Efficient Multimodal Pedestrian Trajectory Prediction with Vectorized Representations

Marwan Mashra [1] **Julien Moreau** [2] **Lina Achaji** [2]

Abstract

In today's rapidly evolving landscape of transportation, we're on the edge of an era where intelligent systems such as Autonomous vehicles (AVs) are becoming increasingly popular. In order to navigate safely in their environment, Autonomous vehicles rely on predicting future trajectories of surroundings agents (pedestrians, vehicles...etc) to plan their future movements, anticipate risks, and avoid collisions. A popular approach called goal-based methods consist of first predicting multiple end goals for each agent, then forecasting their trajectories towards those goals. For pedestrian trajectory predictions, most top-performing goal-based approaches have been using the same exact method to predict the goals, which is a CNN-based network introduced in the Y-net architecture in 2020. This approach falls when it comes to efficiency and speed for both training and inference, making it less interesting for AVs applications. In this work, we focus on efficient predictions, borrowing the concept of vectorized representations from the field of vehicle trajectory predictions. We use a new approach to create vectorized representations from the semantic segmentation of 2D scene images. We then adapt the state-of-the-art model DenseTNT, originally designed for vehicle trajectory predictions, reducing by more than 85% the training time of its second stage, while maintaining its performance. Our final model shows a significant efficiency advantage over CNN-based approaches, while keeping competitive performance.

[1]Université Paris-Saclay, Gif-sur-Yvette 91190, France [2]Stellantis Group, Technical center of Velizy 78140, France. Correspondence to: Marwan Mashra <marwan.mashra@universite-paris-saclay.fr>, Lina Achaji <lina.achaji@stellantis.com>, Julien Moreau <julien.moreau@stellantis.com>.

1. Introduction

We are approaching an era where fully self-driving cars are expected to become increasingly common in our streets. These intelligent systems rely on different planning and prediction algorithms to navigate autonomously and safely through their surrounding complex environment. One of those algorithms is trajectory prediction, which allow the system to forecast future trajectories of different surrounding agents (vehicles, pedestrians, cyclists...etc) based on their past observed positions. This plays an important role in planning future movements, predicting potentials risks, and avoiding collisions. Most research in the field of Trajectory predictions for Autonomous systems has been focusing on Vehicle trajectory predictions. However, as autonomous vehicles continue to transition from highway driving to navigating complex urban environments, accurate predictions of pedestrian trajectories become particularly more critical.

Accurately Predicting a human trajectory is a hard task that requires addressing two main aspects. The first being the social interactions between different agents. It is clear that a pedestrian's trajectory is directly impacted by other pedestrians' trajectories (Helbing & Molnár, 1995). Therefore, forecasting future trajectories requires modeling pedestrians' dynamics and social interactions. This problem was addressed in many papers including Social-LSTM (Alahi et al., 2016), Social-GAN (Gupta et al., 2018), NSP (Yue et al., 2023), PreTR (Achaji et al., 2022) and others.

In addition to modeling pedestrians' social interactions, a second important aspect revolves around the uncertainty in people's behaviors and intentions, as well as the randomness in their movements. To account for these uncertainties, it is very common to predict multiple possible trajectories for each pedestrian, which is known as Multimodal Pedestrian trajectory prediction.

Although generative models such as GANs (Gupta et al., 2018) are capable of generating multiple different trajectories, they often suffer from mode collapse (Thanh-Tung & Tran, 2020), where all generated trajectories end at almost the same location instead of covering all different modes or possible destinations.

To tackle the question of diversity in the agent's predicted trajectories, another family of methods called Goal-based methods have become more popular in the field of trajectory predictions. Goal-based methods start by first predicting a predetermined number of end goals or possible destinations of the agent, and then complete the trajectories leading to those goals. One of the most well-known goal-based methods for pedestrian trajectory predictions is Y-net (Mangalam et al., 2020a). Introduced in 2020 by researchers from UC Berkeley and the University of Munich, Y-net achieved state-of-the-art performance by utilizing several CNN-based networks (Ronneberger et al., 2015) to segment the scene image, predict the goals, and even generate the trajectories.

Since 2020, almost every single top-performing goal-based method for pedestrian trajectory prediction has been relying on the same exact CNN-based goal prediction module introduced in Y-net. Most of the focus has been towards modeling the social interactions through an attention-based backbone (Chiara et al., 2022), or through the social force model (Yue et al., 2023). In spite of the good performance, this CNN-based goal prediction module falls when it comes to efficiency for both training and inference. This efficiency problem was already addressed in the field of vehicle trajectory prediction with the introduction of vectorized representations (Gao et al., 2020).

For vehicle trajectory predictions, most datasets provide HD maps which are highly accurate maps containing all the structural information such as lanes, intersections, traffic lights...etc. Traditionally, these maps are converted to images in the pixel space to be later encoded through a ConvNet. In contrast, the idea of vectorized representations is to convert all input data, including trajectories and HD maps, to vectors to avoid the intensive computation. This was first introduced in VectorNet (Gao et al., 2020) and has since been dominating the field of vehicle trajectory predictions (Liang et al., 2020), (Gilles et al., 2022). However, its use in pedestrian trajectory predictions has been limited.

In this work, we aim to explore the use of vectorized representations for pedestrian trajectory predictions. (1) We tackle the lack of HD maps in pedestrian datasets, presenting a method to create vectorized map information based on semantic segmentation of 2D scene images. (2) We then revisit DenseTNT (Gu et al., 2021), a goal-based state-of-the-art model for vehicle trajectory predictions with a VectorNet backbone, improving its efficiency, and adopting it as a goal prediction module for pedestrian trajectory predictions. (3) We combine this goal prediction module to PreTR (Achaji et al., 2022) to generate multimodal trajectory predictions. (4) Finally, we show that our model presents a significant efficiency advantage over CNN-based approaches in both training and inference time, while keeping competitive performance.

2. Related Work

2.1. Vectorized representations in VectorNet

For vehicle trajectory predictions, most datasets such as Argoverse (Chang et al., 2019) provide a High-definition map along with the trajectories of all surrounding agents. A High-definition map, also known as an HD map, is usually captured through advanced radar sensors. HD maps are precise at a centimeter level and contain details such as lanes, intersections, crosswalks, traffic lights and signs, as illustrated in Figure 1 (left).

Prior to VectorNet, the common approach of encoding these maps was to render them as images, also known as rasterized representations, and then encode them through a ConvNet which is a computationally intensive and inefficient step. In 2020, researchers from Google and Waymo introduced VectorNet (Gao et al., 2020), where they propose to encode agents trajectories as well as all HD Map elements into vectors, as shown in Figure 1 (right).

