Multimodal Trajectory Prediction Conditioned on Lane-Graph Traversals

Nachiket Deo [*]
University of California San Diego
`ndeo@ucsd.edu`

Eric M. Wolff
Motional
`eric.wolff@motional.com`

Oscar Beijbom
Motional
`oscar.beijbom@motional.com`

Abstract:
Accurately predicting the future motion of surrounding vehicles requires reasoning about the inherent uncertainty in goals and driving behavior. This uncertainty can be loosely decoupled into lateral (e.g., keeping lane, turning) and longitudinal (e.g., accelerating, braking). We present a novel method that combines learned discrete policy rollouts with a focused decoder on subsets of the lane graph. The policy rollouts explore different goals given our current observations, ensuring that the model captures lateral variability. The longitudinal variability is captured by our novel latent variable model decoder that is conditioned on various subsets of the lane graph. Our model achieves state-of-the-art performance on the nuScenes motion prediction dataset, and qualitatively demonstrates excellent scene compliance. Detailed ablations highlight the importance of both the policy rollouts and the decoder architecture.

Keywords: Motion prediction, autonomous vehicles, graph neural networks

1 Introduction

To safely and efficiently navigate through complex traffic scenes, autonomous vehicles need the ability to predict the intent and future trajectories of surrounding vehicles. There is inherent uncertainty in predicting the future, making trajectory prediction a challenging problem. However, there's structure to vehicle motion that can be exploited. Drivers usually tend to follow traffic rules and follow the direction ascribed to their lanes. High definition (HD) maps of driving scenes provide a succinct representation of the road topology and traffic rules, and have thus been a critical component of recent trajectory prediction models as well as public autonomous driving datasets.

Early work [1] encodes HD maps using a rasterized bird's eye view image and convolutional layers. While this approach exploits the expressive power of modern CNN architectures, rasterization of the map can be computationally inefficient, erase information due to occlusions, and require large receptive fields to aggregate context. The recently proposed VectorNet [2] and LaneGCN [3] models directly encode structured HD maps, representing lane polylines as nodes of a graph. VectorNet aggregates context using attention [4], while LaneGCN proposes a dilated variant of graph convolution [5] to aggregate context along lanes. These approaches achieve state-of-the-art performance using fewer parameters than rasterization-based approaches.

The above methods take advantage of representing the high-definition map as a graph to encode the input context. The context vector is then used by the output header to compute a multimodal prediction for the agent's future motion. Since the entire scene is aggregated together and used to compute all predictions, it places a strong requirement on the output header. In particular, the multimodal prediction header needs to account for both *lateral* or *route* variability (e.g. will the

[*]Work done during an internship at Motional.

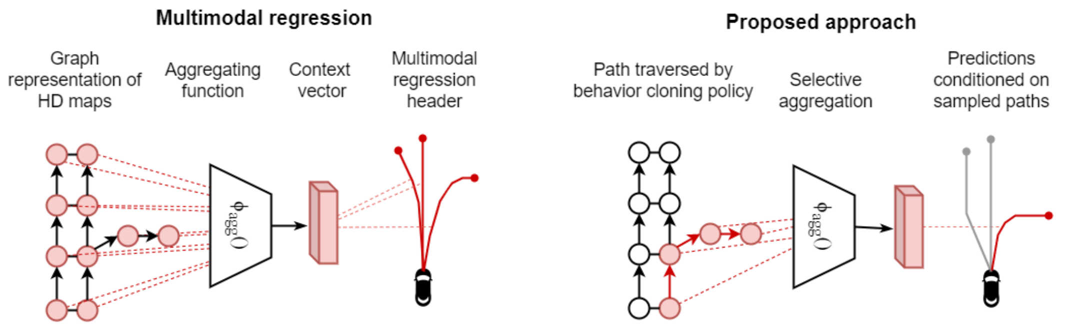
Multimodal regression
Graph representation of HD maps
Aggregating function
Context vector
Multimodal regression header
$\phi_{agg}()$
Proposed approach
Path traversed by behavior cloning policy
Selective aggregation
Predictions conditioned on sampled paths
$\phi_{agg}()$

Figure 1: **Overview of our approach.** We encode HD maps and agent tracks using a graph representation of the scene. However, instead of aggregating the entire scene context into a single vector and learning a one-to-many mapping to multiple trajectories, we condition our predictions on selectively aggregated context based on paths traversed in the graph by a discrete policy.

driver change lane, will they turn right etc.) as well as *longitudinal* variability (e.g. will the driver accelerate, brake, maintain speed) of trajectories.

Our core insight is that the graph structure of the scene can additionally be leveraged to explicitly model the lateral variability in future trajectories. We propose a novel approach for trajectory prediction which we term Prediction via Graph-based Policy (PGP). Our approach relies on two key ideas.

