the scene. A lower ENV-COL value indicates that the model is better at producing environment compliant future trajectories.

4.2.4.1 Implementation Details

To compute the environment collision metric, we superimpose the predicted trajectories on the binary map of the scene, and count the number of predicted trajectories with at least one point over an obstacle. Then, we compute the percentage of trajectories that collide with the obstacles in the scene, by dividing the number

of colliding trajectories by the total number of predicted trajectories (comprising all sampled paths for all pedestrians).
Formally, we denote the set of samples that collide with the obstacles in the scene for a pedestrian i as:

$$C_i = \{k \in \{1, 2, \dots, K\} \mid \text{collides}(\hat{Y}_i^k, I)\} \tag{4.9}$$

where $\text{collides}(\hat{Y}_i^k, I_i)$ is a function that returns true if the k-th sampled trajectory $\hat{Y}_i^k$ collides with the obstacles in the scene map I.
Given a binary image I of the scene, where the pixel value is 0 if the pixel is an obstacle, and 1 otherwise, and a future trajectory $\hat{Y}_i^k$ of a pedestrian i, we define the function $\text{collides}(\hat{Y}_i^k, I)$ as:

$$\text{collides}(\hat{Y}_i^k, I) = \begin{cases} \text{True} & \text{if } \exists t \in T \text{ such that } I(g(\hat{y}_{i,t}^k)) = 0 \\ \text{False} & \text{otherwise} \end{cases} \tag{4.10}$$

where $g(\hat{y}_{i,t}^k)$ is the function that maps the trajectory point $\hat{y}_{i,t}^k$ to the corresponding pixel coordinates, and $I(g(\hat{y}_{i,t}^k))$ is the value of the pixel in the image I at the trajectory position $\hat{y}_{i,t}^k$. T is the set of future time steps defined as:

$$T = \{T_{\text{obs}} + 1, \dots, T_{\text{pred}}\} \tag{4.11}$$

Then, the environment collision metric for the whole dataset is computed as:

$$\text{ENV-COL} = \frac{1}{N} \sum_{i=1}^{N} |C_i| \tag{4.12}$$

where N is the number of predicted trajectories in the dataset (comprising all sampled paths).

4.3 Experiments

In this section, we present the experiments conducted to evaluate and validate the proposed models. We start by studying the performance of the proposed Social Interaction module (see 3.2.2) in section 4.3.1. Then, in section 4.3.2, we verify the effectiveness of the Social-NCE module [5] (see 3.2.5) in improving the collision avoidance capabilities of the model. Next, in section 4.3.3, we investigate the effectiveness of our proposed Map-NCE module (see 3.2.6) in improving the model's ability to predict environment-compliant trajectories. On a similar line, in section 4.3.4, we assess the efficacy of the Environment Collision loss (see 3.3.2), and then, in section 4.3.5, we measure the impact of using an offset in the scene map patch extraction (see section 3.2.3 and figure 3.1). Then, in section 4.3.6, we investigate the usage of a synthetic dataset (see 3.4) to pre-train the model, and together we analyze how a more regularized model performs. Finally, in section 4.3.7, we examine the inference speed of the models, to understand if they are suitable for real-time applications. The experiments are conducted on the ETH/UCY dataset, however, as illustrated in section 4.1.1, there are two versions of the dataset, the original one and a sped up version. Most of the experiments are conducted on the sped up "ETH/UCY 2" version, which is likely the most common in the literature. When necessary for a deeper understanding, the experiments are also conducted on the original version.

4.3.1 Social Interaction Module Ablation

In this experiment, we evaluate the performance of the proposed Social Interaction module (see 3.2.2) by comparing different variants of the module on the model1 architecture. In particular, the version that predicts only one trajectory for each pedestrian is used, that is no noise is passed to the model. This is standard practice in the literature, when evaluating the collision avoidance capabilities of the model, since there is no straightforward way to synchronize the walking mode encoded by the latent noise.
We compare the following variants:

- **No**: no social interaction module. This is the baseline model, it does not consider the interactions between pedestrians, so, each pedestrian is completely unaware of the others.

- **Attn**: attention-based social interaction module. This module uses a simple multi-head self-attention mechanism to aggregate the information from the other pedestrians. The query, key, and value vectors are computed from the pedestrian's history embeddings, and relative positions of the other pedestrians at the last time step (as explained in section 3.2.2).

- **T1**: transformer-based social interaction module with 1 layer. This variant uses the same transformer encoder based architecture proposed (see section 3.2.2), but with only one layer. Essentially the difference with the "Attn" (multi-head self-attention only) module is that the transformer encoder based module contains a feed-forward sub-layer that provides additional nonlinearity.

- **T2**: transformer-based social interaction module with 2 layers. This variant is exactly the proposed one (see 3.2.2). It is composed of two subsequent transformer layers. The idea is that the two layers can perform additional computation, and learn to deal better with the complex interactions between pedestrians.

- **T3**: transformer-based social interaction module with 3 layers. Same as the previous variant, but with 3 transformer encoder layers.

The results are shown in table 4.1. The table reports the Prediction Collision (COL-PRED, COL-I) and the Ground Truth Collision (COL-GT, COL-II) metrics for each scene, and the average over all scenes. The results show that, as the complexity of the social interaction module increases, the model's ability to avoid collisions improves. The baseline model (No) has the worst performance, as expected, since it does not consider the interactions between pedestrians, and has no way to optimize the trajectories to avoid collisions. The attention-based social interaction module (Attn) performs better than the baseline model, but worse than the transformer-based modules. The transformer-based modules (T1, T2, T3) achieve the best results. The model with just one layer (T1) performs slightly worse than the model with two layers (T2). The models with two and three layers (T2, T3) achieve similar results, in fact there is no significant difference between them, with the two layers model being slightly better. This suggests that the two layers are enough to capture the interactions between pedestrians, and adding more layers does not provide additional benefits.

Social	ETH	HOTEL	UNIV	ZARA1	ZARA2	AVG
No	2.76/1.23	1.13/1.44	2.58/2.01	1.58/1.67	1.72/1.59	1.95/1.59
Attn	2.57/0.89	0.77/0.94	0.85/0.68	1.03/1.08	1.55/1.05	1.35/0.92
T1	2.55/0.88	0.75/0.92	0.84/0.65	1.01/1.07	1.52/1.04	1.33/0.91
T2	2.47/**0.80**	**0.68**/0.89	**0.79**/**0.63**	**0.96**/**1.02**	**1.45**/**0.97**	**1.27**/**0.86**
T3	**2.41**/**0.80**	0.70/**0.87**	**0.79**/0.64	0.99/1.10	1.48/0.99	**1.27**/0.88

Table 4.1: Social Interaction module ablation study. COL-PRED/COL-GT. Lower is better. Comparison between different variants of the Social Interaction module on model1. No: no social interaction module. Attn: attention-based social interaction module. T1, T2, T3: different variants of the transformer-based social interaction module, with 1, 2, 3 layers.

4.3.2 Social-NCE Module Ablation

This experiment verifies the effectiveness of the Social-NCE module, introduced by Liu et al. [5], and described in section 3.2.5, in improving the "social" collision avoidance capabilities of the model. We compare the model1 architecture with and without the Social-NCE module. As in the previous experiment, we use the version that predicts only one trajectory for each pedestrian. Verifying the effectiveness of the Social-NCE module is important for us, since its architecture is the inspiration for our proposed Map-NCE module. If the Social-NCE module is effective, as reported in the original paper [5], then the Map-NCE module could also be effective.

The variants compared are:

- **Without Social-NCE**: the model1 without the Social-NCE module. This is the proposed model, referred to as T2 in the previous experiment, but without the Social-NCE module.

- **With Social-NCE**: the model1 with the Social-NCE module. This is exactly the proposed model1 architecture, including the Social-NCE module.

The results are shown in table 4.2. The table reports the Best-of-20 Average Displacement Error (ADE) and Final Displacement Error (FDE) metrics for each scene, expressed in meters (first row in each cell), and the COL-PRED/COL-GT metrics (second row in each cell). The results show that, from the point of view of the collision avoidance capabilities, the Social-NCE module is quite effective. The model with the Social-NCE module improves the Prediction Collision (COL-PRED) and Ground Truth Collision (COL-GT) metrics in all scenes, by about 0.2 percentage points on average. At the same time, it does not degrade the ADE and FDE metrics, in fact, there are no significant differences between the two models in terms of ADE and FDE.

Social-NCE	ETH	HOTEL	UNIV	ZARA1	ZARA2	AVG
No	**0.54/0.68**	**0.28/0.58**	**0.33/0.51**	**0.22/0.33**	**0.22/0.32**	**0.32/0.48**
	2.64/1.00	0.87/1.14	1.08/0.87	1.10/1.12	1.61/1.11	1.46/1.05
Yes	0.56/**0.68**	0.29/0.60	**0.33**/0.53	**0.22**/0.34	**0.22/0.32**	**0.32**/0.49
	2.47/0.80	**0.68/0.89**	**0.79/0.63**	**0.96/1.02**	**1.45/0.97**	**1.27/0.86**

Table 4.2: Social-NCE module [5] ablation study. Cell format: upper row Best-of-20 ADE/FDE in meters, lower row COL-PRED/COL-GT. Lower is better. Comparison between model1 without and with the Social-NCE module.

Additional experiments were conducted to tune the hyperparameters of the Social-NCE module, such as the temperature τ, the size of the representation space in which the similarity is computed, and the weight of the NCE loss in the total loss. As long as the hyperparameters were within a reasonable range, the model's performance was not significantly affected. In short, reasonable values for the hyperparameters should ensure that the representation space is smaller than the hidden representation input to the module (the one that is passed to the decoder). Moreover, since both the representation space dimension and the temperature affects the magnitude of the loss, the weight of the NCE loss should be tuned in such a way that it is in the order of the main task loss, so it does not take all the "gradient bandwidth", but it still has a significant impact on the model training. The hyperparameters that worked best for us are reported in the hyperparameters appendix A.

