AgentFormer: Agent-Aware Transformers for Socio-Temporal Multi-Agent Forecasting Supplementary Material

Ye Yuan[1] Xinshuo Weng[1] Yanglan Ou[2] Kris Kitani[1]
[1]Carnegie Mellon University [2]Penn State University
https://www.ye-yuan.com/agentformer

Contents

1. Handling a Time-Varying Number of Agents

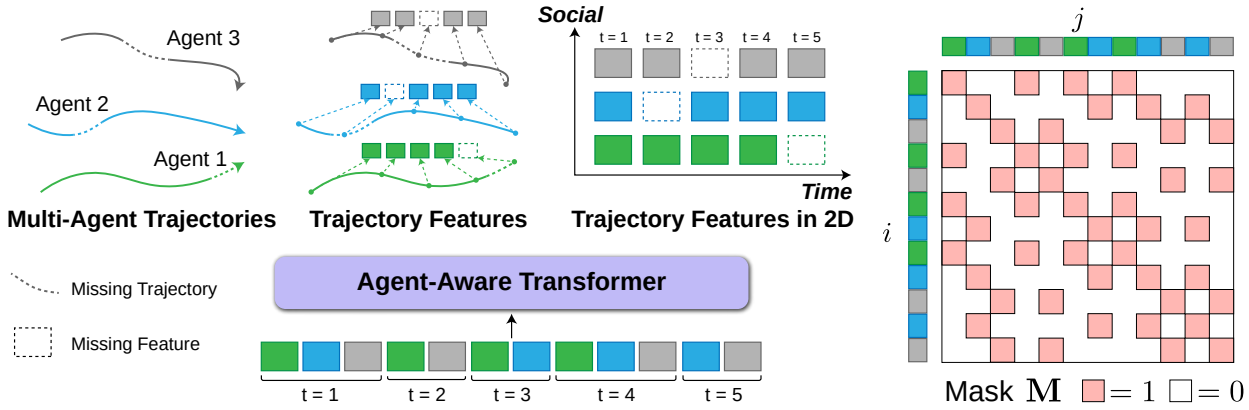

Figure 1. Our method can naturally handle a time-varying number of agents because of the flexible sequence representation of multi-agent trajectories. We can simply remove the trajectory features of missing agents at each timestep from the sequence. The mask $\mathbf{M}$ of the example sequence (when applying self-attention) is computed based on the agreement of agent identity between each query and key.

For clarity and ease of exposition, we assume the number of agents remains the same across timesteps in the main paper. However, this assumption is not necessary, and our method can easily generalize to use cases where the number of agents changes over time due to agents going out of the scene or being missed by detection. We illustrate how to apply our method to such cases in Fig. 1. Owning to the flexible sequence representation we employ for multi-agent trajectories, we can simply remove the features of missing agents at each timestep from the sequence. The reason why we do not need to fill the missing features is that our method uses time encoding to preserve time information, unlike RNNs which have to use recurrence to encode timesteps and thus necessitate the features of all timesteps. As the number of agents is no longer N for all timesteps,

the computation of the mask $\mathbf{M}$ in agent-aware attention needs to be changed accordingly:

$$M_{ij} = \mathbb{1}(\text{Agent}(i) = \text{Agent}(j)) \tag{1}$$

where $\text{Agent}(\cdot)$ extracts the agent index of a query/key and $\mathbb{1}(\cdot)$ denotes the indicator function. An example of mask $\mathbf{M}$ is shown in Fig. 1 (Right).

2. Additional Implementation Details

Encoding Semantic Maps. The semantic map $\mathbf{I}_n \in \mathbb{R}^{H \times W \times C}$ for each agent n has spatial dimensions $(100, 100)$ with 3 meters between adjacent pixels. It has $C = 3$ channels annotating drivable areas, road dividers, and lane dividers obtained using the official nuScenes software development kit. Since the semantic map is relatively easy to parse, we use a simple hand-designed CNN to extract visual features $\mathbf{v}_n$ from it. In particular, the CNN has four convolutional layers with channels (32, 32, 32, 1), kernel size (5, 5, 5, 3), and strides (2, 2, 1, 1). A final linear layer is used to obtain a 32-dimensional feature.