Predictions conditioned on traversals: We selectively aggregate part of the scene context for each prediction. We perform this selective aggregation by sampling path traversals from a learned behavior cloning policy as shown in Fig. 1. By more directly selecting the subset of the graph that is used for each prediction, we lessen the representational demands on the output decoder. Additionally, the probabilistic policy leads to a diverse set of sampled paths and captures the lateral variability of the multimodal trajectory distribution.

Latent variable for longitudinal variability: To account for longitudinal variability of trajectories, we additionally condition our predictions with a sampled latent variable. This allows our model to predict distinct trajectories even for identical path traversals. We show through our experiments that this translates to greater longitudinal variability of predictions.

We summarize our main contributions on multimodal motion prediction using high-definition maps:

- A novel combination of discrete policy rollouts with a decoder on subsets of the lane graph.
- Qualitatively capture both lateral and longitudinal variations in motion.
- State-of-the-art performance on the nuScenes motion prediction challenge.

2 Related Work

The large body of recent work in motion prediction precludes a complete review of related work, so we limit our attention to the most relevant graph-based, multimodal, and goal-conditioned models.

Graph representation of HD maps: Most self-driving cars have access to high-definition vector maps, which include detailed geometric information about objects such as lanes, crosswalks, stop signs, and more. VectorNet [2] encodes the scene context a hierarchical representation of map objects and agent trajectories. Each component is represented as a sequence of vectors, which are then processed by a local graph network. The resulting features are aggregated globally via a fully-connected graph network. LaneGCN [3] extracts a lane graph from the HD map, and uses a graph convolutional network to compute lane features. These features are combined with both agent and other lane features in a fusion network. Both methods utilize the entire graph for making predictions, relying on the header to identify the most relevant features.

Multimodal trajectory prediction: Researchers have proposed a variety of ways to model the multiple possible future trajectories that vehicles may take. One approach is to model that output as a probability distribution over trajectories, using either regression [1], ordinal regression [6], or classification [7]. Another approach models the output as a spatial-temporal occupancy grid [8].

Sampling methods use stochastic policies [9, 10] roll outs or sampled latent variables [11, 12]. These models must learn a one-to-many mapping from the entire input context (except the random variable) to multiple trajectories, and can lead to predictions that are not scene compliant.

Goal-conditioned trajectory prediction: Rather than learning a one to many mapping from the entire context to multiple future trajectories, methods such as TnT [13], LaneRCNN [14], and PEC-Net [15] condition each prediction on goals of the driver. Conditioning trajectory predictions on future goals makes intuitive sense and helps leverage the high-definition map by restricting goals to be near the lanes. However, one limitation is that over moderate time horizons, there can be multiple paths that reach a given goal location. Additionally, certain plausible goal locations might be unreachable due to constraints in the scene that are not local to the goal location, e.g., a barrier that blocks a turn lane. In contrast, our method conditions on paths traversed in a lane graph, which ensures that the inferred goal is reachable. Furthermore, the traversed path provides a stronger inductive bias than just the goal location. A similar stream of work conditions on candidate lane centerlines as goals (e.g., WIMP [16], GoalNet [17]). While the lane centerline provides more local context than just the goal, accounting for lane changes can be difficult. Additionally, routes need to be deterministically chosen, with multiple trajectories predicted along the selected route. Our approach allows for probabilistic sampling of both routes and motion profiles. In scenes with just a single plausible route, our model can use its prediction budget of K trajectories purely for different plausible motion profiles. The most closely related work is P2T [18]. They predict trajectories conditioned on paths explored by an IRL policy over a grid defined over the scene. However, they use a rasterized BEV image for the scene, which leads to inefficient encoders and loss of connectivity information due to occlusions. Additionally, their model cannot generate different motion profiles along a sampled path.

3 Formulation

We predict the future trajectories of vehicles of interest, conditioned on their past trajectory, the past trajectories of nearby vehicles and pedestrians, and the static scene around them represented by an HD map. We represent the scene and predict trajectories in the bird's eye view and use an agent-centric frame of reference aligned along the agent's instantaneous direction of motion.

3.1 Trajectory representation

We assume access to past trajectories of agents in the scene obtained from on-board detectors and multi-object trackers. We represent the past trajectory of agent i as a sequence of motion state vectors $s^i_{-t_h:0} = [s^i_{-t_h}, ..., s^i_{-1}, s^i_0]$ over the past t_h time steps. Each $s^i_t = [x^i_t, y^i_t, v^i_t, a^i_t, \omega^i_t, \mathcal{I}^i]$, where x^i_t, y^i_t are the bird's eye view location co-ordinates, v^i_t, a^i_t and ω^i_t respectively are the speed, acceleration and yaw-rate of the agent at time t and $\mathcal{I}^i$ is a binary indicator with value 1 if the agent is a pedestrian and 0 if the agent is a vehicle. We nominally assign the index 0 to the target vehicle being predicted, and timestamp 0 to the time of prediction.

3.2 Representing HD maps as lane graphs

Similar to [3], we represent the HD map as a directed graph $\mathcal{G}(V, E)$, with nodes V denoting lane centerlines and edges E denoting lane connectivity.