4.3.3 Map-NCE Module Ablation

In this experiment, we investigate the effectiveness of the proposed Map-NCE module (see 3.2.6) in improving the model's ability to predict environment-compliant trajectories, that is, trajectories that avoid collisions with the obstacles in the scene. For this experiment, we use the multimodal version of the model, that is the version that predicts multiple trajectories for each pedestrian, in this case 20, as it is standard practice in the literature. To better understand the effect of the Map-NCE module, we include in the comparison also the model1, that does not use any scene context information.
So, the variants compared are:

- **No**: the proposed model1 architecture (see 3.2). This model does not consider the scene context information, that is, each pedestrian is completely unaware of the obstacles in the scene. Therefore, it is useful as a baseline model to understand the starting point.

- **Map**: the model2 architecture (see 3.2), without the Map-NCE module. This is useful to understand the effect of including the scene context information in the model, but without using additional tricks.

- **M-NCE**: the proposed model2 architecture, (see 3.2) with the Map-NCE module (3.2.6).

Table 4.3 shows the results of the ablation study. As before, the table reports the Best-of-20 Average Displacement Error (ADE) and Final Displacement Error (FDE) metrics for each scene, expressed in meters (first row in each cell), and the COL-PRED/COL-GT metrics (second row in each cell). But it now also reports the Environment Collision (ENV-COL) metric (see 4.2.4) (third row in each cell). Several observations can be made from the results:

- By just adding the scene context information to the model (Map), the model's ability to predict environment-compliant trajectories improves significantly. It achieves better results in terms of the Environment Collision (ENV-COL) metric, in all scenes, compared to the model that does not use the scene context information (No). On average, it improves the ENV-COL metric by about 33%, decreasing from 6.98 to 4.65.
- The model that uses the Map-NCE module (M-NCE) achieves the best in terms of ENV-COL metric, in all scenes, compared to both the other two models. On average, it improves the ENV-COL metric by about 50% with respect to the model that does not use the scene context information, and by about 25% with respect to the Map model, bringing the ENV-COL metric down to 3.49.
- Despite the improvements in the ENV-COL metric, the models that use the scene context information (Map, M-NCE) worsen the ADE and FDE metrics compared to the model that does not use it (No). However, the degradation is on the order of 5 centimeters, which, depending on the application, could be acceptable, given the improvements in the ENV-COL metric.
- For what concerns the pedestrian collision avoidance capabilities, including the scene context information in the model, as in the Map and M-NCE models, does not significantly affect the Prediction Collision (COL-PRED) and Ground Truth Collision (COL-GT) metrics.

Scene context	**ETH**	**HOTEL**	**UNIV**	**ZARA1**	**ZARA2**	**AVG**
No	**0.56**/**0.68**	**0.29**/**0.60**	**0.33**/**0.53**	**0.22**/**0.34**	**0.22**/**0.32**	**0.32**/**0.49**
	2.47/0.80	**0.68**/0.89	0.79/0.63	**0.96**/1.02	1.45/0.97	**1.27**/0.86
	8.70	11.17	4.95	4.98	5.08	6.98
Map	0.62/0.75	0.31/0.62	0.37/0.69	0.26/0.42	**0.22**/0.34	0.36/0.56
	1.31/0.70	1.98/**0.82**	**0.78**/**0.62**	1.20/**0.79**	1.31/0.95	1.32/0.78
	5.71	8.42	3.93	2.56	2.62	4.65
M-NCE	0.61/0.73	0.30/0.61	0.35/0.66	0.26/0.41	**0.22**/0.35	0.35/0.55
	1.01/**0.50**	2.06/0.83	0.79/0.63	1.26/0.82	**1.29**/**0.94**	1.28/**0.74**
	4.38	**7.18**	**3.56**	**1.25**	**1.09**	**3.49**

Table 4.3: Map-NCE module (3.2.6) ablation study. Cell format: upper row Best-of-20 ADE/FDE in meters, middle row COL-PRED/COL-GT, lower row ENV-COL. Lower is better. Comparison between "No": no scene context, "Map": with scene context, and "M-NCE": with scene context and Map-NCE module.

Overall, the results suggest that the Map-NCE module is effective in improving the model's ability to predict trajectories that comply with the environmental constraints, without significantly affecting the quality of the predicted trajectories in terms of ADE, FDE, and pedestrian collision avoidance.

4.3.4 Environment Collision Loss Ablation

In this experiment, we analyze the effect of using the Environment Collision Loss (see 3.3.2) in the model training. The Environment Collision Loss is a simple loss that penalizes the model when it predicts trajectories that collide with the obstacles in the scene, in particular, it is a variation of the variety loss [11], where instead of backpropagating the gradients only for the best predicted trajectory, we backpropagate the gradients for all the predicted trajectories that collide with the obstacles in the scene. The idea is to provide additional supervision to the model, to help it learn to predict environment-compliant trajectories. To understand the effect of the Environment Collision Loss, we compare the model2 architecture with and without the loss.
The variants compared are:

- **No**: the model2 architecture without the Environment Collision Loss.
- **Yes**: the model2 architecture with the Environment Collision Loss. This is exactly the proposed model2 architecture.

Table 4.4 shows the results of the ablation study. The table reports the Best-of-20 ADE and FDE metrics for each scene, expressed in meters (first row in each cell), the COL-PRED/COL-GT metrics (second row in each cell), and the ENV-COL metric (third row in each cell). The following observations can be made:

- The model that uses the Environment Collision Loss (Yes) achieves better results in terms of the Environment Collision (ENV-COL) metric, in all scenes, compared to the model that does not use it (No). On average, it improves the ENV-COL metric by about 28%, decreasing from 4.84 to 3.49.
- The model that uses the Environment Collision Loss (Yes) achieves similar, or slightly worse, results in terms of ADE and FDE metrics, compared to the model that does not use it (No). The differences are not really significant, on the order of 1-2 centimeters. This is however expected, since the Environment Collision Loss reduces a bit the diversity of the predicted trajectories, by pulling them a little towards the mean trajectory. This small reduction in diversity is likely the cause of the slight degradation in the ADE and FDE metrics.
- Regarding the pedestrian collision avoidance capabilities, the model that uses the Environment Collision Loss (Yes) obtains similar results to the model that does not use it (No), suggesting that the Environment Collision Loss does not significantly improve or degrade the model's ability to avoid pedestrian collisions.

ENV-COL loss	ETH	HOTEL	UNIV	ZARA1	ZARA2	AVG
No	**0.58**/**0.71**	**0.30**/**0.61**	**0.35**/**0.63**	**0.24**/**0.38**	**0.22**/**0.33**	**0.34**/**0.53**
	1.54/0.78	**2.00**/0.84	**0.79**/**0.62**	**1.23**/0.83	1.32/0.95	1.38/0.80
	5.97	8.71	4.10	2.69	2.74	4.84
Yes	0.61/0.73	**0.30**/**0.61**	**0.35**/0.66	0.26/0.41	**0.22**/0.35	0.35/0.55
	1.01/**0.50**	2.06/**0.83**	**0.79**/0.63	1.26/**0.82**	**1.29**/**0.94**	**1.28**/**0.74**
	4.38	**7.18**	**3.56**	**1.25**	**1.09**	**3.49**

Table 4.4: Environment Collision Loss ablation study. Cell format: upper row Best-of-20 ADE/FDE in meters, middle row COL-PRED/COL-GT, lower row ENV-COL. Lower is better. Comparison between "No": without Environment Collision Loss, and "Yes": with Environment Collision Loss.

In summary, the results indicate that the Environment Collision Loss effectively enhances the model's ability to predict trajectories that adhere to environmental constraints, without compromising the quality of the predicted trajectories in terms of ADE, FDE, and pedestrian collision avoidance.

4.3.5 Map Patch Offset Ablation

One of the components that we argue helps the model to better predict trajectories that avoid collisions with the obstacles in the scene is the usage of an offset in the scene map patch extraction (see section 3.2.3 and figure 3.1). In practice, when we extract a patch around the pedestrian from the scene map, we shift the patch in such a way that it captures the environment in front of the pedestrian. This comes from our world knowledge, that pedestrians tend to walk in the same direction, and when deciding the path to follow, they look at the environment in front of them. Therefore, we argue that by shifting the patch in the direction of the pedestrian's heading, we provide the model with a more informative context that helps it to better predict trajectories. As a reminder, the patch captures a 10x10 meters area, and has a resolution of 100x100 pixels, so, each pixel corresponds to a 10x10 centimeters area.
To verify the effectiveness of this approach, we compare the model2 architecture with and without the offset in the scene map patch extraction. The variants compared are:

- **No**: the model2 architecture without the offset in the scene map patch extraction. The patch is extracted centered around the pedestrian's position, so it captures 5 meters around each direction (front, back, left, right). The patch is still rotated according to the pedestrian's heading.
- **Yes**: the model2 architecture with the offset in the scene map patch extraction. This is exactly the proposed model2 architecture. The patch is extracted in such a way that it captures 9 meters in front, 1 meter behind, and 5 meters on each side of the pedestrian.