Training Trajectory Sampler. The scaling factor σ_d in the trajectory sampler loss L_{samp} (Eq. (9) in the main paper) is set to 5 for ETH/UCY and 20 for nuScenes. We clip the maximum value of the KL term in L_{samp} down to 2. We train the trajectory sampler using the Adam optimizer [1] for 50 epochs on ETH/UCY and nuScenes. We use an initial learning rate of 10^{-4} and halve the learning rate every 5 epochs.

Ablation Study Details. We first provide details for the ablation study of separate social and temporal models (first group of Table 3 and 4 in the main paper). We first use a temporal model (LSTM or Transformer) to extract the temporal feature of each agent and then apply a social model (GCN [2] or Transformer) over the temporal features to obtain social features for each agent; final trajectories are decoded from the social features using either an LSTM or Transformer. For the GCN, we use two graph convolutional layers with channels (256, 256) and residual connections within each layer. The hidden dimensions of the LSTMs are set to 256. The Transformers have two layers with key/query dimensions 256 and 8 heads; the feedforward layer has 512 hidden units, and the dropout ratio is 0.1. We use the positional encoding [4] for the temporal Transformer but not for the social Transformer as agents are permutation-invariant.

Next, we provide details for the ablation study of each key technical component (second group of Table 3 and 4 in the main paper). For the variant without joint latent modeling ("w/o joint latent"), we append the latent codes to the trajectory sequence after the AgentFormer decoder instead of before the decoder. In this way, the latent code of one agent will not affect the future trajectory of another agent. For the variant without the agent-aware attention ("w/o AA attention"), we replace our agent-aware attention with standard scaled dot-product attention used in the original transformer [4]. For the variant with agent encoding ("w/ agent encoding"), in addition to removing the agent aware attention, we also append an agent encoding to each element in the trajectory sequence. The agent encoding is computed similarly as the positional encoding [4] but uses the agent index instead of the position index. For the variant without semantic maps ("w/o semantic map"), we simply do not append any visual features extracted from the semantic maps to the trajectory sequence.

Other Details. Our models are implemented using PyTorch [3] and are trained with a single NVIDIA RTX 2080 Ti and standard CPUs. The training time is approximately one day for each dataset in ETH/UCY and three days for nuScenes.

3. Additional Attention Visualization

As discussed in the main paper, our method can attend to any agent at any previous timestep when predicting the future position of an agent. Here, we provide more visualization of the attention in Fig. 2 to understand the behavior of our model. Across all the examples, it is evident that when predicting the target future position of an agent, the model pays more attention to the agent's own trajectories and recent timesteps, and it also attends more to nearby agents than distant agents.