Nodes: We consider all lane centerlines within an area of [-50, 50] m laterally and [-20, 80] m longitudinally around the vehicle of interest. We divide the lane centerlines into snippets of length $\leq$ 20m, and discretize them to a set of N poses with 1m resolution. Each snippet corresponds to a node in our graph, with a node v represented by a sequence of feature vectors $f^v_{1:N} = [f^v_1, ..., f^v_N]$. Here each $f^v_n = [x^v_n, y^v_n, \theta^v_n, \mathcal{I}^v_n]$, where x^v_n, y^v_n and θ^v_n are the location and yaw of the n^{th} pose of v and $\mathcal{I}^v_n$ is a 2-D binary vector indicating whether the pose lies on a stop line or crosswalk. Thus, our node features capture both the geometry as well as traffic control elements along lane centerlines.

Edges: We constrain edges in the lane graph such that any traversed path through the graph corresponds to a legal route that a vehicle can take in the scene. We consider two types of edges. Successor edges (E_{suc}) connect nodes to the next node along a lane. A given node can have multiple successors if a lane branches out at an intersection. Similarly, multiple nodes can have the same successor if two or more lanes merge. To account for lane changes, we additionally define proximal edges (E_{prox}) between neighboring lane nodes if they are within a distance threshold of each other and their directions of motion are within a yaw threshold.

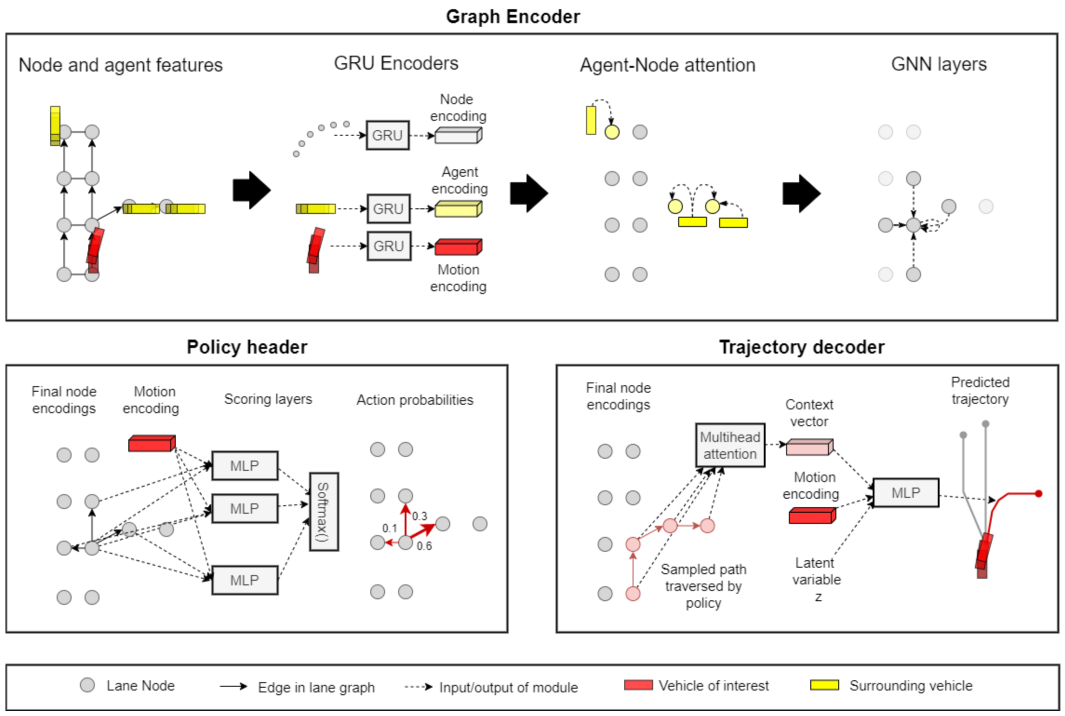

Figure 2: **Proposed model.** PGP consists of three modules trained end-to-end. The graph encoder (top) encodes agent and map context as node encodings of a directed lane-graph. The policy header (bottom-left) learns a discrete policy for sampled graph traversals. The trajectory decoder (bottom-right) predicts trajectories by selectively attending to node encodings along paths traversed by the policy and a sampled latent variable.

3.3 Output representation

To account for multimodality of the distribution of future trajectories, we output a set of K trajectories $[\tau^1_{1:t_f}, \tau^2_{1:t_f}, ..., \tau^K_{1:t_f}]$ for the vehicle of interest consisting of future x-y locations over a prediction horizon of t_f time steps. Each of the K trajectories represents a mode of the predictive distribution, ideally corresponding to different plausible routes or different motion profiles along the same route.

4 Proposed Model

Figure 2 provides an overview of our proposed model. It consists of three interacting modules that are trained end-to-end. The *graph encoder* forms the backbone of our model. It outputs learned representations for each node of the lane graph by incorporating the HD map as well as agent context around the nodes. The *policy header* outputs a discrete probability distribution over outgoing edges at each node, allowing us to probabilistically sample paths in the graph. Finally, our attention based *trajectory decoder* outputs trajectories conditioned on paths traversed by the policy and a sample from a simple latent distribution. Sections 4.1, 4.2 and 4.3 describe the graph encoder, policy header and trajectory decoder in greater detail.