Table 4.5 shows the results of the ablation analysis. The table reports the Best-of-20 ADE and FDE metrics for each scene, expressed in meters (first row in each

cell), the COL-PRED/COL-GT metrics (second row in each cell), and the ENV-COL metric (third row in each cell). The following observations can be made from the results:

- The model that uses the offset in the scene map patch extraction (Yes) improves the Environment Collision (ENV-COL) metric in all scenes, compared to the model that does not use it (No). On average, it improves the ENV-COL metric by about 4%, decreasing from 3.65 to 3.49.
- The employment of the offset in the scene map patch extraction (Yes) provides small but consistent improvements in the ADE and FDE metrics, compared to the model that does not use it (No). The improvements seem to be slightly more significant in the FDE metric. That could be expected, since the offset allows the model to see further ahead, allowing it to better predict the pedestrian's final position.
- The pedestrian collision avoidance abilities of the model that uses the offset (Yes) are a bit better on average than the model that does not use it (No). However, the improvements are less consistent across the scenes, and the differences are likely not significant.

This shows that the signal provided by the offset in the scene map patch extraction is useful for the model to predict better paths, in particular for the avoidance of collisions with the obstacles in the scene.

Map patch offset	**ETH**	**HOTEL**	**UNIV**	**ZARA1**	**ZARA2**	**AVG**
No	0.64/0.76	0.31/0.63	0.37/0.68	0.27/0.43	0.23/0.38	0.36/0.58
	1.08/0.55	**1.99/0.82**	**0.79/0.62**	1.28/0.85	1.30/**0.94**	1.29/0.76
	4.52	7.30	3.78	1.36	1.31	3.65
Yes	**0.61/0.73**	**0.30/0.61**	**0.35/0.66**	**0.26/0.41**	**0.22/0.35**	**0.35/0.55**
	1.01/0.50	2.06/0.83	**0.79**/0.63	**1.26/0.82**	**1.29/0.94**	**1.28/0.74**
	4.38	**7.18**	**3.56**	**1.25**	**1.09**	**3.49**

Table 4.5: Map patch offset ablation. Cell format: upper row Best-of-20 ADE/FDE in meters, middle row COL-PRED/COL-GT, lower row ENV-COL. Lower is better. Comparison between "No": without offset in the scene map patch extraction, and "Yes": with offset in the scene map patch extraction.

4.3.6 Synthetic Dataset

The ETH/UCY datasets [9, 10] that we use to train and evaluate the model have only a few different scene environments, and a small number of pedestrians that interact with the obstacles. This can be quite limiting, as the model may not be able to learn the obstacle collision dynamics well, and may not generalize well to new environments. To address this issue, we experiment with a synthetic dataset that is generated using the ORCA model by Van Den Berg et al. [58] (a more detailed explanation is provided in section 3.4). So, we compare the following variants:

- **No**: the model2 architecture trained only on the ETH/UCY datasets.
- **Yes**: the model2 architecture pretrained on the synthetic dataset and then fine-tuned on the ETH/UCY datasets.

For this experiment, we evaluate the models on both versions of the ETH/UCY datasets, the sped up version (ETH/UCY 2) and the original version (ETH/UCY). The results are shown in table 4.6 for the evaluation on the "ETH/UCY 2" (sped up) dataset, and in table 4.7 for the evaluation on the "ETH/UCY" (original) dataset. Both tables report the Best-of-20 ADE and FDE metrics for each scene, expressed in meters (first row in each cell), the COL-PRED/COL-GT metrics (second row in each cell), and the ENV-COL metric (third row in each cell). Some observations can be made from the results:

- The models pretrained on the synthetic dataset (Yes) achieve better results in terms of the Environment Collision (ENV-COL) metric, compared to the models that are not (No). On average, for the "ETH/UCY 2" dataset, the improvement on the ENV-COL metric is about 18%, decreasing from 3.49 to 2.85. For the "ETH/UCY" dataset, the improvement is about 27%, decreasing from 3.30 to 2.40.
- The improvements of the models pretrained on the synthetic dataset in the ENV-COL metric are less evident, or even negative, in the "zara1", "zara2", and "eth" scenes. This could be due to the fact that the "zara1" and "zara2" scenes are quite similar, and make it easier for the model to use the knowledge learned from one of them to predict the other. In such a way that, having a better model from the point of view of the obstacle avoidance, does not really help. For the "eth" scene, the most likely reason for the degradation when evaluating on the "ETH/UCY 2" dataset is that the scene is sped up, and the model struggles to transfer the knowledge learned from the synthetic dataset to a intrinsically different scenario. In fact, when evaluating on the "ETH/UCY" dataset, the pretrained model obtains much better results in terms of ENV-COL metric.
- The model pretrained on the synthetic dataset (Yes) when evaluated on the "ETH/UCY 2" dataset, achieves worse results in terms of the ADE and FDE metrics, compared to the model that is not (No). The differences are more evident in the FDE metric, where the degradation is on the order of 10 centimeters. However, the degradation is mostly influenced by the "eth" dataset, while the other datasets show similar results. And as before, the degradation is likely due to the fact that the "eth" scene is sped up, since when evaluating on the "ETH/UCY" dataset, the pretrained model obtains similar results to the model that is not, with only a slight degradation in the FDE metric.
- The pedestrian collision metrics (COL-PRED and COL-GT) are similar between the two models, in both the datasets, suggesting that the synthetic dataset does not significantly affect the model's ability to avoid pedestrian collisions.

The above results suggest that the synthetic dataset can be useful to improve the model's ability to predict environment-compliant trajectories. While the ADE

and FDE metrics are negatively affected on average, the degradation is mostly influenced by the "eth" dataset, that is sped up, and so not really representative of the real-world scenarios. In fact, when evaluating on the original "ETH/UCY" dataset, the pretrained ADE and FDE metrics are similar to the model that is not pretrained. Additionally, the synthetic dataset is intrinsically different from the real-world, so the model might struggle to generalize well to real-world scenarios. With further efforts in building a more realistic synthetic dataset, the model could potentially improve even more.

Synthetic	ETH	HOTEL	UNIV	ZARA1	ZARA2	AVG
No	**0.61**/**0.73**	**0.30**/0.61	0.35/**0.66**	**0.26**/**0.41**	**0.22**/**0.35**	**0.35**/**0.55**
	1.01/**0.50**	2.06/0.83	0.79/0.63	1.26/**0.82**	1.29/0.94	**1.28**/**0.74**
	4.38	7.18	3.56	**1.25**	1.09	**3.49**
Yes	0.76/1.08	0.31/**0.60**	**0.32**/**0.66**	**0.26**/0.47	0.23/0.41	0.38/0.64
	2.38/1.08	**1.35**/**0.53**	**0.69**/**0.57**	**1.11**/1.02	**1.22**/**0.88**	1.35/0.82
	4.56	**4.97**	**1.97**	1.68	**1.08**	**2.85**

Table 4.6: Synthetic dataset experiment, with evaluation on "ETH/UCY 2" (sped up) dataset. Cell format: upper row Best-of-20 ADE/FDE in meters, middle row COL-PRED/COL-GT, lower row ENV-COL. Lower is better. Comparison between "No": model2 trained only on the ETH/UCY datasets, and "Yes": model2 pretrained on the synthetic dataset and fine-tuned on the ETH/UCY datasets.

Synthetic	ETH	HOTEL	UNIV	ZARA1	ZARA2	AVG
No	0.30/**0.39**	**0.28**/**0.63**	0.34/0.67	**0.25**/**0.39**	**0.22**/**0.35**	**0.28**/**0.49**
	0.64/**0.50**	1.69/**0.62**	0.75/0.60	**0.89**/**0.74**	**1.32**/**0.87**	**1.06**/**0.67**
	4.22	6.73	3.60	**1.28**	**0.67**	3.30
Yes	**0.27**/0.40	0.31/0.79	**0.33**/**0.65**	**0.25**/0.42	**0.22**/0.38	**0.28**/0.53
	0.75/0.68	**1.59**/0.68	**0.70**/**0.58**	1.17/0.88	1.35/0.91	1.11/0.75
	1.81	**5.59**	**2.31**	1.33	0.95	**2.40**

Table 4.7: Synthetic dataset experiment, with evaluation on "ETH/UCY" (original) dataset. Cell format: upper row Best-of-20 ADE/FDE in meters, middle row COL-PRED/COL-GT, lower row ENV-COL. Lower is better. Comparison between "No": model2 trained only on the ETH/UCY datasets, and "Yes": model2 pretrained on the synthetic dataset and fine-tuned on the ETH/UCY datasets.

For the purpose of further improving the model's abilities to predict environment-compliant trajectories, we also tried to train our best model, with respect to the ENV-COL metric (model2 pretrained on synthetic dataset), with a stronger regularization, to understand if the model could generalize better to new environments. In particular, we tried to increase the dropout rate, and the weight decay. Moreover, we slightly increased the weight of the Environment Collision Loss and of the Map-NCE Loss in the total loss.
Table 4.8 shows that the more regularized model (M2-reg) achieves much better results in terms of the Environment Collision (ENV-COL) metric, compared to the other model (M2). On average, it improves the ENV-COL metric by about

47%, decreasing from 2.85 to 1.52. However, the regularization worsens the ADE and FDE metrics, by about 5-10 centimeters. The pedestrian collision metrics (COL-PRED and COL-GT) remain on the same level.

Model	ETH	HOTEL	UNIV	ZARA1	ZARA2	AVG
M2	**0.76**/**1.08**	**0.31**/**0.60**	**0.32**/**0.66**	**0.26**/**0.47**	0.23/0.41	**0.38**/**0.64**
	2.38/1.08	1.35/0.53	0.69/**0.57**	1.11/**1.02**	1.22/**0.88**	1.35/**0.82**
	4.56	4.97	**1.97**	**1.68**	1.08	2.85
M2-reg	0.81/1.20	0.35/0.84	0.41/0.81	0.33/0.53	0.25/0.44	0.43/0.76
	1.00/**0.78**	**0.88**/**0.39**	**0.68**/0.65	**0.98**/1.63	**1.13**/0.90	**0.93**/0.87
	2.90	**1.13**	2.25	2.25	**0.79**	**1.52**

Table 4.8: Stronger regularization experiment. Cell format: upper row Best-of-20 ADE/FDE in meters, middle row COL-PRED/COL-GT, lower row ENV-COL. Lower is better. Comparison between "M2": model2 pretrained on synthetic dataset and fine-tuned on the ETH/UCY datasets, and "M2-reg": model2 with stronger regularization (still pretrained on synthetic dataset and fine-tuned on the ETH/UCY datasets).