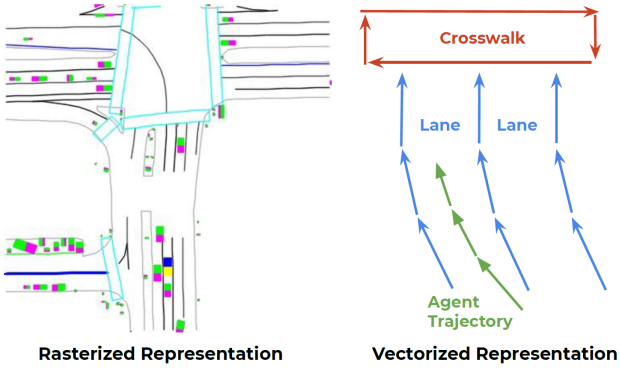

Figure 1. Illustration from the VectorNet paper (Gao et al., 2020). (left) rasterized rendering of an HD map, (righ) vectorized approach to represent an HD map and agent trajectories.

VectorNet is used as a backbone or an encoder to extract the features from the scene and generate encodings of the agents. It takes as an input the vectorized representation of the scene data, and encodes it through a hierarchical graph neural network in two steps. First, it constructs the Polyline subgraphs where every node is a vector from the input data, and two nodes are connected if they belong to the same polyline (same agent, lane, crosswalk...etc). This subgraph allows it to exploit the spatial and semantic locality of the nodes. Second, it constructs the Global interaction graph which is a fully connected graph where every node is an encoding of a Polyline subgraph. This allows it to model the high-order interactions between different subgraphs (different agents, map elements...etc). The output of VectorNet is a feature vector for each agent that encodes both its local features and global interactions with other elements.

2.2. DenseTNT: Goal-based trajectory prediction

A goal-based method aims to predict multiple future trajectories of a target agent by first predicting its possible final goals or destinations. More precisely, in order to generate K future trajectories, it starts by generating a heatmap that represents the probability of each point on the map to be the end goal of the agent. It then uses this heatmap to sample or predict a set of K end goals that are both highly probable and diverse to cover all different modes of the distribution. Finally, it generates a trajectory for each predicted goal.

DenseTNT is a popular goal-based model that was introduced in 2021 and achieved state-of-the-art at the moment for vehicle trajectory prediction (Gu et al., 2021). It ranked 1st on the Argoverse motion forecasting benchmark and won 1st place in the 2021 Waymo Open Dataset Motion Prediction Challenge. It uses a Two-stage training strategy where it first learns to generate the heatmap, and then learns to predict the goals. Furthermore, it introduced an **Offline optimization-based algorithm** to provide multi-future pseudo-labels to train the goal predictor in the second stage.

First stage training. During the first stage, the model is trained to generate the heatmap in 2 steps. The first step is called *Lane scoring*, in which the model tries to predict a small set of lanes where the end goal of the agent is likely to be. This is done to reduce the search space from all surrounding lanes to only a smaller subset, in order to make the computation in the following steps more efficient. In the second step, the model generates *Dense goals* with a certain density on the selected lanes, then uses an attention-based probability estimation module to compute the probability for each goal. These probabilities represent the heatmap outputted in the first stage. Finally, a trajectory completion module, simple MLP, is also trained during the first stage to predict the future trajectory of an agent given the end goal. This trajectory predictor is trained using teacher forcing, meaning that it receives the ground truth end goal during the training.

Second stage training. During the second stage, the model is trained to predict K end goals given the heatmap. Since the ground truth only have one end goal, they propose to use an offline optimization-based algorithm that computes a set of end goals which represent the pseudo-labels used to compute the loss of the goal predictor and optimize the network during training. This algorithm generates pseudo-labels as following: we randomly choose a set of goal candidates, then perform λ number of steps where each step consists of applying a small random perturbation to all goal candidates in the current set, moving them around slightly. Then we keep the new set instead of the current one if it has a smaller expected error. The expected error of the goal set $\mathbf{y}$ is computed as follows:

$$\mathbb{E}[d(\mathbf{y}, Y)] = \sum_{i=1}^{m} h\left(c_i\right) d\left(\mathbf{y}, c_i\right) \tag{1}$$

where $c_1, c_2, ..., c_m$ are all existing goals candidates, and $d(\mathbf{y}, c_i)$ is the minimum distance between all goals of the set $\mathbf{y}$ and the goal candidate c_i, and $h\left(c_i\right)$ is the probability of the goal candidate c_i from the previously generated heatmap.

2.3. PreTR: Multi-agent trajectory prediction

PreTR, or Prediction Transformer, is a non-autoregressive transformer-based pedestrian trajectory prediction model developed by researchers from Stellantis (Achaji et al., 2022). It aims to learn the social dynamics and interactions in multi-agent scenes by employing a spatio-temporal attention module. More precisely, this module consists of two consecutive attention modules, the first one computes self-attention across the temporal domain (all time steps of the agent), and the second one applies self-attention on the spatial axe (between different agents). This allows it to consider every agent independently over all time steps, before attending to all agents, learning both spatial and temporal features.

Furthermore, PreTR achieves non-autoregressive generation of future sequences using a parallel decoding technique (Ghazvininejad et al., 2019). This approach consists of forwarding a set of learned object queries through the decoder of the transformer network. Those learned queries represent trajectory proposals that get refined in the decoder through several layers of cross-attention. At the output of the decoder, each refined trajectory proposal represent a future position of an agent. This parallel decoding technique allows PreTR to decode all future positions at once, which has lower computational complexity than autoregressive decoders.

To summarize, the encoder in PreTR takes as input the encoded positions of the agents, combined with both time and agent encodings, whereas the decoder receives a set of learned queries. Both are forwarded through multiple blocks of attention modules that operate on both spatial and temporal domains. The final output of the decoder contains all future positions of all agents decoded in parallel.

However, it is important to note that the trajectory generation process in PreTR is deterministic, meaning it can only predict one possible trajectory for each agent in a given scene. As a result, PreTR isn't capable of generating multimodal pedestrian trajectory predictions. This makes it hard to evaluate and compare its performance on pedestrian trajectory prediction since most benchmarks measure the performance on K predicted trajectories.