4.1 Encoding scene and agent context

Inspired by the simplicity and effectiveness of graph based encoders for trajectory prediction [2, 3], we seek to encode all agent features and map features as node encodings of our lane graph $\mathcal{G}(V, E)$.

GRU encoders. Both, agent trajectories and lane polylines form sequences of features with a well defined order. We first independently encode both sets of features using gated recurrent unit (GRU) encoders. We use three GRU encoders for encoding the target vehicle trajectory $s^0_{-t_h:0}$, surrounding vehicle trajectories $s^i_{-t_h:0}$ and node features $f^v_{1:N}$. These output the motion encoding h_{motion}, agent encodings h^i_{agent} and initial node encodings h^v_{node} respectively.

Agent-node attention. We wish to incorporate both agent and map features into node encodings to be used by our policy header and trajectory decoder. Similar to LaneGCN [3], we update node encodings with nearby agent encodings using scaled dot product attention [4]. For each node v, we only consider nearby agents within 10m of the lane centerline. We obtain keys and values by linearly projecting encodings h^i_{agent} of nearby agents, and the query by linearly projecting h^v_{node}. Finally, the updated node encoding is obtained by concatenating the output of the attention layer with the original node encoding.

GNN layers. With the node encodings updated with nearby agent features, we aggregate local context at each node by using graph neural network (GNN) layers. We experiment with graph convolution (GCN) [5] and graph attention (GAT) [19] layers here. For the GNN layers, we treat both successor and proximal edges as equivalent and bidirectional. This allows us to aggregate context along all directions around each node. The outputs of the GNN layers serve as the final node encodings learned by the graph encoder.

4.2 Discrete policy for graph traversal

We seek to learn a probabilistic policy π_{route} for graph traversal such that sampled roll-outs of the policy correspond to likely routes that the target vehicle would take in the future. We represent our policy as a discrete probability distribution over outgoing edges at each node in our graph. We additionally also include edges from every node to an *end* state to allow π_{route} to terminate at a goal location. The edge probabilities are output by the policy header shown in figure 2. The policy header uses an MLP with shared weights to output a scalar score for each edge in the graph. For a directed edge $(u, v) \in E$, the score is given by,

$$\text{score}(u, v) = \text{MLP}\left(\text{concat}(h_{motion}, h^u_{node}, h^v_{node}, \mathbb{1}_{(u,v)\in E_{suc}})\right), \tag{1}$$

The scoring function thus takes into account the motion of the target vehicle as well as local context at the specific edge. We then normalize the scores using a softmax layer for all outgoing edges at each node to output the policy for graph traversal,

$$\pi_{route}(u) = \text{softmax}(\{\text{score}(u, v) | (u, v) \in E\}), \tag{2}$$

We train the policy header using behavior cloning. For each prediction instance, we use the ground truth future trajectory to determine which nodes were visited by the vehicle. We assign each pose in the future trajectory of the vehicle to the closest node in the graph with direction of motion within a yaw threshold of the pose. An edge is treated as visited if both its source and destination nodes are visited. We use negative log likelihood of the edge probabilities for all edges visited by the ground truth trajectory as the loss function for training the graph traversal policy, denoted by $\mathcal{L}_{BC}$.

4.3 Decoding trajectories conditioned on traversals

Sampling roll-outs of π_{route} yields plausible future routes for the target vehicle. We posit that the most relevant context for predicting future trajectories is along these routes and propose a trajectory decoder that selectively aggregates context along the sampled routes.

Given a sequence of nodes $[v_1, v_2, ..., v_M]$ corresponding to a sampled policy roll-out, our trajectory decoder uses multi-head scaled dot product attention [4] to aggregate map and agent context over the node sequence as shown in figure 2. We linearly project the target vehicle's motion encoding to obtain the query, while we linearly project the node features $[h^{v_1}_{node}, h^{v_2}_{node}, ..., h^{v_M}_{node}]$ to obtain keys and values for computing attention. The multi-head attention layer outputs a context vector $\mathcal{C}$ encoding the route. Each distinct policy roll-out yields a distinct context vector, allowing us to predict trajectories along a diverse set of routes.

Diversity in routes alone does not account for the multimodality of the distribution of future trajectories. Drivers can accelerate, brake and follow different motion profiles along a planned route. To allow the model to output distinct motion profiles along the same route, we additionally condition our predictions with a latent vector z sampled from the multivariate standard normal distribution.