4.3.7 Inference Speed

The inference speed (or latency) of the model is an important factor to consider, especially when deploying the model in real-world applications. To evaluate the latency of the model, we measure the time it takes to predict the trajectories of all the pedestrians in a scene, for different scene sizes (number of pedestrians). We compare the inference speed of both our models, model1 and model2, using the synthetic dataset, since it contains a more significant number of scenes with a variety of scene sizes. In particular, the scene sizes range from 1 to 17 pedestrians.

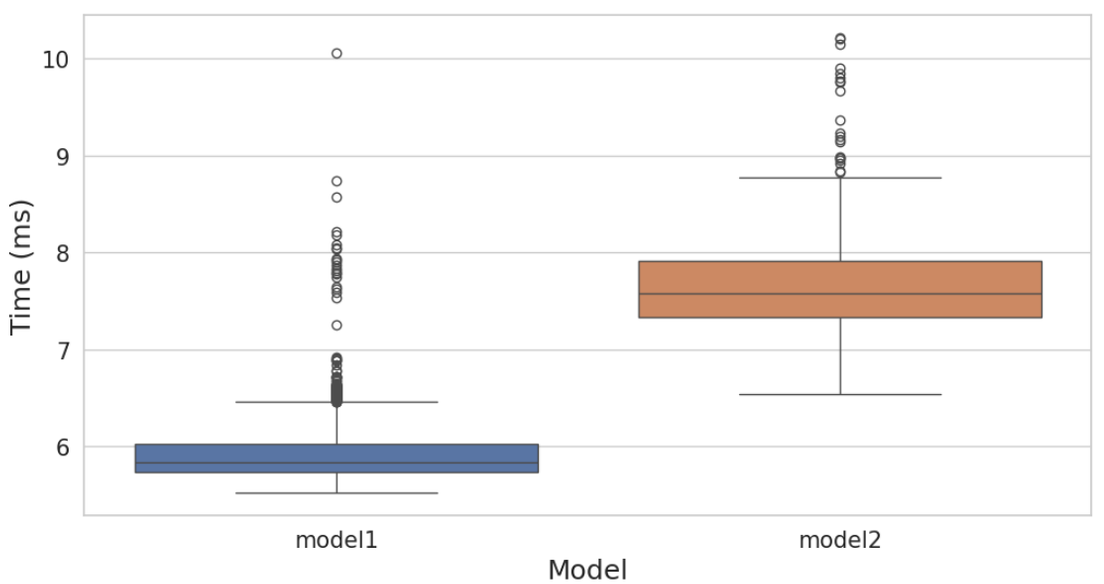

Figure 4.4: Boxplot showing the inference time of the two models. Model1, as expected, clearly outperforms model2 in terms of inference speed, in fact apart from a few outliers, the inference time of model1 is better than all the measured inference times of model2. Moreover, the first and third quartiles (which contain 50% of the data) of both models are quite close, indicating that the inference latency of the two models is quite stable. The same interpretation can be made for the whiskers of the boxplot.

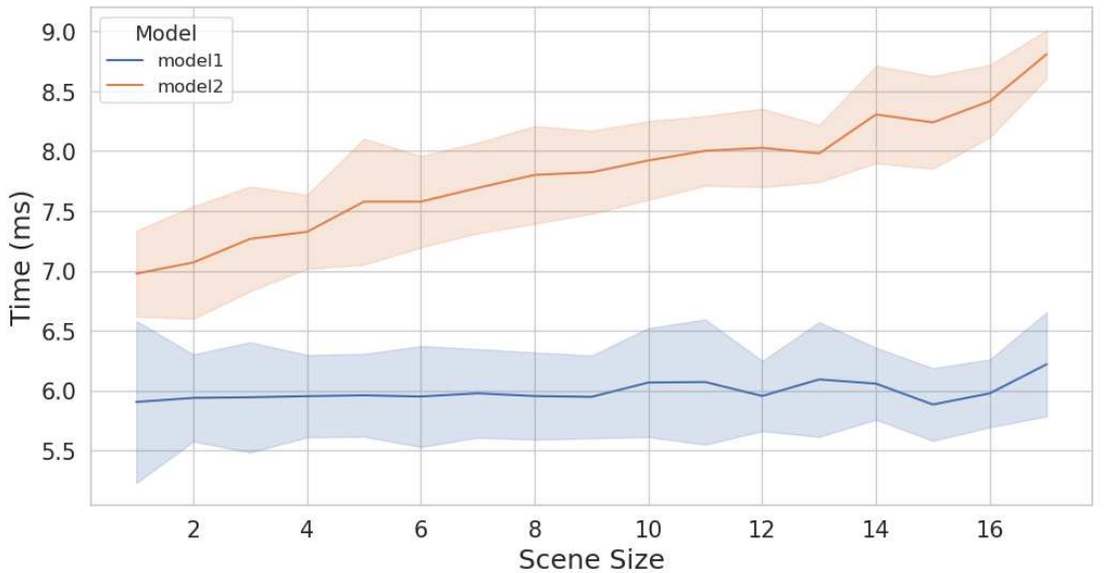

Figure 4.5: Plot showing the relation between the number of pedestrians in the scene and the inference time of the two models. The latency of model1 (blue) is quite stable across the different scene sizes, while the latency of model2 (orange) grows linearly with the number of pedestrians in the scene. Note that the outliers detected in the previous boxplot 4.4 are filtered out for better visualization. The bands around the lines represent the 95% confidence intervals. Best viewed in color.

The results are shown in figure 4.4 and figure 4.5, and are computed on a NVIDIA GeForce GTX 1050 Ti GPU [41] (with an Intel Core i5-8400 CPU [40]). The figures show that the model1 inference time is most of the times around 5.5-6.5 milliseconds, while the model2 inference time is around 6.5-9.0 milliseconds. Moreover, the model2 inference time clearly grows linearly with the number of pedestrians in the scene, while the model1 inference time seems not to be affected. This at least for scenes with up to 17 pedestrians. This is expected since model2 is more complex, and requires to extract and process the scene map patch for each pedestrian in the scene. The use of a transformer as social interaction module, should in principle, make the model latency grow as the number of pedestrians in the scene increase, since the transformer uses a self-attention mechanism that has a quadratic complexity with respect to the number of pedestrians. However, with the relatively small number of pedestrians in the scenes, this effect is not really evident. In conclusion, both models have more than acceptable inference times that allow them to be used in real-time applications.

Chapter 5

Results

In this chapter, we first present the results of the proposed method for human trajectory forecasting. We report the quantitative results on the ETH/UCY datasets [9, 10], comparing with the state-of-the-art methods. We then show some qualitative results on the ETH/UCY datasets and on an internal dataset.

5.1 Quantitative Results

We compare our models with several methods for human trajectory forecasting, including state-of-the-art methods. The results are reported in table 5.1. The first part of the table shows the Best-of-20 ADE/FDE results of the previous works, while the second part shows the results of our models on both versions of the ETH/UCY datasets. As explained in section 4.1.1, there exists two versions of the ETH/UCY datasets, that we refer to as "ETH/UCY" and "ETH/UCY 2". The "ETH/UCY" dataset is the original dataset, while the "ETH/UCY 2" dataset is a version where the "eth" scene is sped up by about 2 times. It was created by error and unfortunately got traction in the literature. Most works don't specify which version of the dataset they used for evaluation, so we report the results for both versions of the datasets. With this in mind, comparisons should be done with caution, since the results on the two versions can be quite different, making it difficult to compare the results of different works. Additionally, we report the COL-PRED/COL-GT (middle row in the table) and ENV-COL (bottom row in the table) metrics for our models. As it is standard in the literature, the evaluation is done using a lag period of 8 timesteps, and for a prediction horizon of 12 timesteps, with a 0.4s interval between timesteps, corresponding to 3.2s of past trajectories and 4.8s of future trajectories. More information about the datasets and evaluation metrics can be found in section 4.1 and 4.2.
We report the results for four models: *model1* (M1), *model2* (M2), *model2-synth* (M2-synth), and *model2-synth-reg* (M2-synth-reg). Model1 (M1) is our model that uses only the past trajectories as input, considering the social interactions between the agents. Model2 (M2), instead, uses also the environment map as input, in addition to the past trajectories. Model2-synth (M2-synth) and model2-synth-reg (M2-synth-reg) are the same as model2, but trained in a different way. They are both pretrained with synthetic data, and in addition, model2-synth-reg (M2-synth-reg) is trained with stronger regularization (as explained in section 4.3.6).
We can observe that our models don't achieve the best results in terms of ADE

and FDE. In particular, model1 (M1) is our best model in terms of ADE/FDE, having an average ADE that is 10-15 cm worse, and an average FDE that is 20-25 cm worse compared to the state-of-the-art, depending on the dataset version (ETH/UCY or ETH/UCY 2). Model2, and its variants (M2, M2-synth, M2-synth-reg), despite using the environment map as input, don't improve the ADE and FDE scores. This might be due to the fact that most of the trajectories in the datasets are not influenced by the environment, and so the environment map doesn't provide much useful information, feeding the model with a sort of noise that hurts the performance. This supposition is also supported by the performance of "Leapfrog" [79], which is particularly close to the state-of-the-art without using the environment map. Of course, an ideal model should be able to leverage the environment map when it is useful, and ignore it when it is not.
Despite the worse ADE/FDE scores, model2 and its variants (M2, M2-synth, M2-synth-reg) are much better than model1 (M1) in terms of obstacle collision avoidance, based on our novel ENV-COL metric. This is an important result, as it shows that our proposed Map-NCE module is effective in enhancing the collision avoidance capabilities of the models, as shown in the ablation analysis in section 4.3.3. However, M2-synth-reg, which is trained with stronger regularization, doesn't achieve satisfactory results in terms of ADE/FDE, and so it is not worth using it in practice, unless the avoidance of obstacles is really important.