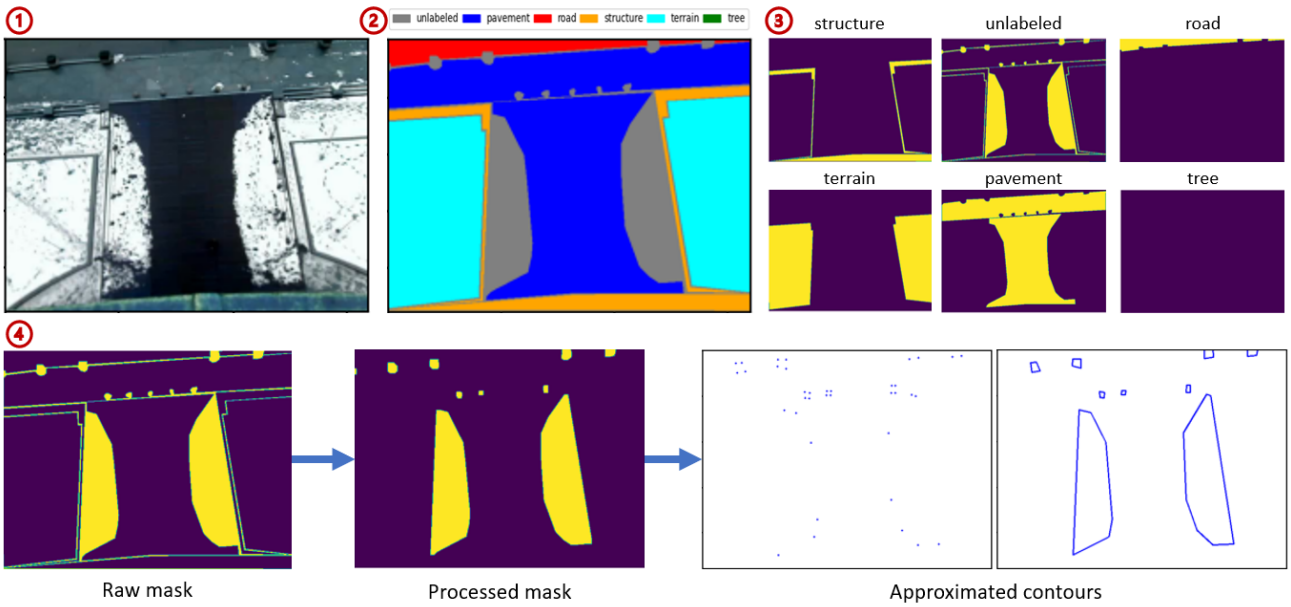

Figure 2. Illustration of the vectorization process of scene images. (1) the bird-eye-view image of the scene. (2) the segmented image using a U-Net network, with 6 semantic classes (unlabeled, pavement, road, structure, terrain, tree). (3) the individual segmentation masks. (4) the processing pipeline of a single mask to show the different processing steps. From left to right: the raw semantic mask, the processed mask after several morphological operations, the final contours (both dotted and solid) after approximation.

3. Material and Methods

Our proposed model has an encoder-decoder architecture. We use VectorNet as an encoder, which requires creating Vectorized representations from scene images since pedestrian's datasets don't provide HD maps. And for the decoder, we use a goal-based decoder that consists of two modules: a goal prediction module, and a trajectory completion module. More concretely, We use a modified version of DenseTNT as a goal prediction module responsible for predicting K end goals for each agent. We then feed those goals to PreTR that acts as a trajectory completion module, one for each predicted end goal. Let's first start by formulating mathematically the problem of trajectory prediction :

3.1. Trajectory prediction problem formulation

We formulate the trajectory prediction problem as follows: let N be the number of observed pedestrians in a given scene, T_{obs} be the observation length, T_{pred} be the prediction length, and $T = T_{obs} + T_{pred}$ the total length of the sequence. We denote an observed sequence $O = \{O_1, ..., O_{T_{obs}}\}$, such as $O_t = \{p_t^1, ..., p_t^N\}$ where $p_t^i \in \mathbb{R}^2$ is the 2D position of the agent i at the time step t, and $t \in \{1, 2, ..., T_{obs}\}$. Let's also denote $v_t^i \in \mathbb{R}^2$ the velocity of the agent i at the past timestep t, which is computed as $v_t^i = \frac{p_t^i - p_{t-1}^i}{t-(t-1)} = p_t^i - p_{t-1}^i$. Now, let X be the input sequence, then $X = \{X_2, X_3, ..., X_{T_{obs}}\}$ is a sequence of length $T_{obs}-1$ such as $X_t = \{(p_t^1, v_t^1), ..., (p_t^N, v_t^N)\}$. This means that for an agent i we have $\{(p_2^i, v_2^i), ..., (p_{T_{obs}}^i, v_{T_{obs}}^i)\}$, since v_1^i is undefined.

For the future trajectory, let's denote ground truth future sequence $Y = \{Y_1, Y_2, ..., Y_{T_{pred}}\}$ such as $Y_t = \{y_t^1, y_t^2, ..., y_t^N\}$, where $y_t^i \in \mathbb{R}^2$ is the ground truth future position of the agent i at the future time step t, and $t \in \{T_{obs}, T_{obs}+1, ..., T_{obs}+T_{pred}\}$. This means that y_t^i could also be seen as $p_{t+T_{obs}}^i$. Similarly, let's denote the predicted future sequence $\hat{Y} = \{\hat{Y}_1, \hat{Y}_2, ..., \hat{Y}_{T_{pred}}\}$ where $\hat{y}_t^i \in \mathbb{R}^2$ is the predicted future position of the agent i at the future time step t.

Finally, let's denote the generated sets of goal candidates $C = \{C^1, C^2, ..., C^N\}$ such that C^i is a discrete set of 2D points representing end goal candidates for the agent i. $C^i = \{c_1^i, c_2^i, ..., c_{m_i}^i\}$ where $c_j \in \mathbb{R}^2$ is a single goal candidate for the agent i. Learning to generate a heatmap for the agent i means learning the distribution $P(Y_{T_{pred}}^i|X)$. In other words, we aim to learn the function h such that $h(c_j^i) = P(Y_{T_{pred}}^i = c_j^i|X)$.

3.2. Vectorized representation of scene images

As previously mentioned, pedestrian's trajectory prediction datasets don't provide HD Maps, but only a bird-eye-view

RGB image of the scene, usually taken from a surveillance camera or a drone. Therefore, we propose to create vectorized representations from the semantic segmentation of the images, as shown in Figure 4.

A pretrained U-net (Ronneberger et al., 2015) segmentation network is used to compute semantic segmentation masks with 6 classes (unlabeled, pavement, road, structure, terrain, tree). Since this segmentation is at a pixel level, we need to process and clean those masks. Therefore, we apply some morphological operations (dilation, opening, erosion) to remove thin lines, noise, and unnecessary details. We then extract contours from the processed masks, and approximate these contours to reduce the small details and curves even further. These processing steps are show in (4) Figure 4, as well as in the algorithm 1.

Finally, each individual element or polygon (building, road, obstacle...etc) is assigned a unique id. We link dots of the same polygon with vectors of a predetermined length, to form a Polyline similar to the original VectorNet Figure 1.

Algorithm 1 Semantic masks processing

Input: a binary mask $m \in \{0,1\}^{HxW}$ representing one semantic class.
$num\_iters \leftarrow 2$
for $i \leftarrow 0$ to num_iters **do**
  $m \leftarrow$ dilation(m)
end for
$m \leftarrow$ opening(m)
for $i \leftarrow 0$ to num_iters **do**
  $m \leftarrow$ erosion(m)
end for
$contours \leftarrow$ findContours(m)
$contours \leftarrow$ approximate($contours$)
Return: $contours$

3.3. Goal prediction module

The core of our goal prediction module is based on DenseTNT (Gu et al., 2021). However, the latter was originally developed for predicting trajectories of vehicles and not of pedestrians. Hence, we'll be adapting its architecture to fit our use case. Just like the original DenseTNT, our model still follows the same two stage training strategy :

3.3.1. First stage: Heatmap generation

During this stage, the model will be trained to generate a heatmap of the end goals for each agent. As mentioned in section 2.2, the original method of generating this heatmap relied on the following steps: first predicting the most likely lanes (Lane Scoring), then generating Dense goals over this small subset of selected lanes, and finally computing the probability of each goal.