Finally, to sample a trajectory $\tau^k_{1:t_f}$ from our model, we sample a roll-out of π_{route} and obtain $\mathcal{C}_k$, we sample z_k from the latent distribution and concatenate both with h_{motion} and pass them through an MLP to output $\tau^k_{1:t_f}$ the future locations over t_f timesteps,

$$\tau^k_{1:t_f} = \text{MLP}(\text{concat}(h_{motion}, \mathcal{C}_k, z_k)). \tag{3}$$

The sampling process can often be redundant, yielding similar or repeated trajectories. However our light-weight encoder and decoder heads allows us to sample a large number of trajectories in parallel. To obtain a final set of K modes of the trajectory distribution, we use K-means clustering and output the cluster centers as our final set of K predictions $[\tau^1_{1:t_f}, \tau^2_{1:t_f}, ..., \tau^K_{1:t_f}]$. Similar to prior work [1, 6, 12], we train our trajectory decoder using the winner takes all variant of average displacement error in order to not penalize the diverse plausible trajectories output by our model,

$$\mathcal{L}_{reg} = \min_k \frac{1}{t_f} \sum_{t=1}^{t_f} \|\tau^k_t - \tau^{gt}_t\|_2, \tag{4}$$

where τ^{gt} is the true future trajectory.

4.4 Training

We train the model end-to-end using a multi-task loss given by,

$$\mathcal{L} = \mathcal{L}_{BC} + \mathcal{L}_{reg}, \tag{5}$$

where $\mathcal{L}_{BC}$ is the negative log likelihood loss for training π_{route} and $\mathcal{L}_{reg}$ is the WTA regression loss described above. For the first few epochs of training, since π_{route} does not produce meaningful traversals, we use the ground truth traversal for sampling trajectories and computing $\mathcal{L}_{reg}$.

5 Experiments

5.1 Dataset

We evaluate our method on nuScenes [20], a self-driving car dataset collected in Boston and Singapore. nuScenes contains 1000 scenes, each 20 seconds, with ground truth annotations and high-definition maps. Vehicles have manually-annotated 3D bounding boxes, which are published at 2 Hz. The prediction task is to use the past 2 seconds of object history and the map to predict the next 6 seconds. We use the standard split from the nuScenes software kit [21].

5.2 Metrics

To evaluate our model, we use the standard metrics on the nuScenes leaderboard [21]. The minimum average displacement error (ADE) over the top K predictions (MinADE$_K$). The miss rate (MissRate$_{K,2}$) only penalizes predictions that are further than 2 m from the ground truth. The offroad rate measures the fraction of predictions that are off the road. Since all examples in nuScenes are on the road, this should be zero.

5.3 Comparison to the state of the art

We report our results on the standard benchmark split of the nuScenes prediction dataset in table 1, comparing with the top performing entries on the nuScenes leaderboard. We achieve state of the art results on almost all metrics, significantly outperforming the previous best entry P2T [18] on the MinADE$_K$ and MissRate metrics, while achieving comparable off-road rate. This suggests that our model achieves better coverage of the modes of the trajectory distribution, while still predicting trajectories that are scene-compliant.

5.4 Encoder ablations

We analyze the effects of our graph structure and components of the graph encoder by performing ablations on the graph encoder reported in table 2. In particular we analyze the effect of includ-

Table 1: Comparison with the state of the art on nuScenes

Model	MinADE$_5$	MinADE$_{10}$	MissRate$_{5,2}$	MissRate$_{10,2}$	Offroad rate
CoverNet [7]	1.96	1.48	0.67	-	-
Trajectron++ [22]	1.88	1.51	0.70	0.57	0.25
SG-Net [23]	1.86	1.40	0.67	0.52	0.04
MHA-JAM [24]	1.81	1.24	**0.59**	0.46	0.07
CXX [25]	1.63	1.29	0.69	0.60	0.08
P2T [18]	1.45	1.16	0.64	0.46	**0.03**
PGP (Ours)	**1.30**	**1.00**	0.61	**0.37**	**0.03**

Table 2: Encoder ablations

Graph structure		Agent-node attention	GNN layers	MinADE$_K$		MissRate$_{K,2}$		Offroad rate
E_{suc}	E_{prox}			K=5	K=10	K=5	K=10	
✓				1.35	1.03	0.64	0.41	0.04
✓	✓			1.33	1.01	0.63	0.38	0.03
✓	✓	✓		**1.30**	**1.00**	**0.61**	**0.37**	**0.03**
✓	✓	✓	GCN $\times$ 1	1.31	1.01	0.62	0.39	0.04
✓	✓	✓	GCN $\times$ 2	1.31	1.01	0.61	0.39	0.04
✓	✓	✓	GAT $\times$ 1	**1.30**	**1.00**	0.62	0.38	0.03
✓	✓	✓	GAT $\times$ 2	1.31	1.01	**0.61**	**0.37**	**0.03**

ing proximal edges, modeling surrounding agent context through agent-node attention and finally updating node encodings with local context using GCN [5] or GAT [19] layers.

We get improvement across all metrics by adding proximal edges, and agent-node attention, suggesting the importance of modeling lane changes and agent context. Somewhat surprisingly, adding GNN layers gives ambiguous results with GCN layers achieving slightly worse results than an encoder without any GNN layers and GAT layers performing on par with the encoder without GNN layers. This could be because the multi-head attention layer aggregates context across the entire traversed path, making the GNN based aggregation redundant.