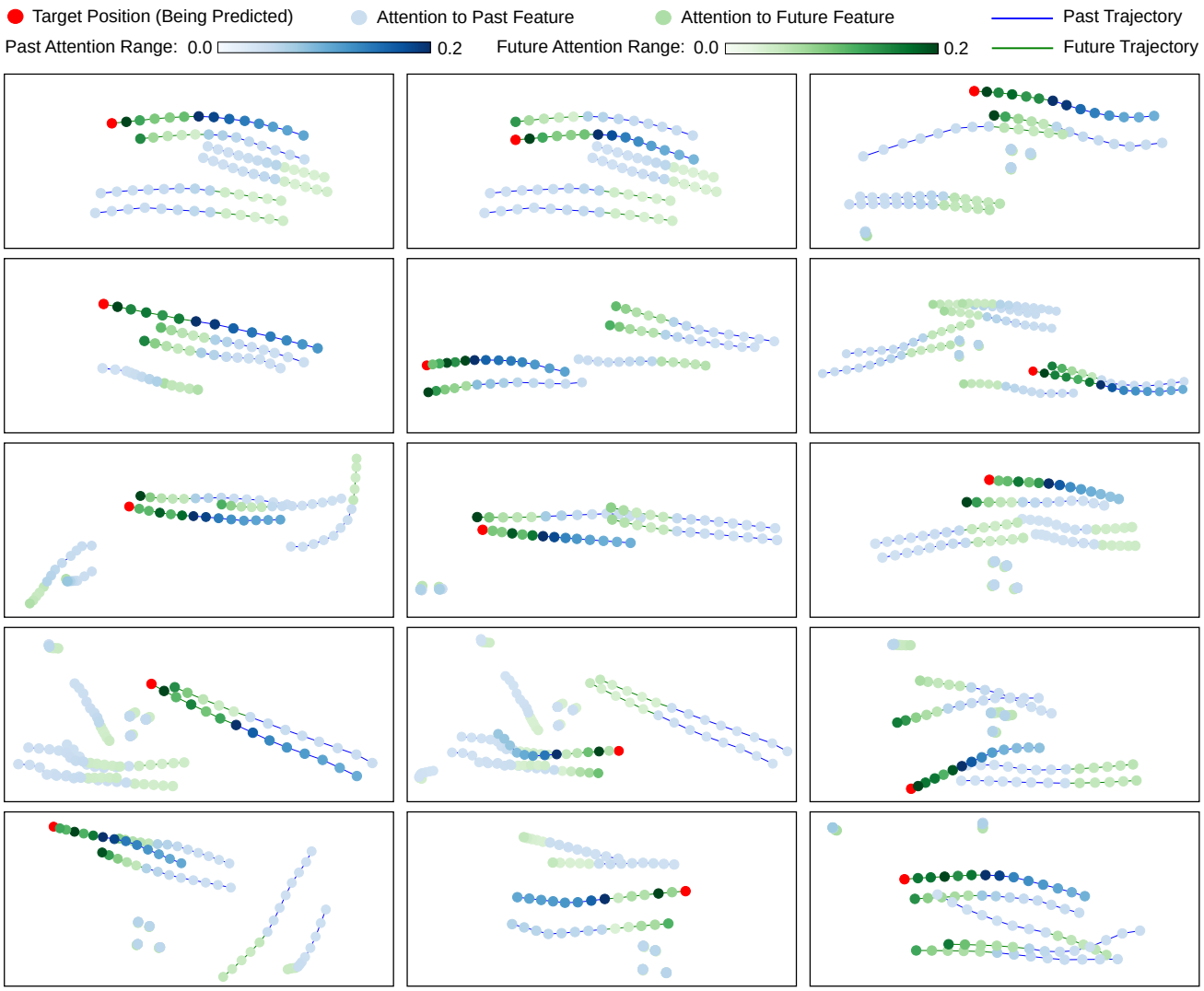

Figure 2. **Attention Visualization on the ETH/UCY dataset.** We plot the attention to past (blue) and future (green) trajectory features of all agents when inferring a target position (red). Darker color means higher attention. We can see that when predicting the target future position of an agent, the model pays more attention to the agent's own trajectories and recent timesteps, and it also attends more to nearby agents than distant agents.

4. Trajectory Sample Visualization

To demonstrate the importance of agent-aware attention, we also provide qualitative comparisons of our method against the variant without agent-aware attention (w/o AA attention) on the nuScenes dataset in Fig. 3. We can observe that the future trajectory samples produced by our method using agent-aware attention cover the ground truth (GT) future trajectories significantly better. Our method also produces much fewer implausible trajectories such as those going out of the road.