Unlike vehicles, pedestrians walk in a significantly less structured environment and follow no traffic rules. Thus, the Lane scoring step from the original architecture can't be applied for our use case. However, we can still apply a similar principle. The main goal of the lane selection process is to restrain the search space to a smaller subset of the map. This is done to avoid generating dense goals across an unnecessarily large area, allowing to reduce the computational cost. A similar idea was introduced in (Gilles et al., 2022) where the authors proposed to generate a very low resolution heatmap, and then iteratively increase the resolution in the high probability areas, until reaching the desired resolution.

Similarly, we start by generating *Sparse goals* at a low density α_{sparse} across the entire walkable are of the map. Then, we use the probability estimation module to predict the probability of each sparse goal candidate. Finally, we select the most likely sparse goals, and generate *Dense goals* with a high density α_{dense} around them. This two-steps heatmap generation idea is illustrated in Figure 3.

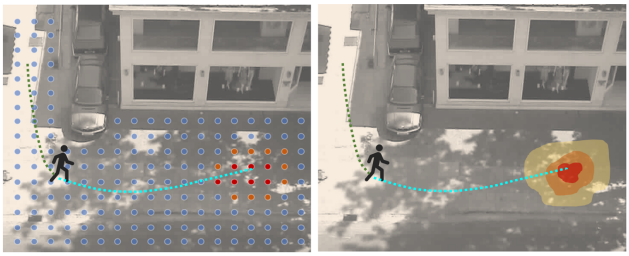

Figure 3. Illustration of the two steps heatmap generation idea. Note that this is just an illustration and not the actual output of the model. (left) Sparse goals generated with a low density, (right) Dense goals generated with a high density over the selected area.

Sparse goals selection. Let $C = \{c_1, c_2, ..., c_m\}$ be a discrete set of sparse goals generated uniformly across the entire walkable area of the map, Figure 3 (left). And $H = \{h(c_1), h(c_2), ..., h(c_m)\}$ the heatmap containing the sparse goals probabilities. We can apply the softmax to H, denoting $H_{softmax} = softmax(H)$, where $\sum_{i=1}^{m} h_{softmax}(c_i) = 1$. We can also sort the sparse goals candidates according to their probabilities from higher to lower, let's call the sorted set $\bar{C} = \{\bar{c}_1, \bar{c}_2, ..., \bar{c}_m\}$. Finally, we define the top selected sparse goals C_{top} as the smallest set of highest probability sparse goals with a CDF (Cumulative distribution function) of 0.90. More formally, $C_{top} = \{\bar{c}_1, \bar{c}_2, ..., \bar{c}_{j-1}, \bar{c}_j\}$ where :

$$\sum_{i=1}^{j-1} h_{softmax}(\bar{c}_i) \leq 0.90 \leq \sum_{i=1}^{j} h_{softmax}(\bar{c}_i) \quad (2)$$

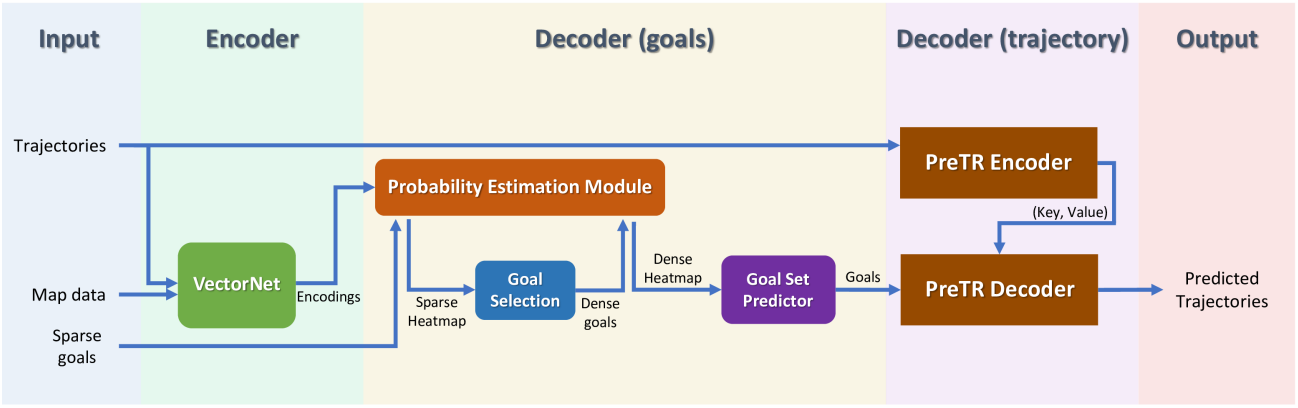

Figure 4. Illustration of the encoder-decoder architecture of our final model. It shows the three main component of the model being: First, the encoder, which is a vectornet backbone that generates global and subgraph encodings used then in the decoder. Second, the goal prediction module (decoder goals), and third, the trajectory completion module (decoder trajectory).

Probability estimation module. We use a similar probability estimation module to the original DenseTNT. Let C be a set of goal candidates, and Enc_{global}, Enc_{sub} be the global encodings and subgraph encodings, respectively, from VectorNet. Then the scores $H = h(c_1, ..., c_m)$ can be computed as follows :

$$Enc_{goals} = \varphi_1([\rho_1(C) \oplus Enc_{global}]) \quad (3)$$
$$Enc_{attn} = Attn(Enc_{goals}, Enc_{sub}, Enc_{sub}) \quad (4)$$
$$Enc = [Enc_{global} \oplus Enc_{goals} \oplus Enc_{attn}] \quad (5)$$
$$Scores = \rho_2([Enc \oplus \varphi_2(Enc)]) \quad (6)$$
$$H = \text{LogSoftmax}(Scores) \quad (7)$$

Where φ_1, φ_2 are MLP, and ρ_1, ρ_2 are fully connected layers.

3.3.2. Second stage: Goal prediction

After obtaining the heatmap, we need to predict or sample K goals that are both highly probable and diverse, covering all different modes of the heatmap. Some approaches suggested sampling techniques that insures diversity in the sampled goals, such as a clustering-based sampling (Mangalam et al., 2020a), or a learned sampling method (Gilles et al., 2021). However, in the original DenseTNT, the authors introduced a goal set prediction module that predicts a diverse set of goals given the heatmap. We decided to go with the same module, but change its training strategy.