5.5 Decoder ablations

We next analyze the effect of our traversal and latent variable based decoder. We compare several decoders, all built on top of our proposed encoder with both types of edges, agent-node attention and 2 GAT layers. First, we consider the multimodal regression header from MTP [1]. Next we consider ablations of our decoder without the graph traversals and without the latent variable conditioning. Finally, we consider a model that conditions predictions on sampled goals at different node locations, rather than sampled traversals. Table 3 reports quantitative results while figure 3 shows qualitative results comparing the different decoders. We make the following observations.

MTP generally fares worse compared to the other decoders, particularly in terms of offroad rate. We note from fig. 3 that while it generates a diverse set of trajectories, it tends to predict several off-road trajectories.

The decoders conditioned purely on the latent variable or purely on traversals both fare worse compared to our decoder conditioned on both. Qualitatively we observe that this is for different reasons. The latent variable only decoder almost always predicts trajectories along a single route lacking lateral variability, but generates trajectories with longitudinal variability. On the other hand the traversal only decoder predicts trajectories over a variety of routes, but lacks longitudinal variability.

Finally, the goal + latent variable conditioned decoder also fares worse compared to our traversal + latent variable conditioned decoder, again, especially in terms of off-road rate. From the qualitative examples we observe that this is due to two types of errors. First, the model tend to predict spurious goals which aren't reachable for the target vehicle (rows 3 and 4), and second, while it predicts the correct goals it generates trajectories that don't follow accurate paths to those goals (rows 2 and 6).

Table 3: Decoder ablations

Decoder	MinADE$_5$	MinADE$_{10}$	MissRate$_{5,2}$	MissRate$_{10,2}$	Offroad rate
MTP [1]	1.59	1.12	**0.57**	0.48	0.08
Latent var (LV) only	1.38	1.08	0.65	0.43	0.05
Traversal only	1.37	1.10	0.65	0.44	0.04
Goals + LV	1.33	1.02	0.60	0.42	0.06
Traversals + LV	**1.31**	**1.01**	0.61	**0.37**	**0.03**