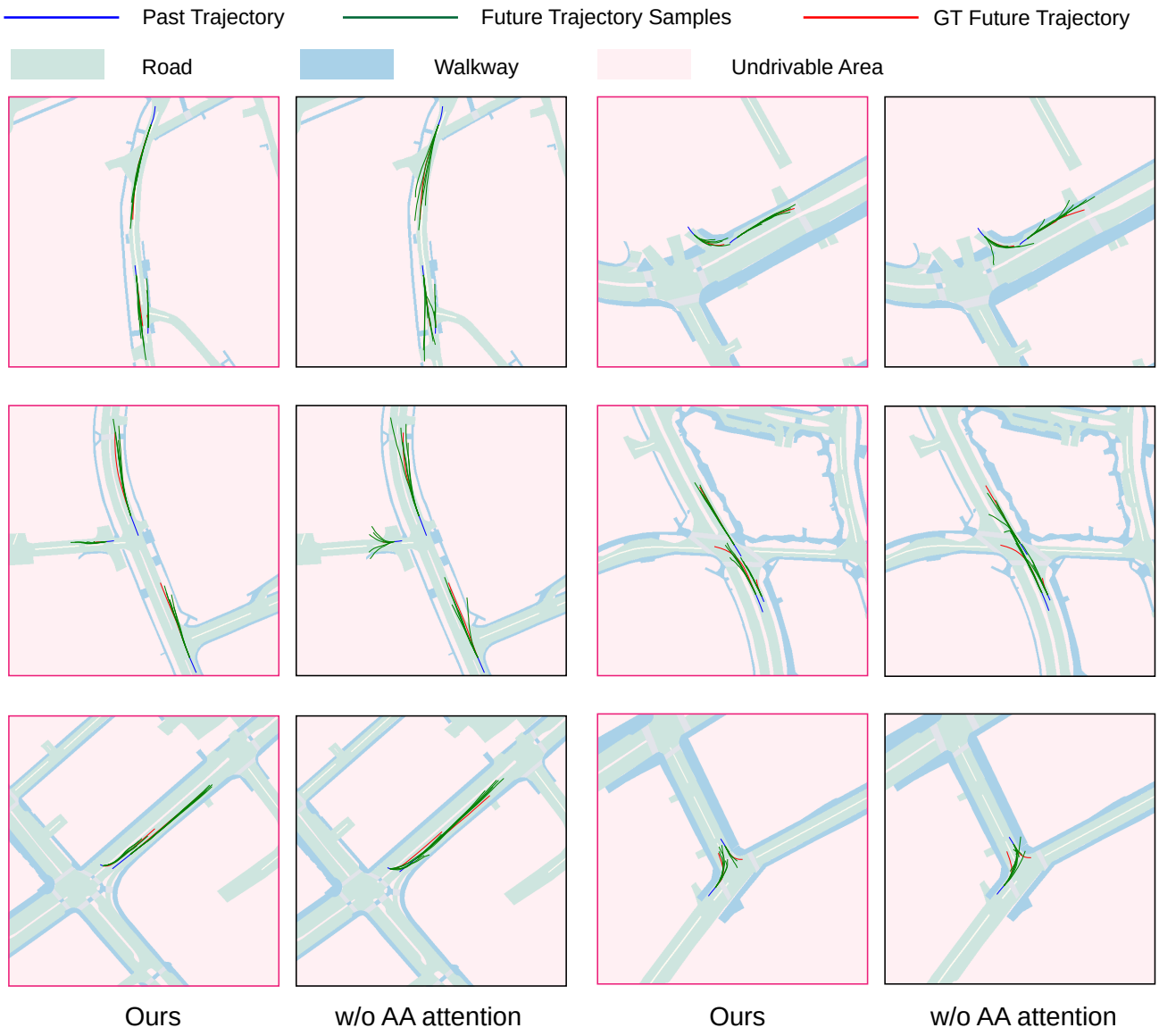

Figure 3. **Trajectory Sample Visualization on nuScenes.** We compare our method against the variant without agent-aware attention (w/o AA attention). The future trajectory samples produced by our method using agent-aware attention cover the ground truth (GT) future trajectories significantly better. Our method also produces much fewer implausible trajectories such as those going out of the road.

References

[1] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

[2] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 2

[3] 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. *arXiv preprint arXiv:1912.01703*, 2019. 2

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017. 2