The problem with training a goal set prediction module is that we don't have multi-future ground truth to compare with, but rather only one end goal ground truth. That's why in DenseTNT, the authors suggested an offline optimization-based algorithm that generates multi-future pseudo-labels, as explained in section 2.2. These pseudo-labels are then compared with the output of the model, to compute the loss and optimize the network. However, the proposed algorithm has a complexity of $O(K \times \lambda \times m)$ where K is the size of the goal set, λ is the number of performed iterations during the optimization, and m is the total number of goal candidates. This makes the training of the second stage of DenseTNT slow and computationally expensive.

We decided to drop this optimization-based algorithm, and replace the pseudo-labels loss with a Winner-Takes-All loss in order to train the goal set prediction module more efficiently.

Winner-Takes-All Loss. this loss is computed similarly to d in Equation 1. It consists of minimizing the minimum displacement error between a set of goals, and the ground truth goal. More formally :

$$\mathcal{L}_{\text{wta}} = \min_{c_i \in \hat{C}} \left\| c_i - y_{T_{pred}} \right\| \quad (8)$$

where $\hat{C}$ is the predicted set of goals, and $y_{T_{pred}}$ is the ground truth end goal.

3.4. Trajectory completion module

Once we predicted the end goals of an agent, We use PreTR to generate the trajectories leading to those goals. As explained in section 2.3, PreTR is a transformer-based model that can generate a single deterministic trajectory of an agent given its past observations. Our strategy is to feed the K predicted goals to PreTR, so that it generates K trajectories, one for each goal. In other words, we aim to condition the trajectory generation process in PreTR on an end goal, which allows us to obtain multimodal trajectory predictions from a deterministic model.

In order to condition the trajectory generation process in

PreTR on a given goal, we use teacher forcing during the training, feeding it encodings of the ground truth end goal. This teaches the model to generate a trajectory that ends on, or close to, the provided goal. Then we feed it encodings of the predicted goals during testing. Despite this strategy resulting in trajectories that end at the provided goals, we find that the way we generate goals' encodings has a significant impact on the quality of the generated trajectories, and that only encoding the 2D coordinates of the goals isn't enough to generate coherent trajectories.

3.4.1. Feeding goals encodings to PreTR

In order to guide the trajectory generation, We leverage the parallel decoding in PreTR. As previously mentioned, PreTR is a transformer-based model that uses a parallel decoding technique, in which a set of learned object queries is forwarded through the decoder to be refined. Each one of those queries will then result in a future position of an agent, allowing the model to generate all future positions of the agents in a non-autoregressive manner. We concatenate a goal encoding to each of these queries, before forwarding them through the decoder as shown in Figure 5.

More formally, let $l_t^i \in \mathbb{R}^{dim}$ be a learned query associated with the agent i and the future time step t, that will be refined to $\hat{y}_t^i$. Then $\{l_1^i, l_2^i, ..., l_{T_{pred}}^i\}$ are queries of the agent i. Finally, let $\{e_1^i, ..., e_{T_{pred}}^i\}$ be the goal's encodings of the goal g_i and the agent i, which computation is explained in the next section 3.4.2. We concatenate to each query l_t^i a goal encoding e_t^i, then encode them through an MLP and forward it to the decoder.

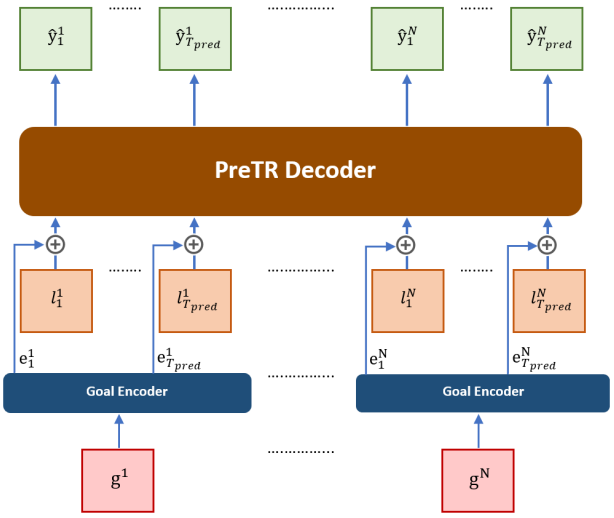

Figure 5. Illustration of feeding the goals' encodings to the decoder of PreTR, where g^i are the goals, e_t^i, are the goals' encodings, l_t^i are the learned queries, and $\hat{y}_t^i$ are the predicted future positions.

3.4.2. Computing goals encodings

Our goal encoder generates the goal encoding e_t^i given a predicted goal g^i of an agent i, its last observed position $o_{T_{obs}}^i$, and a future time step t. In other words :

$$e_t^i = \psi_{goal}(g^i, o_{T_{obs}}^i, t) \tag{9}$$

Such as the goal encoder ψ_{goal} can be expressed as:

$$\psi_{goal}(g^i, o_{T_{obs}}^i, t) = \varphi([g^i \oplus v_t^i \oplus \tau_t^i]) \tag{10}$$

where φ is an MLP, $v_t^i \in \mathbb{R}$ and $\tau_t^i \in \mathbb{R}^2$ represent the desired velocity, and an intermediate point, respectively.

Desired velocity. It represents the expected velocity of the agent at the future time step t. Since we don't predict the trajectory iteratively, we can't use the previous velocity v_{t-1}^i to estimate v_t^i. Therefore, we have to make the assumption that the agent keeps a constant velocity during its future trajectory, thus, $v_1^i = v_2^i = ... = v^i$, and v_t^i can be computed as:

$$v^i = \frac{\left\| g^i - o_{T_{obs}}^i \right\|}{T_{pred}} \tag{11}$$

Intermediate point. It represents the expected future position of the agent at future timestep t. This position is generated based on the assumption that future velocity is constant, and the agent is walking in a straight line. Due to these strong assumptions, we expect an important discrepancy between the intermediate point and the true future position at timestep t. However, this still provides the model with a general idea of the expected distance of the agent from the end goal at timestep t, as well as a general idea of the future position allowing it to focus on learning the offset. The intermediate point τ_t^i is computed as:

$$\tau_t^i = o_{T_{obs}}^i + t \times \frac{g^i - o_{T_{obs}}^i}{T_{pred}} \tag{12}$$

3.5. Loss function

First stage. similar to DenseTNT, we use the same loss term for training both the sparse and the dense goals. It's a binary cross-entropy loss between the predicted goal scores H_{pred}, and the ground goal scores H_{gt} :

$$\mathcal{L}_{goal} = \mathcal{L}_{CE}(H_{pred}, H_{gt}) \tag{13}$$

where the ground truth score of the goal closest to the final position is 1, and the others are 0.