Model	ETH	HOTEL	UNIV	ZARA1	ZARA2	AVG
Social-GAN [11]	0.81/1.52	0.72/1.61	0.60/1.26	0.34/0.69	0.42/0.84	0.58/1.18
SoPhie [61]	0.70/1.43	0.76/1.67	0.54/1.24	0.30/0.63	0.38/0.78	0.54/1.15
Social-BiGAT [62]	0.69/1.29	0.49/1.01	0.55/1.32	0.30/0.62	0.36/0.75	0.48/1.00
MG-GAN [72]	0.47/0.91	0.14/0.24	0.54/1.07	0.36/0.73	0.29/0.60	0.36/0.71
Trajectron++ [65]	0.67/1.18	0.18/0.28	0.30/0.54	0.25/0.41	0.18/0.32	0.32/0.55
TF Nets [73]	0.61/1.12	0.18/0.30	0.35/0.65	0.22/0.38	0.17/0.32	0.31/0.55
STAR [92]	0.36/0.65	0.17/0.36	0.26/0.55	0.22/0.46	0.31/0.62	0.26/0.53
Agentformer [75]	0.45/0.75	0.14/0.22	0.25/0.45	0.18/0.30	0.14/0.24	0.23/0.39
SMEMO [71]	0.39/0.59	0.14/0.20	0.23/0.41	0.19/0.32	0.15/0.26	0.22/0.35
MID [78]	0.39/0.66	0.13/0.22	0.22/0.45	0.17/0.30	0.13/0.27	0.21/0.38
Leapfrog [79]	0.39/0.58	0.11/0.17	0.26/0.43	0.18/**0.26**	0.13/0.22	0.21/0.33
Y-net [89]	0.28/0.33	0.10/0.14	0.24/0.41	0.17/0.27	0.13/0.22	0.18/0.27
NSP-SFM [80]	**0.25**/**0.24**	**0.09**/**0.13**	**0.21**/**0.38**	**0.16**/0.27	**0.12**/**0.20**	**0.17**/**0.24**
ETH/UCY 2						
M1 (ours)	0.56/0.68	0.29/0.60	0.33/0.53	0.22/0.34	0.22/0.32	0.32/0.49
	2.47/0.80	0.68/0.89	0.79/0.63	0.96/1.02	1.45/0.97	1.27/0.86
	8.70	11.17	4.95	4.98	5.08	6.98
M2 (ours)	0.61/0.73	0.30/0.61	0.35/0.66	0.26/0.41	0.22/0.35	0.35/0.55
	1.01/0.50	2.06/0.83	0.79/0.63	1.26/0.82	1.29/0.94	1.28/0.74
	4.38	7.18	3.56	1.25	1.09	3.49
M2-synth (ours)	0.76/1.08	0.31/0.60	0.32/0.66	0.26/0.47	0.23/0.41	0.38/0.64
	2.38/1.08	1.35/0.53	0.69/0.57	1.11/1.02	1.22/0.88	1.35/0.82
	4.56	4.97	1.97	1.68	1.08	2.85
M2-synth-reg (ours)	0.81/1.20	0.35/0.84	0.41/0.81	0.33/0.53	0.25/0.44	0.43/0.76
	1.00/0.78	0.88/0.39	0.68/0.65	0.98/1.63	1.13/0.90	0.93/0.87
	2.90	1.13	2.25	2.25	0.79	1.52
ETH/UCY						
M1 (ours)	$\overline{0.27}$/0.40	0.32/$\overline{0.54}$	0.34/$\overline{0.57}$	$\overline{0.21}$/$\overline{0.33}$	0.23/$\overline{0.32}$	$\overline{0.27}$/$\overline{0.43}$
	$\overline{0.62}$/0.51	1.99/1.29	0.79/0.61	$\overline{0.54}$/$\overline{0.53}$	1.41/1.08	1.07/0.80
	5.68	14.98	4.95	5.59	4.87	7.21
M2 (ours)	0.30/$\overline{0.39}$	$\overline{0.28}$/0.63	0.34/0.67	0.25/0.39	$\overline{0.22}$/0.35	0.28/0.49
	0.64/$\overline{0.50}$	1.69/0.62	0.75/0.60	0.89/0.74	1.32/$\overline{0.87}$	1.06/$\overline{0.67}$
	4.22	6.73	3.60	$\overline{1.28}$	$\overline{0.67}$	3.30
M2-synth (ours)	$\overline{0.27}$/0.40	0.31/0.79	$\overline{0.33}$/0.65	0.25/0.42	$\overline{0.22}$/0.38	0.28/0.53
	0.75/0.68	1.59/0.68	0.70/$\overline{0.58}$	1.17/0.88	1.35/0.91	1.11/0.75
	1.81	5.59	2.31	1.33	0.95	2.40
M2-synth-reg (ours)	0.40/0.81	0.38/0.90	0.40/0.79	0.31/0.51	0.26/0.47	0.35/0.70
	0.81/1.17	$\overline{0.92}$/$\overline{0.46}$	$\overline{0.64}$/0.66	0.94/1.45	$\overline{1.03}$/1.13	$\overline{0.87}$/0.97
	$\overline{0.63}$	$\overline{0.76}$	$\overline{2.03}$	1.56	0.72	$\overline{1.14}$

Table 5.1: Model comparison table. The first part of the table shows the Best-of-20 ADE/FDE (meters) results of the previous works, comprising also the state-of-the-art methods. Since the previous works do not report on which version of the dataset they evaluated, comparisons should be taken with caution. The second part of the table shows the results of our models on both versions of the ETH/UCY datasets: "ETH/UCY 2" (the sped up version) and "ETH/UCY" (the original version). Our models report not only the Best-of-20 ADE/FDE (first row), but also the COL-PRED/COL-GT (second row) and the ENV-COL (third row) metrics. The results are reported for 12 future timesteps, given the previous 8. **Bold**: overall best results. Underlined: our best results on "ETH/UCY 2". Overlined: our best results on "ETH/UCY".

5.2 Qualitative Results

In order to better understand the behavior of our models, we show some qualitative results on the ETH/UCY datasets, on the synthetic dataset and on the first-person view internal dataset. Moreover we try to explore the latent space of the model, to find examples of scenes that the model might predict.

5.2.1 ETH/UCY Datasets

Figure 5.1 shows an example of generated trajectories on the ETH/UCY eth dataset, in particular on the "zara2" scene. On the left, the trajectories predicted by model1, on the right, the trajectories predicted by model2. We can see that model1 predicts many trajectories that collide with the obstacles, since it doesn't use the scene context to predict the trajectories, while model2 predicts trajectories that adapt to the environment, for example by slowing down to avoid the obstacles, or predicting less dispersed trajectories when the pedestrian is walking near an obstacle.

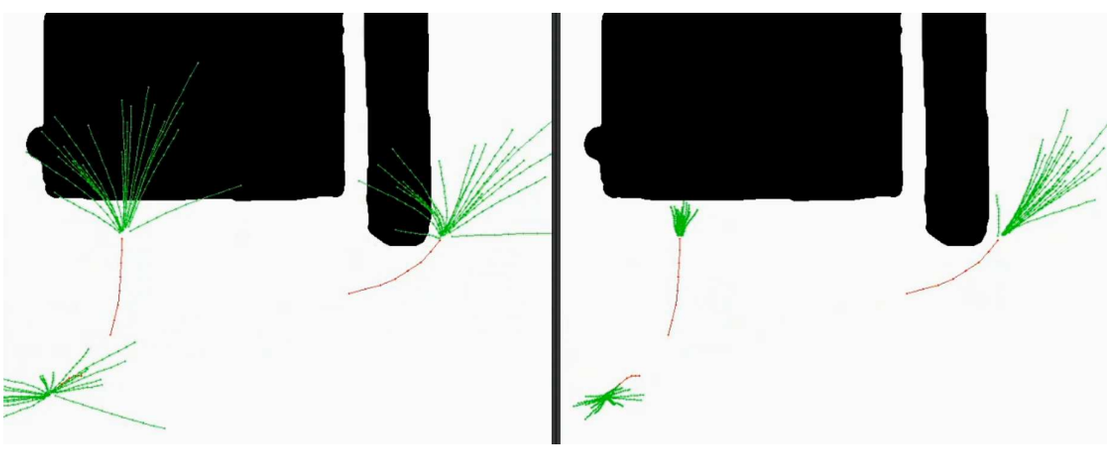

Figure 5.1: Example of generated trajectories on the ETH/UCY zara2 dataset. On the left, the trajectories predicted by model1, on the right, the trajectories predicted by model2. The past trajectories are shown in red, while the future trajectories are shown in green. The segmented environment map is shown in the background, with the obstacles in black. Model1 predicts many trajectories that collide with the obstacles, while model2 predicts trajectories that avoid the obstacles, for example by slowing down.

5.2.2 Synthetic Dataset

Similarly, figure 5.2 shows an example of generated trajectories on a scene of the synthetic dataset. As before, on the left, the trajectories predicted by model1, and on the right, the trajectories predicted by model2. We can see a similar behavior as in the previous example, where model1 predicts trajectories that collide with the obstacles, while model2 predicts trajectories that avoid the obstacles, while still maintaining a sufficient diversity of the predicted samples when far from the obstacles.