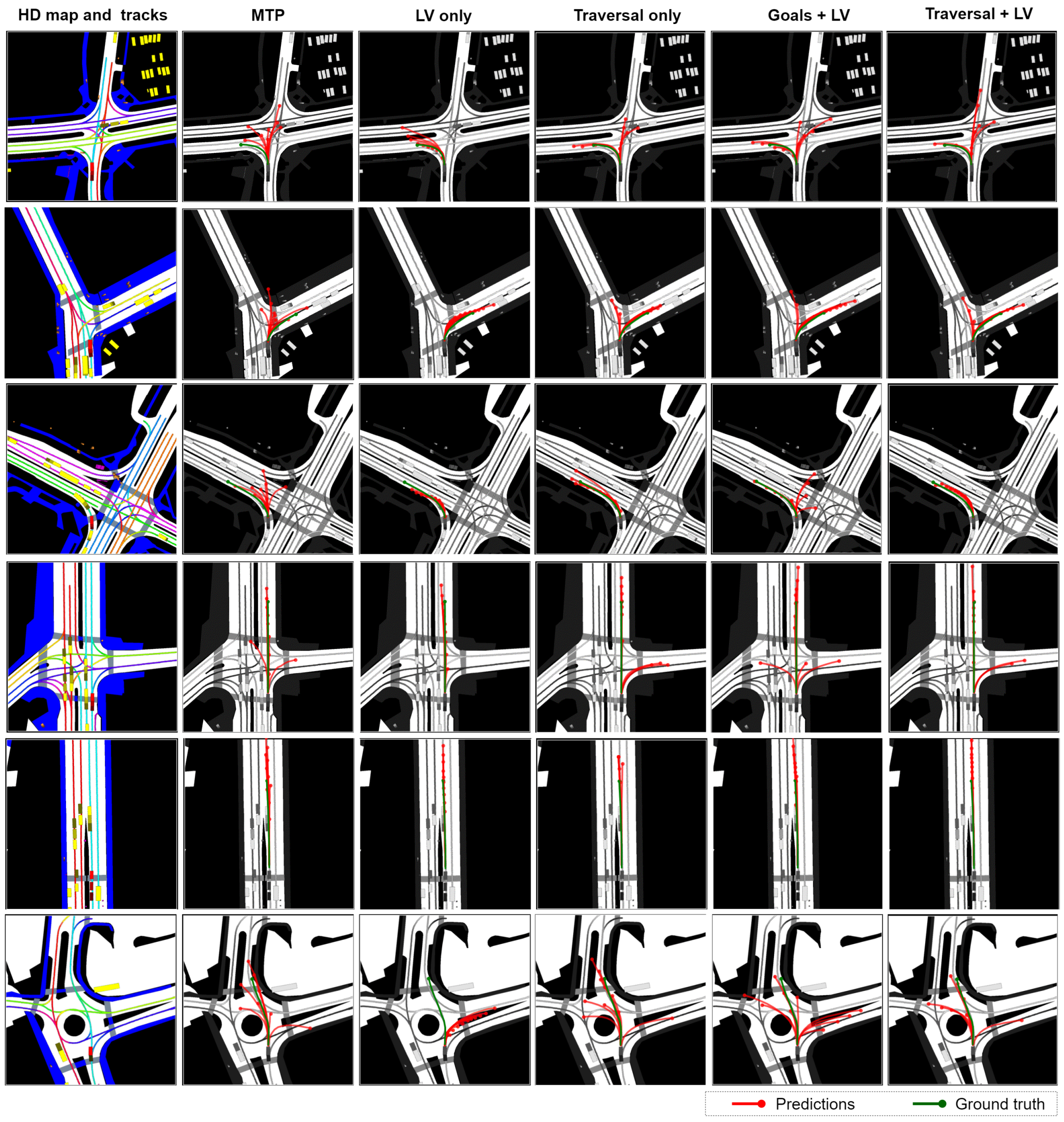

Figure 3: Qualitative comparison of decoders

6 Conclusions

We presented a novel method for multimodal trajectory prediction conditioned on lane-graph traversals. A learned discrete policy creates lane-graph traversals, which let the decoder attend to a subset of the lane graph. These traversals explore different possible goals, ensuring that the model captures lateral variability in the agent's future motion. Longitudinal variability is captured by a latent variable model in the decoder. Our model achieves state-of-the-art performance on the nuScenes dataset, and qualitatively demonstrates excellent scene compliance.

References

[1] H. Cui, V. Radosavljevic, F. Chou, T. Lin, T. Nguyen, T. Huang, J. Schneider, and N. Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In *International Conference on Robotics and Automation (ICRA)*, May 2019.

[2] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid. VectorNet: Encoding HD maps and agent dynamics from vectorized representation, 2020. https://arxiv.org/abs/2005.04259.

[3] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun. Learning lane graph representations for motion forecasting. *CoRR*, abs/2007.13732, 2020. URL https://arxiv.org/abs/2007.13732.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[5] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[6] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov. MultiPath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. In *3rd Conference on Robot Learning (CoRL)*, November 2019.

[7] T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom, and E. M. Wolff. CoverNet: Multimodal behavior prediction using trajectory sets, 2019. https://arxiv.org/abs/1911.10298.

[8] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun. End-to-end interpretable neural motion planner. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[9] N. Rhinehart, K. M. Kitani, and P. Vernaza. R2P2: A reparameterized pushforward policy for diverse, precise generative path forecasting. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[10] N. Rhinehart, R. McAllister, K. Kitani, and S. Levine. PRECOG: Prediction conditioned on goals in visual multi-agent settings. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[11] N. Lee, W. Choi, P. Vernaza, C. Choy, P. Torr, and M. Chandraker. DESIRE: Distant future prediction in dynamic scenes with interacting agents. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

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

[13] H. Zhao, J. Gao, T. Lan, C. Sun, B. Sapp, B. Varadarajan, Y. Shen, Y. Shen, Y. Chai, C. Schmid, C. Li, and D. Anguelov. Tnt: Target-driven trajectory prediction, 2020.

[14] W. Zeng, M. Liang, R. Liao, and R. Urtasun. Lanercnn: Distributed representations for graph-centric motion forecasting, 2021.

[15] K. Mangalam, H. Girase, S. Agarwal, K.-H. Lee, E. Adeli, J. Malik, and A. Gaidon. It is not the journey but the destination: Endpoint conditioned trajectory prediction. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *Computer Vision – ECCV 2020*, pages 759–776, Cham, 2020. ISBN 978-3-030-58536-5.

[16] S. Khandelwal, W. Qi, J. Singh, A. Hartnett, and D. Ramanan. What-if motion prediction for autonomous driving, 2020.

[17] Z. Cao, H. Gao, K. Mangalam, Q.-Z. Cai, M. Vo, and J. Malik. Long-term human motion prediction with scene context. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *Computer Vision – ECCV 2020*, pages 387–404, 2020. ISBN 978-3-030-58452-8.

[18] N. Deo and M. M. Trivedi. Trajectory forecasts in unknown environments conditioned on grid-based plans, 2021.

[19] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[20] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuScenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[21] nuScenes Contributors. nuScenes. `https://www.nuscenes.org/`, 2020.

[22] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone. Trajectron++: Multi-agent generative trajectory forecasting with heterogeneous data for control. *CoRR*, abs/2001.03093, 2020. URL `http://arxiv.org/abs/2001.03093`.

[23] C. Wang, Y. Wang, M. Xu, and D. J. Crandall. Stepwise goal-driven networks for trajectory prediction. *arXiv preprint arXiv:2103.14107*, 2021.