Thus, the first stage loss is :

$$\mathcal{L}_{first} = \mathcal{L}_{goal\_sparse} + \mathcal{L}_{goal\_dense} \tag{14}$$

Second stage. The second stage loss is a combination of two losses: The first being the goal module loss, which is

the Winner-Takes-All loss presented in equation 8. And the second being the trajectory completion loss of PreTR, which is an MSE loss between the predicted trajectory and the ground truth trajectory expressed as follows :

$$\mathcal{L}_{MSE} = \frac{1}{N \times T_{pred}} \sum_{i=1}^{N} \sum_{t=1}^{T_{pred}} (\hat{y}_t^i - y_t^i)^2 \qquad (15)$$

Thus, the loss for the second stage is :

$$\mathcal{L}_{second} = \mathcal{L}_{wta} + \mathcal{L}_{MSE} \qquad (16)$$

3.6. Evaluation

Datasets. We trained and evaluated our model on the ETH/UCY datasets (Yuan et al., 2017) (Lerner et al., 2007), which are widely adapted benchmark datasets for pedestrian multi-agent trajectory prediction. They're made of 5 different datasets (Eth, Univ, Hotel, Zara1, Zara2). These datasets contain a total of 1536 pedestrians, captured with a bird's eye view perspective at 4 different locations, and annotated at 2.5 FPS (frame per second).

Evaluation Protocol. Following the common evaluation protocol on these datasets, we adopt the leave-one-out cross-validation strategy. This strategy consists of training the model on four of the datasets and testing it on the one left dataset. Also following the standards in this benchmark, We predict 20 trajectories for each agent ($K = 20$), each trajectory has a past observation length of 8 timesteps ($T_{obs} = 8$), and a future prediction length of 12 timesteps ($T_{pred} = 12$), making the total trajectory length is 20 timesteps ($T = 20$).

Metrics. We report the two common metrics used for the pedestrian trajectory evaluation task :

- Minimum Average Displacement Error (minADE) :

$$\text{minADE} = \min_{\hat{y} \in \hat{\mathbf{y}}} \frac{1}{T_{pred}} \sum_{t=1}^{T_{pred}} \|\hat{y}_t - y_t\| \qquad (17)$$

- Minimum Final Displacement Error (minFDE) :

$$\text{minFDE} = \min_{\hat{y} \in \hat{\mathbf{y}}} \left\|\hat{y}_{T_{pred}} - y_{T_{pred}}\right\| \qquad (18)$$

where $\hat{\mathbf{y}}$ is a set of K predicted trajectories, and y is the ground truth trajectory.

3.7. Implementation details

Goals generation details. We generate sparse goals with a density α_{sparse} of 1m, and dense goals with a density α_{dense} of 0.25m. We only generate sparse goals inside the walkable semantic classes, that we consider to be *pavement*, *terrain*, and *tree*. When generating dense goals, we generate them within a range of 2m around the selected top sparse goals. We also preprocess the dense heatmap in the second stage before predicting the goals to save on the memory, we keep top dense goals with a CDF of 0.90, and a minimum of 200 goals and a maximum of 400.

Data processing details. Regarding the processing of the data, we normalize the scene data, centering around the barycenter of last observations of all agents in the scene. We also apply random rotations to different scene for data augmentation. We also pad the agents in each scene, with a maximum number of agents of 60 for the Univ dataset, and 20 for the others. We use a skip window of size 1 for the trajectories similar to (Mangalam et al., 2020a), meaning that an agent with a trajectory length of 25, will be represented as 5 agents of trajectory length 20.

Training details. Regarding the training, we train on an NVIDIA A100 GPU. We train the first stage for 50 epochs, and the second stage for 16 epochs. We use a learning rate 0.001, and a batch size of 16.

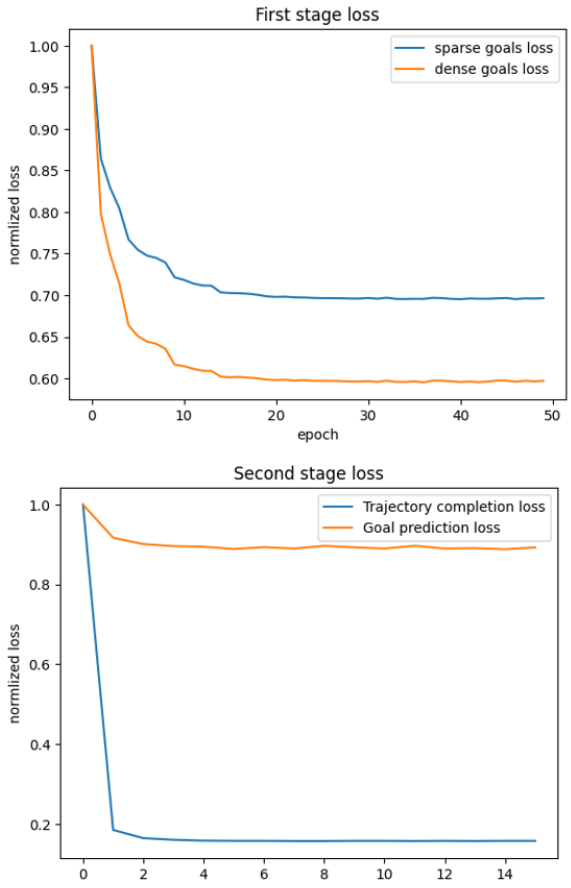

Figure 6. The loss of the first and second stage of our model. We normalize each loss for better visualization of the curves.

Evaluation Metrics: $\text{minADE}_{20} \downarrow$ / $\text{minFDE}_{20} \downarrow$ [meters]						
Method	**ETH**	**HOTEL**	**UNIV**	**ZARA1**	**ZARA2**	**AVG**
Social-LSTM (Alahi et al., 2016)	1.09/2.35	0.79/1.76	0.67/1.40	0.47/1.00	0.56/1.17	0.72/1.54
Social-GAN (Gupta et al., 2018)	0.81/1.52	0.72/1.61	0.60/1.26	0.34/0.69	0.42/0.84	0.58/1.18
Goal-GAN (Dendorfer et al., 2020)	0.59/1.18	0.19/0.35	0.60/1.19	0.43/0.87	0.32/0.65	0.43/0.85
ST-GAT (Huang et al., 2019)	0.65/1.12	0.35/0.66	0.52/1.10	0.34/0.69	0.29/0.60	0.43/0.83
MG-GAN (Dendorfer et al., 2021)	0.47/0.91	0.14/0.24	0.54/1.07	0.36/0.73	0.29/0.60	0.36/0.71
Transformer-TF (Giuliari et al., 2020)	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 (Yu et al., 2020)	0.36/0.65	0.17/0.36	0.31/0.62	0.26/0.55	0.22/0.46	0.26/0.53
PECNet (Mangalam et al., 2020b)	0.54/0.87	0.18/0.24	0.35/0.60	0.22/0.39	0.17/0.30	0.29/0.48
Trajectron++ (Salzmann et al., 2021)	0.39/0.83	0.12/0.21	0.20/0.44	0.15/0.33	0.11/0.25	0.19/0.41
AgentFormer (Yuan et al., 2021)	0.45/0.75	0.14/0.22	0.25/0.45	0.18/0.30	0.14/0.24	0.23/0.39
Goal-SAR (Chiara et al., 2022)	0.28/0.39	0.12/0.17	0.25/0.43	0.17/0.26	0.15/0.22	0.19/0.29
Y-net (Mangalam et al., 2020a)	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 (Yue et al., 2023)	0.25/0.24	0.09/0.13	0.21/0.38	0.16/0.27	0.12/0.20	0.17/0.24
Ours	0.52/0.65	0.17/0.24	0.32/0.50	0.24/0.37	0.21/0.31	0.30/0.42
$\pm$ *std*	*± 0.01/0.03*	*± 0.00/0.01*	*± 0.01/0.01*	*± 0.00/0.01*	*± 0.00/0.01*	*± 0.00/0.01*
Ours (offline)	0.54/0.69	0.17/0.27	0.32/0.49	0.26/0.39	0.22/0.34	0.30/0.44
$\pm$ *std*	*± 0.01/0.04*	*± 0.00/0.00*	*± 0.00/0.00*	*± 0.01/0.01*	*± 0.02/0.01*	*± 0.00/0.01*