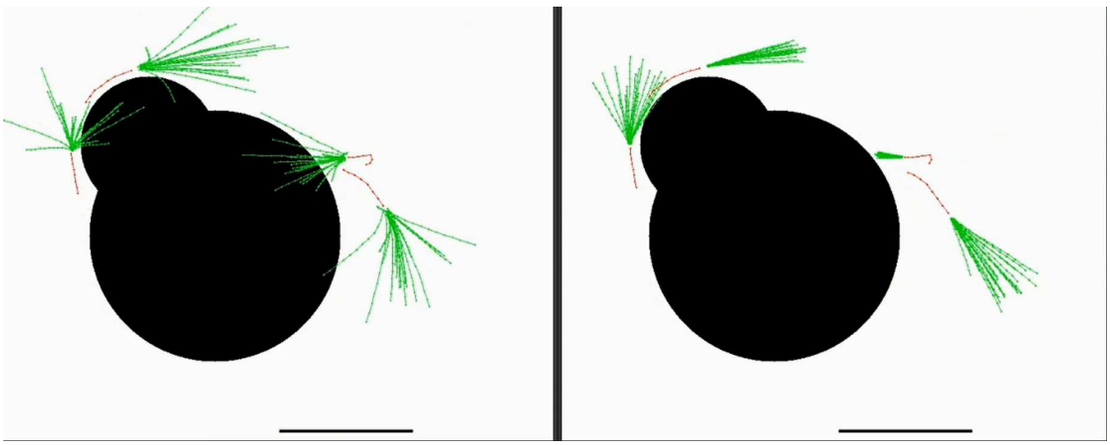

Figure 5.2: Example of generated trajectories on the synthetic dataset. On the left, the trajectories predicted by model1, on the right, the trajectories predicted by model2. The past trajectories are shown in red, while the future trajectories are shown in green. The segmented environment map is shown in the background, with the obstacles in black. Model1 predicts many trajectories that collide with the obstacles, while model2 predicts trajectories that avoid the obstacles, for example by slowing down.

5.2.3 Internal First-Person View Dataset

Figure 5.3 displays an example of generated trajectories on the internal first-person view dataset. On the left, the first-person view image, on the right, the bird's-eye view of the scene. For visualization purposes, we show five predicted trajectories for each person. This image makes it evident that the past trajectories are quite noisy, representing an out-of-distribution scenario, with respect to the training data. However, the smoothing of the trajectories, that can be observed by zooming in the image, helps to counter this noise, and the model is able to predict plausible trajectories, even in this challenging scenario.

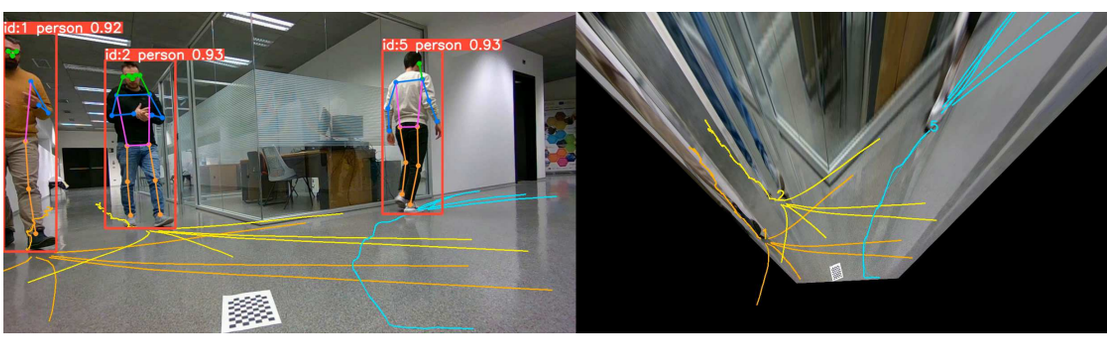

Figure 5.3: Example of generated trajectories on the internal first-person view dataset. On the left, the first-person view image, on the right, the bird's-eye view of the scene. Five predicted trajectories are shown for each person.

5.2.4 Latent Space Exploration

Finally, figure 5.4 shows an example of a predicted scene, obtained by manually exploring the latent space of the model. The past trajectories are shown in blue, while the future trajectories are shown in red. We can see that this scene may represent the meeting of two people, where one person changes direction to continue walking with the other person. This example shows the potential to sample different possible futures, with different levels of likelihood, by varying the likelihood of the latent noise, allowing to cover many possible scenarios, even the less common

ones. This example is generated with a relatively rare latent noise, as expected, since it represents a not so common event.

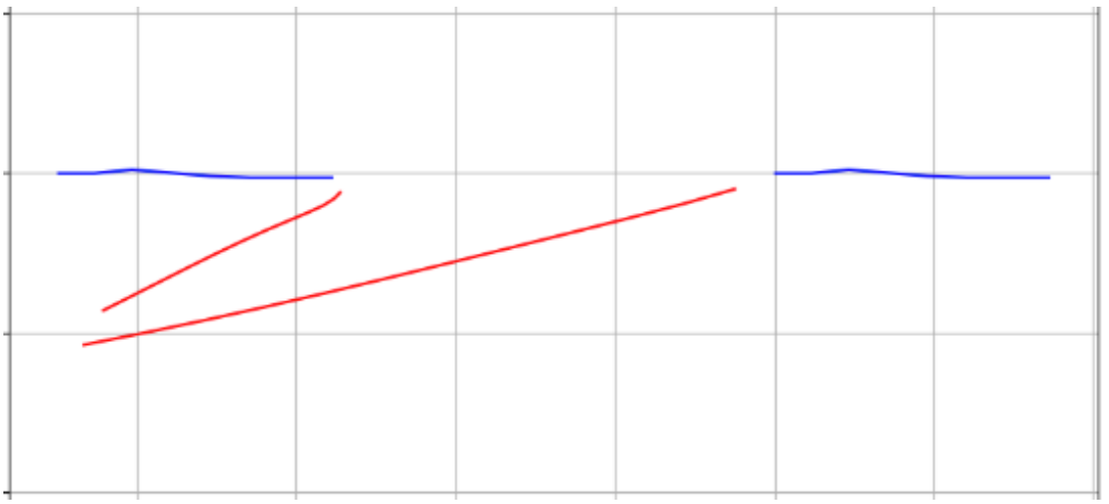

Figure 5.4: Example of predictable scene found by exploring the latent space of the model. The past trajectories are shown in blue, while the future trajectories are shown in red. We can see that this scene may represent the meeting of two people, where one person changes direction to continue walking with the other person.

Chapter 6

Conclusions

6.1 Summary

In this thesis, we have analyzed and addressed the problem of human trajectory forecasting, considering mostly the bird's-eye view setting, that is more common in the literature, but we also discussed the first-person view setting, which is potentially more useful in real-world applications. In particular, we discussed our prediction pipeline, and built our small first-person view dataset that we used to visualize the predictions of our models. Then, we proposed some new models, that exploit not only the past trajectories of the agents, but also the map of the environment, and we introduced the Map-NCE module, a contrastive learning based module, inspired by the Social-NCE module [5], that helps the model to predict environment-aware trajectories. For the same reason, we introduced the Environment Collision loss, a variation of the variety loss [11] that penalizes the model when it predicts trajectories that collide with obstacles, and developed a metric to evaluate the model's ability to predict environment-compliant trajectories, that we use to compare our models. We experimented with a synthetic dataset that we generated to provide our models with more data of pedestrians interacting with the environment. We trained and evaluated our models on the ETH/UCY datasets, highlighting the not so well known fact that there exists two versions of these datasets, the original one and a version that has the "eth" scene sped up. So, we reported results on both versions of the datasets. We performed extensive ablation studies and experiments, comparing the different variants of our models. The results of our models are not state-of-the-art, but the newly introduced Map-NCE module gives encouraging results, and the Environment Collision (ENV-COL) metric is a useful tool to evaluate the model's ability to predict environment-compliant trajectories.

6.2 Limitations and Future Work

Environment map The models that we proposed assume that the binary map of the environment is given. For forecasting of scenes with a fixed camera, framing a static environment this is a reasonable assumption, but in more complex scenarios, where the environment is dynamic, and the camera is moving, like in the first-person view setting when the camera is mounted on a robot, this assumption is not valid. In this case, the model should be able to perceive the environment, and build a map of it. Segmentation models could be used to build a semantic map of

the environment, even more detailed than a simple binary map, for example, a map with different classes of obstacles. Potential imprecisions in the map segmentation could hinder the model performance, but the greater amount of information could boost the model's ability to predict environment-compliant trajectories. So, in the future, we plan to investigate how to make the model perceive the environment, and build a map of it, and how to use this map to improve the model's predictions.

Environment collision avoidance The prediction of environment-compliant trajectories is not completely solved, since the models still predict trajectories that collide with obstacles. Future work could focus on further improving the usage of the environment. Similarly, given that our proposed Map-NCE module is model-agnostic, it could be interesting to use it in combination with other models that predict environment-compliant trajectories, to see if it can improve their performance.

Latent space interpretability The latent space learned by the models to predict multimodal future trajectories is not easily interpretable. Future work could focus on making it more interpretable. This could allow to sample more meaningful trajectories, for example, if we know that a certain dimension of the latent space corresponds to the speed of the agent, we could sample trajectories with different speeds, or if we know that a certain dimension corresponds to the direction of the agent, we could sample trajectories with different directions.

Mode collapse The models that we proposed, that accounts for the scene context, are more prone to mode collapse for pedestrians close to obstacles, that is, the samples generated by the models are less diverse, converging to a mean behavior. Studying this problem, and finding a solution to it, would improve the quality of the predictions.