[24] K. Messaoud, N. Deo, M. M. Trivedi, and F. Nashashibi. Multi-head attention with joint agent-map representation for trajectory prediction in autonomous driving. *CoRR*, abs/2005.02545, 2020. URL `https://arxiv.org/abs/2005.02545`.

[25] C. Luo, L. Sun, D. Dabiri, and A. Yuille. Probabilistic multi-modal trajectory prediction with lane attention for autonomous vehicles. *arXiv preprint arXiv:2007.02574*, 2020.

A Implementation details

We implement our model using Pytorch[2]. Here, we provide details of our model architecture, ablations and training.

A.1 GRU encoders

We embed both agent and node features using linear layers of size 16, followed by a leaky ReLU non-linearity. We use GRUs with depth 1 and hidden state dimension 32 on top of the embeddings for both the agent and node encoders.

A.2 Agent-node attention

We use scaled dot-product attention with a single attention head for the agent-node attention layers. We use 32×32 weight matrices for projecting the node and agent encodings for obtaining the queries, and keys and values respectively. The outputs of the attention layer are concatenated with the original node encodings and passed through a linear layer of size 32, followed by a leaky ReLU non-linearity to obtain updated node encodings of the same size as the original node encodings.

A.3 GNN layers

We use Pytorch geometric[3] for implementing the GCN and GAT layers of our model. For GCN layers, we use the layer-wise propagation rule from [5]. Our adjacency matrix includes both successor and proximal edges (treated as bidirectional), as well as self loops. The outputs at each node have the same dimension, 32, as the inputs. For GAT layers, we use the layer-wise propagation rule from [19]. We use a single attention head, with the outputs again having the same dimension as the inputs.

A.4 Policy header

The policy header is implemented as an MLP with 2 hidden layers of size 32 each and a scalar output. The input to the policy header for each edge is a vector of size 98, consisting of the source node encoding, destination node encoding and motion encoding of the target agent each of size 32, and a one-hot encoding for the edge type of size 2.

A.5 Trajectory decoder

We aggregate context along nodes traversed by the policy using a multi-head scaled dot-product attention layer. The attention layer has 32 parallel attention heads, and outputs a context vector $\mathcal{C}$ of size 128. We model the latent variable as a multivariate standard normal distribution. $z \sim \mathcal{N}(0, \mathbf{I})$, where $\mathbf{I}$ is a 5×5 identity matrix. We output a trajectory for each sampled $\mathcal{C}_k$, z_k and h_{motion} using an MLP with a hidden layer of size 128, and output of size 24 (x and y co-ordinates over the prediction horizon of 6 seconds at 2 Hz). We sample 200 trajectories from the model and cluster to obtain K=10 trajectories during training to compute the winner takes all regression loss $\mathcal{L}_{reg}$.

A.6 Training

We train the model using Adam, with learning rate 1e-4, and a batch size of 32. We pre-train the model using the ground truth path traversed through the graph for 100 epochs. We then finetune using paths sampled from π_{route} for 100 epochs. We train our model using an AWS "p3-8xlarge" instance with 4 NVIDIA Tesla V100 GPUs. Each pre-training epoch takes roughly 1 minute and each finetuning epoch takes roughly 5 minutes for nuScenes.

A.7 Decoder ablation details

MTP: For the MTP header, we first aggregate context over the entire graph using a multi-head scaled dot-product attention layer identical to our trajectory decoder, with 32 parallel attention heads and

[2]https://pytorch.org/
[3]https://github.com/rusty1s/pytorch_geometric

an output context vector $\mathcal{C}$ of size 128. We then use two fully connected layers of size 240 and 10 respectively to output K=10 trajectories, and K probabilities.

LV only: For the LV only decoder, similar to the MTP header, we first aggregate context over the entire graph using a multi-head attention layer with 32 attention heads and output $\mathcal{C}$ of size 128. The decoder then outputs trajectories conditioned on $\mathcal{C}$, h_{motion} and a sample z_k of the latent variable using the final MLP layer.

Traversal only: The traversal only decoder is identical to the trajectory decoder of our complete model, except for the final MLP layer, which outputs trajectories conditioned only on $\mathcal{C}_k$ and h_{motion} and not on the sampled latent variable z_k.

Goals + LV: The Goals + LV decoder consists of two output headers: A goal prediction header that outputs a scalar score at each node normalized using a softmax layer to give goal probabilities, and a trajectory decoder that outputs goal conditioned trajectories. We model the goal prediction header using an MLP with 2 hidden layers, each of size 32, and a scalar output. The input to the goal prediction header at each node is obtained by concatenating h_{node} and h_{motion}. The trajectory decoder consists of a multi-head attention layer with 32 heads that aggregates context over the entire graph to output a context vector $\mathcal{C}$ of size 128. $\mathcal{C}$ is concatenated with h_{motion}, a sampled latent vector z_k and the node encoding of a sampled goal $h_{node}^{u_k}$ and passed through an MLP with a hidden layer of size 128, and output size 24 corresponding to a goal conditioned trajectory.