Table 1. Results on ETH/UCY. We report results considering the minimum ADE/FDE of 20 predicted samples. We run each of our experiments 5 times and report the mean $\pm$ std (0.00 indicates $std < 0.005$). **Ours** denotes our proposed model with the WTA loss, whereas **Ours (offline)** denote the model with the original pseudo-labels loss based on the offline optimization-based algorithm.

4. Results

Table 1 shows our results on the ETH/UCY datasets. We run each of our experiments 5 times and report the mean $\pm$ std. **Ours** denotes our proposed model with the WTA loss, whereas **Ours (offline)** denote the model with the original pseudo-labels loss based on the offline optimization-based algorithm. We separate **Goal-SAR**, **Y-net**, and **NSP-SFM** from the rest because these three methods are using the same exact CNN-based goal prediction module originally proposed in Y-net.

Quantitative results. We notice that all three CNN-based methods outperform our model on all datasets, which is mainly due to the goal prediction performance of the Y-net module. Using a CNN does provide several advantages such as being able to generate a higher resolution heatmap, as well as encoding both the scene information and the trajectory in the same pixel dimension, keeping their alignment and spatial features.

Nevertheless, when comparing the minFDE with other non CNN-based methods, we see that on average, our model performs on par with the other approaches. Especially, we notice that our model outperforms those methods by a significant margin when it comes to difficult datasets such as ETH where the displacement error of all methods is quite high. However, it struggles with easier datasets such as ZARA1 and ZARA2 where all methods tend to perform well, and the displacement error gets to lower levels. Finally, we notice that our model (using WTA loss) and the version with the offline optimization algorithm (using the original pseudo-labels loss) performs similarly.

Limitations. A limitation of our approach is the resolution of the heatmap, where we only generate goals with a density of 0.25m. This means our performance is expected to fall as we get to high precision levels in the prediction, such as in ZARA1 and ZARA2. However, having such high precision doesn't offer real advantages for Autonomous vehicles applications, where the goal is to predict the general movements and directions of the pedestrian.

Efficiency. We compare the efficiency of our model using vectorized representations with Goal-SAR (Chiara et al., 2022), which is a very simple model that uses the popular CNN-based Y-net goal prediction along with a basic attention-based trajectory completion module. We also compare our model with the one using the offline optimization algorithm. Table 2 shows training and inference time for two test datasets: UNIV, ZARA1.

Model	**Training**		**Inference**	
	UNIV	ZARA1	UNIV	ZARA1
Goal-SAR	16:54:45	15:33:48	02:12:16	00:06:10
Ours (offline)	04:36:58	08:38:04	00:04:55	00:00:28
Ours	02:31:15	04:49:40	00:04:52	00:00:26

Table 2. Training and Inference time on two example datasets UNIV and ZARA1. The time is expressed in hh:mm:ss, and was measured on an NVIDIA A100 GPU.

From Table 2, we can notice the advantage that the vectorized approach has over CNN-based approach's on both training and inference. Furthermore, we can also notice the faster training of our model compared to the version using the offline optimization algorithm with the pseudo-labels loss. This faster training didn't result in any drop in performance as we've previously seen in Table 1.

When only looking at the training time of the second stage, we can further see the impact of replacing the pseudo-labels loss with the WTA loss. Our second stage training is more than 85% faster on average, as illustrated in Table 3.

Second Stage	Training	
	UNIV	ZARA1
with pseudo-labels loss	02:26:28	04:30:24
with WTA loss	00:20:45	00:42:00

Table 3. Training time of only the second stage on two example datasets. The time is expressed in hh:mm:ss, and was measured on an NVIDIA A100 GPU.

Ablation study. As we mentioned when discussing the PreTR integration in our architecture, the quality of the generated trajectories is very sensitive to how we compute the goals' encodings, due to the non-autoregressive generation of the trajectory. Thus, simply encoding the 2D position of the end goal doesn't result in a coherent trajectory. Therefore, we generate goals' encodings using: the goals, the desired velocity, and the intermediate points.

Here we investigate the contribution of both desired velocity and intermediate points on the quality of the generated trajectory. We do so by measuring the impact of removing these elements from the goals' encodings on the minADE. As shown in Table 4, intermediate points have a significant impact on the generated trajectory, unlike desired velocity. However, We also see that encoding both of them will consistently yield the best results.

	Ours w/o interm	Ours w/o vel	Ours
ETH	0.59	0.56	**0.52**
HOTEL	0.19	**0.17**	**0.17**
UNIV	0.52	0.36	**0.32**
ZARA1	0.27	0.26	**0.24**
ZARA2	0.22	**0.21**	**0.21**
AVG	0.36	0.31	**0.30**

Table 4. Ablation study to measure the impact of encoding the intermediate points (interm) and the desired velocity (vel) on the quality of the generated trajectory. We use the $\text{minADE}_{20} \downarrow$ to evaluate the quality of the generated trajectories.

Visualizations. Figure 7 illustrates the output of our model at each step of the generation process: sparse goals, dense heatmap, goal prediction, and finally trajectory generation.