Ego motion In the first-person view setting, the ego motion of the camera should be considered, since the camera is mounted on a robot, and the robot moves. Our models do not consider the ego motion, and assume that the camera is static. In the future, we plan to include the ego motion in the model, to make it more suitable for the first-person view setting.

Goal conditioning The models that we proposed do not allow to condition the predictions on some goals. The explicit modeling of goals is part of many state-of-the-art approaches, such as [80, 89]. This could be useful in many real-world applications, for example, in first-person view, a robot may identify some goals in the scene, such as a door, or an arbitrary point of interest, and condition its predictions on these goals. In the future, we plan to investigate how to include goal conditioning in our models.

Synthetic dataset The synthetic dataset that we generated is not perfect. A more realistic dataset could help the models to generalize better to real-world scenarios.

First-person view dataset The dataset that we built is really small, and has no ground truth annotations of pedestrian trajectories, as our goal was to visualize qualitatively the predictions of our models. In the future, we plan to collect more videos to create a larger and more diverse dataset. Moreover, annotating the videos with pedestrian trajectories could be useful if one wants to train or quantitatively evaluate a model on this dataset. Hopefully, when the dataset will be mature enough, we will release it to the public.

Bibliography

[1] Christopher Olah. Understanding lstm networks, Aug 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/. 1, 12, 13

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 14, 16

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 1, 16, 17

[4] MathWorks, 2024. URL https://www.mathworks.com/discovery/autoencoder.html. 1, 17

[5] Yuejiang Liu, Qi Yan, and Alexandre Alahi. Social nce: Contrastive learning of socially-aware motion representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15118–15129, 2021. 4, 8, 28, 36, 37, 40, 51, 53, 54, 69

[6] Mahir Gulzar, Yar Muhammad, and Naveed Muhammad. A survey on motion prediction of pedestrians and vehicles for autonomous driving. *IEEE Access*, 9:137957–137969, 2021. 7

[7] Giada Galati, Stefano Primatesta, Sergio Grammatico, Simone Macrì, and Alessandro Rizzo. Game theoretical trajectory planning enhances social acceptability of robots by humans. *Scientific Reports*, 12(1):21976, 2022. 7

[8] Holger Regenbrecht, Sander Zwanenburg, and Tobias Langlotz. Pervasive augmented reality—technology and ethics. *IEEE Pervasive Computing*, 21(3):84–91, 2022. 7

[9] Stefano Pellegrini, Andreas Ess, Konrad Schindler, and Luc Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *2009 IEEE 12th international conference on computer vision*, pages 261–268. IEEE, 2009. 8, 41, 42, 46, 58, 63

[10] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. *Computer Graphics Forum*, 26, 2007. URL https://api.semanticscholar.org/CorpusID:17374844. 8, 41, 42, 46, 58, 63

[11] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2255–2264, 2018. 8, 26, 27, 29, 35, 40, 56, 65, 69

[12] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006. 10

[13] Ching-Yao Chuang, Joshua Robinson, Yen-Chen Lin, Antonio Torralba, and Stefanie Jegelka. Debiased contrastive learning. *Advances in neural information processing systems*, 33:8765–8775, 2020. 11

[14] Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. Hard negative mixing for contrastive learning. *Advances in neural information processing systems*, 33:21798–21809, 2020. 11

[15] Senthil Purushwalkam and Abhinav Gupta. Demystifying contrastive self-supervised learning: Invariances, augmentations and dataset biases. *Advances in Neural Information Processing Systems*, 33:3407–3418, 2020. 11

[16] Joshua Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive learning with hard negative samples. *arXiv preprint arXiv:2010.04592*, 2020. 11

[17] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018. 11

[18] Sanjeev Arora, Hrishikesh Khandeparkar, Mikhail Khodak, Orestis Plevrakis, and Nikunj Saunshi. A theoretical analysis of contrastive unsupervised representation learning. *arXiv preprint arXiv:1902.09229*, 2019. 11

[19] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020. 11

[20] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020. 11

[21] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020. 11

[22] Lajanugen Logeswaran and Honglak Lee. An efficient framework for learning sentence representations. *arXiv preprint arXiv:1803.02893*, 2018. 11

[23] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. Unsupervised learning of sentence embeddings using compositional n-gram features. *arXiv preprint arXiv:1703.02507*, 2017. 11

[24] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 11

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 12

[26] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014. 13

[27] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 14

[28] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 14

[29] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023. 14

[30] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 15

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 16

[32] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989. 16

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 16

[34] Marc'Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *2007 IEEE conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2007. 17, 27

[35] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 17, 27

[36] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015. 18, 27

[37] Julio A Placed, Jared Strader, Henry Carrillo, Nikolay Atanasov, Vadim Indelman, Luca Carlone, and José A Castellanos. A survey on active simultaneous localization and mapping: State of the art and new frontiers. *IEEE Transactions on Robotics*, 2023. 18

[38] Iman Abaspur Kazerouni, Luke Fitzgerald, Gerard Dooly, and Daniel Toal. A survey of state-of-the-art on visual slam. *Expert Systems with Applications*, 205:117734, 2022. 18

[39] Ultralytics. Yolov8, Jan 2023. URL `https://docs.ultralytics.com/models/yolov8/`. 19

[40] Intel. Intel core i5-8400 processor (9m cache, up to 4.00 ghz) - product specifications, Oct 2017. URL `https://www.intel.com/content/www/us/en/products/sku/126687/intel-core-i58400-processor-9m-cache-up-to-4-00-ghz/specifications.html`. 19, 24, 62

[41] NVIDIA. Geforce gtx 1050 ti specifications, Oct 2016. URL `https://www.nvidia.com/en-gb/geforce/graphics-cards/geforce-gtx-1050-ti/specifications/`. 19, 24, 62

[42] Nir Aharon, Roy Orfaig, and Ben-Zion Bobrovsky. Bot-sort: Robust associations multi-pedestrian tracking. *arXiv preprint arXiv:2206.14651*, 2022. 21

[43] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960. 21

[44] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 21

[45] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 22, 45

[46] Abraham Savitzky and Marcel JE Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964. 23

[47] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995. 25, 29

[48] Kota Yamaguchi, Alexander C Berg, Luis E Ortiz, and Tamara L Berg. Who are you with and where are you going? In *CVPR 2011*, pages 1345–1352. IEEE, 2011. 25

[49] Matthias Luber, Johannes A Stork, Gian Diego Tipaldi, and Kai O Arras. People tracking with human motion predictions from social forces. In *2010 IEEE international conference on robotics and automation*, pages 464–469. IEEE, 2010. 25

[50] Ramin Mehran, Alexis Oyama, and Mubarak Shah. Abnormal crowd behavior detection using social force model. In *2009 IEEE conference on computer vision and pattern recognition*, pages 935–942. IEEE, 2009. 25

[51] Stefano Pellegrini, Andreas Ess, and Luc Van Gool. Improving data association by joint modeling of pedestrian trajectories and groupings. In *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part I 11*, pages 452–465. Springer, 2010. 25

[52] Alexandre Alahi, Vignesh Ramanathan, and Li Fei-Fei. Socially-aware large-scale crowd forecasting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2203–2210, 2014. 25

[53] Alexandre Robicquet, Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. Learning social etiquette: Human trajectory understanding in crowded scenes. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*, pages 549–565. Springer, 2016. 25

[54] Jos Elfring, René Van De Molengraft, and Maarten Steinbuch. Learning intentions for improved human motion prediction. *Robotics and Autonomous Systems*, 62(4):591–602, 2014. 25

[55] Andrey Rudenko, Luigi Palmieri, Achim J Lilienthal, and Kai O Arras. Human motion prediction under social grouping constraints. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3358–3364. IEEE, 2018. 25

[56] Andrey Rudenko, Luigi Palmieri, and Kai O Arras. Joint long-term prediction of human motion using a planning-based social force approach. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4571–4577. IEEE, 2018. 25

[57] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE international conference on robotics and automation*, pages 1928–1935. Ieee, 2008. 25

[58] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics Research: The 14th International Symposium ISRR*, pages 3–19. Springer, 2011. 25, 41, 58

[59] Markus Kuderer, Henrik Kretzschmar, Christoph Sprunk, and Wolfram Burgard. Feature-based prediction of trajectories for socially compliant navigation. In *Robotics: science and systems*, volume 8, pages 193–200, 2012. 25

[60] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016. 26, 27, 29

[61] Amir Sadeghian, Vineet Kosaraju, Ali Sadeghian, Noriaki Hirose, Hamid Rezatofighi, and Silvio Savarese. Sophie: An attentive gan for predicting paths compliant to social and physical constraints. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1349–1358, 2019. 26, 27, 28, 29, 65

[62] Vineet Kosaraju, Amir Sadeghian, Roberto Martín-Martín, Ian Reid, Hamid Rezatofighi, and Silvio Savarese. Social-bigat: Multimodal trajectory forecasting using bicycle-gan and graph attention networks. *Advances in neural information processing systems*, 32, 2019. 26, 27, 28, 29, 65

[63] Hao Xue, Du Q Huynh, and Mark Reynolds. Ss-lstm: A hierarchical lstm model for pedestrian trajectory prediction. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1186–1194. IEEE, 2018. 26, 27, 28, 29

[64] Pu Zhang, Wanli Ouyang, Pengfei Zhang, Jianru Xue, and Nanning Zheng. Sr-lstm: State refinement for lstm towards pedestrian trajectory prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12085–12094, 2019. 26, 27

[65] Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII 16*, pages 683–700. Springer, 2020. 26, 27, 28, 29, 65

[66] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 336–345, 2017. 26, 27, 28, 29

[67] Tharindu Fernando, Simon Denman, Sridha Sridharan, and Clinton Fookes. Soft+ hardwired attention: An lstm framework for human trajectory prediction and abnormal event detection. *Neural networks*, 108:466–478, 2018. 26, 27

