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, 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 the training time of its second stage while improving 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 its surrounding complex environment. One of those algorithms is trajectory predictions, which allow the system to forecast future trajectories of different surrounding agents (vehicles, pedestrians, cyclists...etc) giving their past observed locations. This plays an important role in planning future movements, anticipate risk, and avoid 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 behavior 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 predictions.

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 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 became 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., 2020). 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 unnecessary 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 very 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 performance and 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, are precise at a centimetre 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).

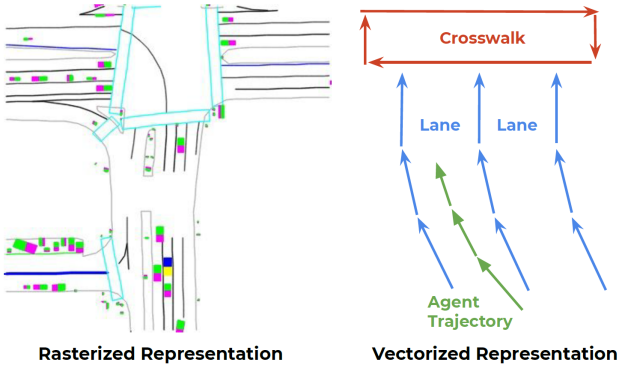

Figure 1. Illustration from the VectorNet paper (Gao et al., 2020). (left) rasterized rendering of an HD map, (righ) vectorized approach to represent an HD map and agent trajectories.

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 a the 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 trajectory 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 benchmarkn 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 step 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 a 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 goals candidates in the current set, moving them around slightly, and then we keep the new set if it has a smaller expected error than the current one. 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 gets 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 encoding, whereas the decoder receives a set of learned queries. Both are forwarded through multiple blocks of attention modules that operates 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 trajectory.

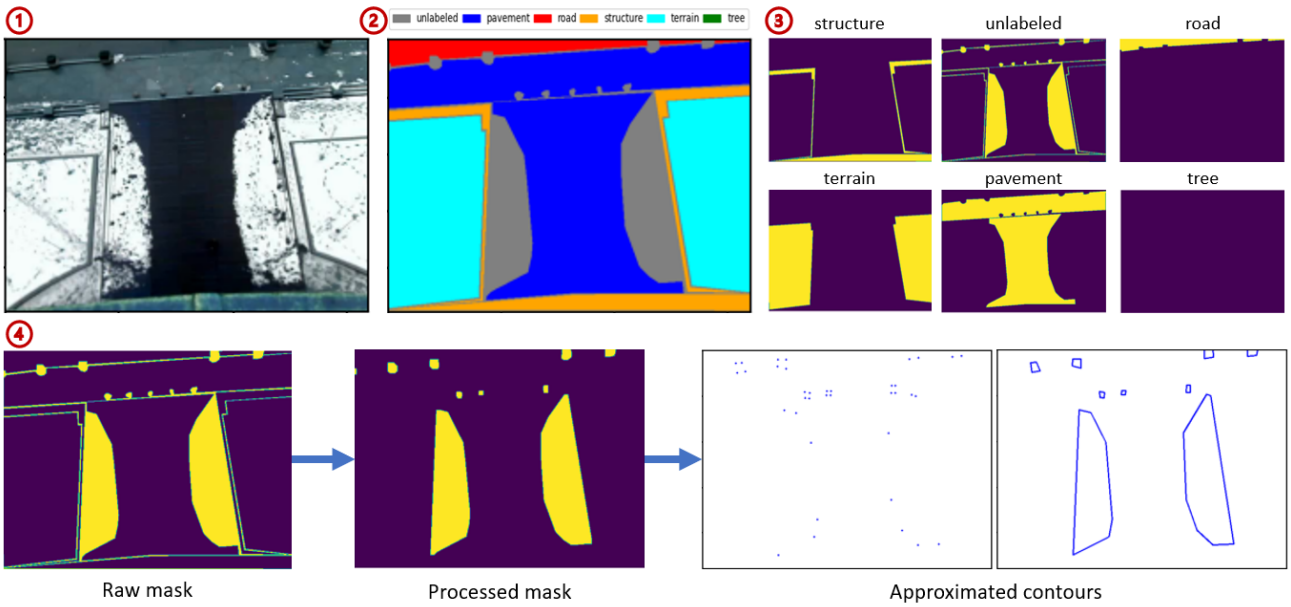

Figure 2. Illustration of the vectorization process of scene images. (1) the bird-eye-view image of the scene. (2) the segmented image using a U-Net network, with 6 semantic classes (unlabeled, pavement, road, structure, terrain, tree). (3) the individual segmentation masks. (4) the processing pipeline 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 trajectory completion module. More concretely, We use a modified version of DenseTNT as a goal prediction module responsible of predicting K end goals for each agent. We then feed those goals to PreTR that acts as trajectory completion module, generating K future trajectory per agent, one for each predicting 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, and T_{pred} be the prediction length, and $T = T_{obs} + T_{pred}$ the total length of the sequence. We denote an observed sequence $O = \{O_1, O_2, ..., O_{T_{obs}}\}$, such as $O_t = \{p_t^1, p_t^2, ..., p_t^N\}$ 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 time step t computed as $v_t^i = p_t^i - p_{t-1}^i$. 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 $X_t = \{(p_t^1, v_t^1), (p_t^2, v_t^2), ..., (p_t^N, v_t^N)\}$. This means that for an agent i, we have $\{(p_2^i, v_2^i), ..., (p_{T_{obs}}^i, v_{T_{obs}}^i)\}$ because 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 of 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 compute 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 principal. The main goal of the lane selection process was to restrain the search space on 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 propose to generate a very low resolution heatmap, and then iteratively increasing the resolution in the high probability areas, until reaching the desired goal density.