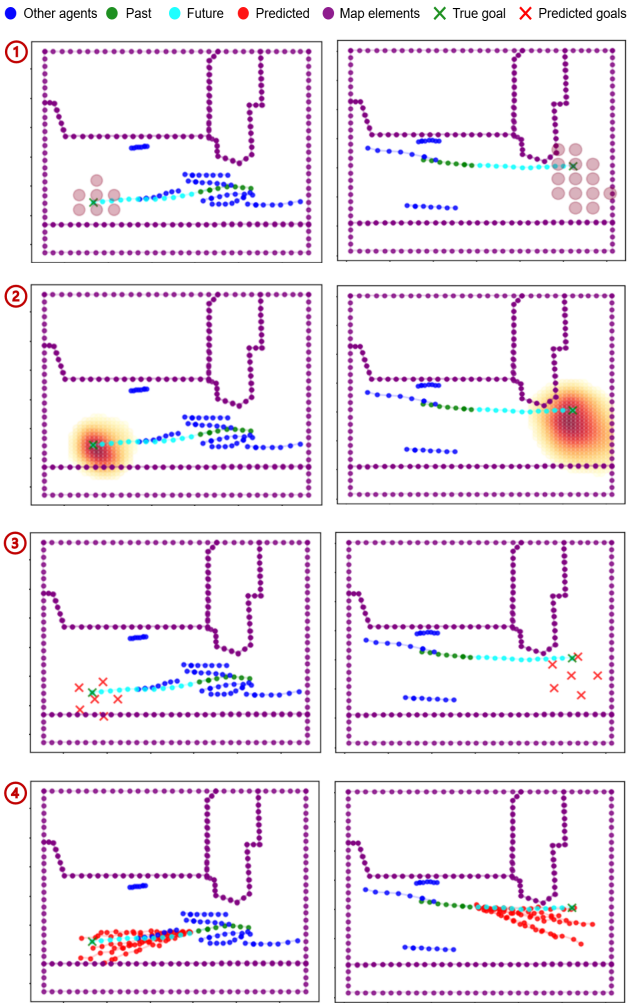

Figure 7. Visualization of the output of our model on the ZARA1 dataset at different generation steps: (1) sparse heatmap and sparse goals selection, (2) dense heatmap generation, (3) goal prediction from the dense heatmap, (4) trajectory generation based on the predicted goals. For the purpose of visualization, we predict 6 trajectories instead of the 20 used for the evaluation.

5. Conclusion

In this work, we tackle the problem of multimodal pedestrian trajectory predictions using goal-based methods. We try to replace the standard state-of-the-art CNN-based methods, by a more efficient and less computationally intensive approach. Thus, we borrow the concept of vectorized representations, wildly adopted in the field of vehicle trajectory prediction, and explore its applicability to pedestrian trajectory prediction. We show the substantial advantage that our proposed model has in terms of efficiency for both training and inference compared to CNN-based approaches. However, we notice an important performance gap with Y-net goals predictions. The average displacement error for

pedestrian trajectory predictions tends to be smaller, and the predictions tend to be done at a higher level of precision compared to vehicle predictions. In addition, unlike vehicles, pedestrians navigate a less structured environment that allows for free movements and no traffic rules. Further work needs to be done to adjust this vectorized representation concept and better take into account the fundamental differences between vehicles' and pedestrians' predictions.

References

Achaji, L., Barry, T., Fouqueray, T., Moreau, J., Aioun, F., and Charpillet, F. Pretr: Spatio-temporal non-autoregressive trajectory prediction transformer, 2022.

Alahi, A., Goel, K., Ramanathan, V., Robicquet, A., Fei-Fei, L., and Savarese, S. Social lstm: Human trajectory prediction in crowded spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 961–971, 2016. doi: 10.1109/CVPR.2016.110.

Chang, M.-F., Lambert, J., Sangkloy, P., Singh, J., Bak, S., Hartnett, A., Wang, D., Carr, P., Lucey, S., Ramanan, D., and Hays, J. Argoverse: 3d tracking and forecasting with rich maps, 2019.

Chiara, L. F., Coscia, P., Das, S., Calderara, S., Cucchiara, R., and Ballan, L. Goal-driven self-attentive recurrent networks for trajectory prediction, 2022.

Dendorfer, P., Ošep, A., and Leal-Taixé, L. Goal-gan: Multimodal trajectory prediction based on goal position estimation, 2020.

Dendorfer, P., Elflein, S., and Leal-Taixé, L. Mg-gan: A multi-generator model preventing out-of-distribution samples in pedestrian trajectory prediction, 2021.

Gao, J., Sun, C., Zhao, H., Shen, Y., Anguelov, D., Li, C., and Schmid, C. Vectornet: Encoding hd maps and agent dynamics from vectorized representation, 2020.

Ghazvininejad, M., Levy, O., Liu, Y., and Zettlemoyer, L. Mask-predict: Parallel decoding of conditional masked language models, 2019.

Gilles, T., Sabatini, S., Tsishkou, D., Stanciulescu, B., and Moutarde, F. Home: Heatmap output for future motion estimation, 2021.

Gilles, T., Sabatini, S., Tsishkou, D., Stanciulescu, B., and Moutarde, F. Thomas: Trajectory heatmap output with learned multi-agent sampling, 2022.

Giuliari, F., Hasan, I., Cristani, M., and Galasso, F. Transformer networks for trajectory forecasting, 2020.

Gu, J., Sun, C., and Zhao, H. Densetnt: End-to-end trajectory prediction from dense goal sets, 2021.

Gupta, A., Johnson, J., Fei-Fei, L., Savarese, S., and Alahi, A. Social gan: Socially acceptable trajectories with generative adversarial networks, 2018.

Helbing, D. and Molnár, P. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, may 1995. doi: 10.1103/physreve.51.4282. URL https://doi.org/10.1103%2Fphysreve.51.4282.

Huang, Y., Bi, H., Li, Z., Mao, T., and Wang, Z. Stgat: Modeling spatial-temporal interactions for human trajectory prediction. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6271–6280, 2019. doi: 10.1109/ICCV.2019.00637.

Lerner, A., Chrysanthou, Y., and Lischinski, D. Crowds by Example. *Computer Graphics Forum*, 2007. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2007.01089.x.

Liang, M., Yang, B., Hu, R., Chen, Y., Liao, R., Feng, S., and Urtasun, R. Learning lane graph representations for motion forecasting, 2020.

Mangalam, K., An, Y., Girase, H., and Malik, J. From goals, waypoints paths to long term human trajectory forecasting, 2020a.

Mangalam, K., Girase, H., Agarwal, S., Lee, K.-H., Adeli, E., Malik, J., and Gaidon, A. It is not the journey but the destination: Endpoint conditioned trajectory prediction, 2020b.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation, 2015.

Salzmann, T., Ivanovic, B., Chakravarty, P., and Pavone, M. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data, 2021.

Thanh-Tung, H. and Tran, T. On catastrophic forgetting and mode collapse in generative adversarial networks, 2020.

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

Yuan, Y., Lu, Y., and Wang, Q. Tracking as a whole: Multi-target tracking by modeling group behavior with sequential detection. *IEEE Transactions on Intelligent Transportation Systems*, 18(12):3339–3349, dec 2017. doi: 10.1109/tits.2017.2686871. URL https://doi.org/10.1109%2Ftits.2017.2686871.

Yuan, Y., Weng, X., Ou, Y., and Kitani, K. Agentformer: Agent-aware transformers for socio-temporal multi-agent forecasting, 2021.

Yue, J., Manocha, D., and Wang, H. Human trajectory prediction via neural social physics, 2023.