[68] Federico Bartoli, Giuseppe Lisanti, Lamberto Ballan, and Alberto Del Bimbo. Context-aware trajectory prediction. In *2018 24th international conference on pattern recognition (ICPR)*, pages 1941–1946. IEEE, 2018. 26, 27, 28

[69] Ronny Hug, Stefan Becker, Wolfgang Hübner, and Michael Arens. Particle-based pedestrian path prediction using lstm-mdl models. In *2018 21st international conference on intelligent transportation systems (ITSC)*, pages 2684–2691. IEEE, 2018. 26

[70] Anirudh Vemula, Katharina Muelling, and Jean Oh. Social attention: Modeling attention in human crowds. In *2018 IEEE international Conference on Robotics and Automation (ICRA)*, pages 4601–4607. IEEE, 2018. 26, 27

[71] Francesco Marchetti, Federico Becattini, Lorenzo Seidenari, and Alberto Del Bimbo. Smemo: social memory for trajectory forecasting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024. 26, 29, 65

[72] Patrick Dendorfer, Sven Elflein, and Laura Leal-Taixé. Mg-gan: A multi-generator model preventing out-of-distribution samples in pedestrian trajectory prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13158–13167, 2021. 26, 28, 29, 65

[73] Luca Franco, Leonardo Placidi, Francesco Giuliari, Irtiza Hasan, Marco Cristani, and Fabio Galasso. Under the hood of transformer networks for trajectory forecasting. *Pattern Recognition*, 138:109372, 2023. 26, 27, 29, 65

[74] Yicheng Liu, Jinghuai Zhang, Liangji Fang, Qinhong Jiang, and Bolei Zhou. Multimodal motion prediction with stacked transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7577–7586, 2021. 26

[75] Ye Yuan, Xinshuo Weng, Yanglan Ou, and Kris M Kitani. Agentformer: Agent-aware transformers for socio-temporal multi-agent forecasting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9813–9823, 2021. 26, 29, 65

[76] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 26

[77] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR, 2021. 26

[78] Tianpei Gu, Guangyi Chen, Junlong Li, Chunze Lin, Yongming Rao, Jie Zhou, and Jiwen Lu. Stochastic trajectory prediction via motion indeterminacy diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17113–17122, 2022. 26, 27, 29, 65

[79] Weibo Mao, Chenxin Xu, Qi Zhu, Siheng Chen, and Yanfeng Wang. Leapfrog diffusion model for stochastic trajectory prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5517–5526, 2023. 26, 27, 29, 64, 65

[80] Jiangbei Yue, Dinesh Manocha, and He Wang. Human trajectory prediction via neural social physics. In *European conference on computer vision*, pages 376–394. Springer, 2022. 26, 27, 29, 65, 70

[81] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 26

[82] Tianyang Zhao, Yifei Xu, Mathew Monfort, Wongun Choi, Chris Baker, Yibiao Zhao, Yizhou Wang, and Ying Nian Wu. Multi-agent tensor fusion for contextual trajectory prediction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12126–12134, 2019. 26, 27, 28, 29

[83] Patrick Dendorfer, Aljosa Osep, and Laura Leal-Taixé. Goal-gan: Multimodal trajectory prediction based on goal position estimation. In *Proceedings of the Asian Conference on Computer Vision*, 2020. 26

[84] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. *Advances in neural information processing systems*, 30, 2017. 26

[85] Boris Ivanovic, Edward Schmerling, Karen Leung, and Marco Pavone. Generative modeling of multimodal multi-human behavior. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3088–3095. IEEE, 2018. 27

[86] Boris Ivanovic and Marco Pavone. The trajectron: Probabilistic multi-agent trajectory modeling with dynamic spatiotemporal graphs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2375–2384, 2019. 27

[87] Thomas Gilles, Stefano Sabatini, Dzmitry Tsishkou, Bogdan Stanciulescu, and Fabien Moutarde. Home: Heatmap output for future motion estimation. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 500–507. IEEE, 2021. 27

[88] Thomas Gilles, Stefano Sabatini, Dzmitry Tsishkou, Bogdan Stanciulescu, and Fabien Moutarde. Gohome: Graph-oriented heatmap output for future motion estimation. In *2022 international conference on robotics and automation (ICRA)*, pages 9107–9114. IEEE, 2022. 27

[89] Karttikeya Mangalam, Yang An, Harshayu Girase, and Jitendra Malik. From goals, waypoints & paths to long term human trajectory forecasting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15233–15242, 2021. 27, 28, 29, 65, 70

[90] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015. 27

[91] Parth Kothari, Sven Kreiss, and Alexandre Alahi. Human trajectory forecasting in crowds: A deep learning perspective. *IEEE Transactions on Intelligent Transportation Systems*, 23 (7):7386–7400, 2021. 27, 49, 50

[92] Cunjun Yu, Xiao Ma, Jiawei Ren, Haiyu Zhao, and Shuai Yi. Spatio-temporal graph transformer networks for pedestrian trajectory prediction. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XII 16*, pages 507–523. Springer, 2020. 27, 29, 65

[93] Amir Sadeghian, Ferdinand Legros, Maxime Voisin, Ricky Vesel, Alexandre Alahi, and Silvio Savarese. Car-net: Clairvoyant attentive recurrent network. In *Proceedings of the European conference on computer vision (ECCV)*, pages 151–167, 2018. 28

[94] Jiachen Li, Hengbo Ma, and Masayoshi Tomizuka. Conditional generative neural system for probabilistic trajectory prediction. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6150–6156. IEEE, 2019. 28

[95] Karttikeya Mangalam, Harshayu Girase, Shreyas Agarwal, Kuan-Hui Lee, Ehsan Adeli, Jitendra Malik, and Adrien Gaidon. It is not the journey but the destination: Endpoint conditioned trajectory prediction. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 759–776. Springer, 2020. 28

[96] Matteo Lisotto, Pasquale Coscia, and Lamberto Ballan. Social and scene-aware trajectory prediction in crowded spaces. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019. 28

[97] Takuma Yagi, Karttikeya Mangalam, Ryo Yonetani, and Yoichi Sato. Future person localization in first-person videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7593–7602, 2018. 28, 29

[98] Huikun Bi, Ruisi Zhang, Tianlu Mao, Zhigang Deng, and Zhaoqi Wang. How can i see my future? fvtraj: Using first-person view for pedestrian trajectory prediction. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*, pages 576–593. Springer, 2020. 28, 29

[99] Benjamin Stoler, Meghdeep Jana, Soonmin Hwang, and Jean Oh. T2fpv: Dataset and method for correcting first-person view errors in pedestrian trajectory prediction. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4037–4044. IEEE, 2023. 28

[100] Jianing Qiu, Lipeng Chen, Xiao Gu, Frank P-W Lo, Ya-Yen Tsai, Jiankai Sun, Jiaqi Liu, and Benny Lo. Egocentric human trajectory forecasting with a wearable camera and multi-modal fusion. *IEEE Robotics and Automation Letters*, 7(4):8799–8806, 2022. 28

[101] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016. 35

[102] Jamie Snape and lithander. snape/rvo2: v2.0.2, sep 2022. URL `https://doi.org/10.5281/zenodo.7039667`. 41

[103] Sybren Stüvel. Sybrenstuvel/python-rvo2: Optimal reciprocal collision avoidance, python bindings, Aug 2020. URL `https://github.com/sybrenstuvel/Python-RVO2`. 41

[104] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 43

[105] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 43

[106] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010. 44

[107] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013. 44

[108] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 44

[109] William Falcon and The PyTorch Lightning team. PyTorch Lightning, mar 2019. URL `https://github.com/Lightning-AI/lightning`. 44

[110] Google. tensorflow/tensorboard, 2016. URL `https://github.com/tensorflow/tensorboard`. 44

Appendix A

Hyperparameters

Hyperparameter	M1	M2	M2-synth	M2-synth-reg
TRAJ ENCODER				
hidden_size	32	32	32	32
SOCIAL MODULE				
hidden_size	32	32	32	32
num_layers	2	2	2	2
nhead	4	4	4	4
dim_feedforward	64	64	64	64
dropout	0.2	0.3	0.3	0.5
norm_first	false	false	false	false
activation	relu	relu	relu	relu
TRAJ DECODER				
hidden_size	32	48	48	48
noise_dim	8	8	8	8
noise_distrib	gaussian	gaussian	gaussian	gaussian
SOCIAL-NCE				
loss_weight	3	3	3	3
temperature	0.5	0.5	0.5	0.5
projection_size	16	16	16	16
MAP-NCE				
num_points	—	10	10	10
loss_weight	—	3	3	3.5
temperature	—	0.5	0.5	0.5
projection_size	—	16	16	16
MAP ENCODER				
bottleneck_size	—	64	64	64
output_size	—	8	8	8
ENV-COL				
loss_weight	—	0.5	0.5	0.5
DATA				
batch_size	32	32	32	32
rotate_prob	1	1	1	1
flip_prob	1	1	1	1
noise_prob	0.1	0.1	0.1	0.1
noise_scale	0.05	0.05	0.05	0.05
OPTIMIZER				
name	Adam	Adam	Adam	Adam
lr	3e-4	3e-4	3e-4	3e-4
betas	0.9/0.999	0.9/0.999	0.9/0.999	0.9/0.999
weight_decay	1e-5	1e-5	1e-5	2e-4
early_stop_patience	10	10	10	10
LR SCHEDULER				
name	ReduceLROnPlateau	ReduceLROnPlateau	ReduceLROnPlateau	ReduceLROnPlateau
factor	0.5	0.5	0.5	0.5
patience	5	5	5	5

Table A.1: Hyperparameters used for the different models.