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 idea two steps heatmap generation idea is illustrated in Figure 3.

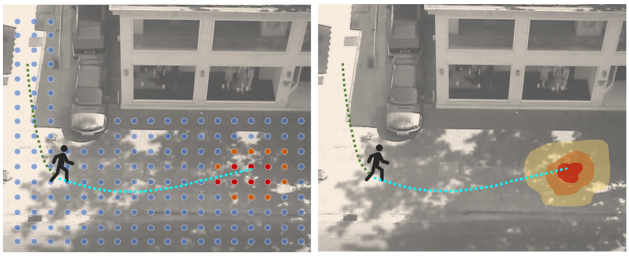

Figure 3. Illustration of the two steps heatmap generation idea. Note that this is just an illustration and not the actual output of the model. (left) Sparse goals generated with a low density, (righ) Dense goals generated with a high density over the selected area.

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 the highest probability sparse goals with a CDF (Cumulative distribution function) of 0.90, with a minimum and maximum number of goals. More formally, $C_{top} = \{\bar{c}_1, \bar{c}_2, ..., \bar{c}_{j-1}, \bar{c}_j\}$ where :

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

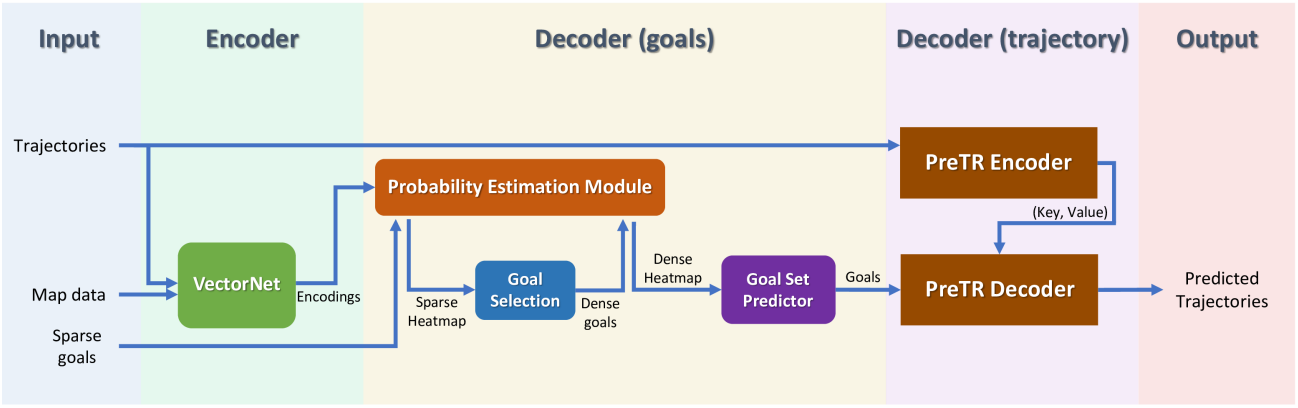

Figure 4. Illustration of the encoder-decoder architecture of our final model. It shows the three main component of the model being: First, the encoder, which a a vectornet backbone that generates global and subgraph encodings used then in the decoder. Second, the goal prediction module (decoder goals), and 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., 2020), 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 end goal ground truth. That's why in DenseTNT, the authors suggested an 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 iteration 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 trajectory 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 ends 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.

We'll start by explaining how we feed the goals' encodings to PreTR, then we'll come back to how we compute those encodings :

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, and $\{l_1^1, ..., l_{T_{pred}}^1, ..., l_1^N, ..., l_{T_{pred}}^N\}$ is the entire set of queries. 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.

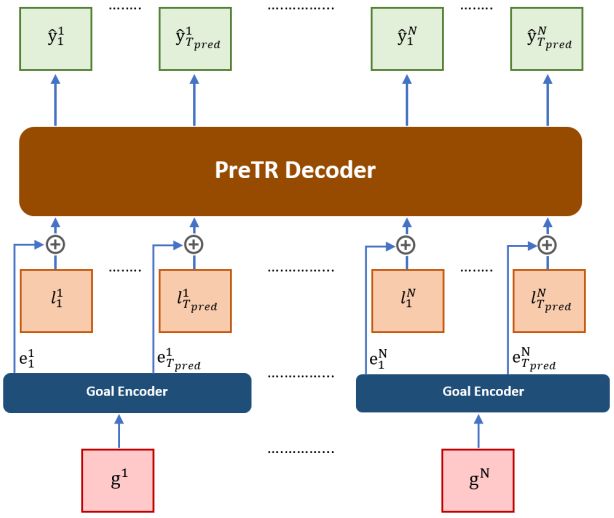

Figure 5. Illustration of feeding the goals' encodings to the decoder of PreTR, where g^i are the goals, e_t^i, are the goals' encodings, l_t^i are the learned queries, and $\hat{y}_t^i$ are the predicted trajectories.

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:

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

where φ is an MLP, $v_t^i \in \mathbb{R}$ and $d_t^i \in \mathbb{R}^2$ represent the desired velocity, and the 2D distance vector, respectively.

Desired velocity. It represents the expected velocity of the agent at the future time step t. Since we don't predict the trajectory attractively, 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 its velocity during its future trajectory, thus, $v_1^i = v_2^i = ... = v^i$, and v_t^i can be computed as:

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

Distance vector. It represents the expected distance, in both 2D coordinates, from the goal g_i at the future time step i. Once again we have to assume the constant future velocity of the agent because of the non interactive prediction. Therefore, we compute it as follow:

$$d_t^i = \frac{g^i - o_{T_{obs}}^i}{T_{pred}} \times (T_{pred} - t) \tag{12}$$

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.

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.

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.

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, 2020.

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

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

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