Learning and Aggregating Lane Graphs for Urban Automated Driving

- Supplementary Material -

Martin Büchner*, Jannik Zürn*, Ion-George Todoran, Abhinav Valada, and Wolfram Burgard

In our supplementary material, we expand upon multiple aspects of our paper. In Sec. S.1, we visualize exemplary data from our compiled *UrbanLaneGraph* dataset and detail pre- and post-processing methods. We also discuss the proposed benchmark for evaluating lane graph prediction models.

In Sec. S.2 we give additional detail on evaluation metrics. In Sec. S.3, we discuss the sampling of the annotated graph into a representation suitable to train our LaneGNN model. In Sec. S.4, we provide additional explanations on the model architectures, the training procedures, and hyperparameter selection. In Sec. S.5, we explain our graph aggregation in more detail compared to the main manuscript. Finally, in Sec.. S.6, S.7, and S.8, we provide additional ablation studies and evaluations for our Successor-LGP, Full-LGP, and Planning tasks, respectively.

S.1. UrbanLaneGraph Dataset Details

As detailed in the main manuscript, we introduced a large-scale lane graph dataset including aligned high-resolution aerial images. In the following, we give a thorough description of the dataset curation process and how we designed the lane graph prediction benchmark.

S.1.1. Dataset Curation

S.1.1.1 Annotation Pre-Processing

Our graph annotation source is the large-scale Argoverse2 [29] autonomous driving dataset, which includes HD map annotations, including lane graphs, for all scenarios entailed in the dataset. In the context of the dataset, a scenario is a small-scale region ($50\,\mathrm{m}$ diameter). Our goal is to estimate arbitrarily large lane graphs, rendering the per-scenario graph annotation scheme insufficient. We aggregated all scenario graph annotations into a per-city global graph. However, we found that many scenarios overlap while the nodes in the respective overlaps do not have a perfect positional match. Therefore, we implemented an annotation merging procedure, producing the desired globally consistent ground-truth graph.

S.1.1.2 Image-Graph Alignment

The coordinate system of the annotations is not consistent with our aerial image coordinate frame. We therefore, transform the graph annotations into the image coordinate system employing the Kabsch-Umeyama [23] algorithm, in which a set of selected point pairs in the source and in the target frame are aligned, minimizing a least-squares objective function. The solution to the minimization problem comprises the optimal translation, rotation, and scaling that maps points from one frame into the other.

S.1.1.3 Regional Train-Test Splits

We split the dataset into disjoint train and test splits on a geospatial basis. Concretely, for the experiments carried out in this work, we select a challenging subset of the annotated regions within each city as a separate test split. The remaining regions are leveraged for model training. For all tasks and models, we consistently use the same training and testing regions.

S.1.1.4 Graph Sampling into Crops

In order to generate samples for training our LaneGNN model, we sample the dataset into crops. As illustrated in our manuscript, for our successor lane graph prediction task, the input to our model is a $256\,\mathrm{px} \times 256\,\mathrm{px}$ crop of the original birds-eye view image. A sample consists of the aerial image crop and the lane successor graph that starts at the bottom center position of the crop. Crop positions are selected according to the position of nodes in the lane graph annotations. We also crop the annotated graph in order to use only graph nodes and edges as learning targets that are visible in the respective crop.

As described in the main manuscript, for aggregating the locally predicted successor graphs into a global graph, we facilitate an iterative aggregation scheme where the position and orientation of the next crop are determined according to the graph prediction in the current crop. This procedure can be interpreted as imitation learning from expert data, which are the crops present in the training data. This can produce unstable trajectories that do not follow lanes when deploying the model if the training data distribution does not cover the full distribution of crops from arbitrary positions and orientations. We, therefore, add Gaussian noise w.r.t the crop position (x_{crop}, y_{crop}) and orientation γ_{crop} when sampling training data crops. Concretely, we sample $x_{crop} \sim \mathcal{N}(x_{gt},\, 5)$, $y_{crop} \sim \mathcal{N}(y_{gt},\, 5)$, and $\gamma_{crop} \sim \mathcal{N}(\gamma_{gt},\, 0.3)$. The units are px and rad, respectively. This crop sampling scheme allows the model

Figure S.1. Exemplary visualization of aerial imagery with a pixel-aligned lane graph for multiple US cities. Our *UrbanLaneGraph* dataset features a wide range of environments, including rural, suburban, and urban areas. The graph annotations feature a wide range of topological complexity scales from straight road sections to large-scale intersection scenarios with multiple incoming and outgoing lanes. The aerial images provided with the dataset have challenging visual properties such as prominent shadows, and occlusions of streets due to trees and other vegetation.

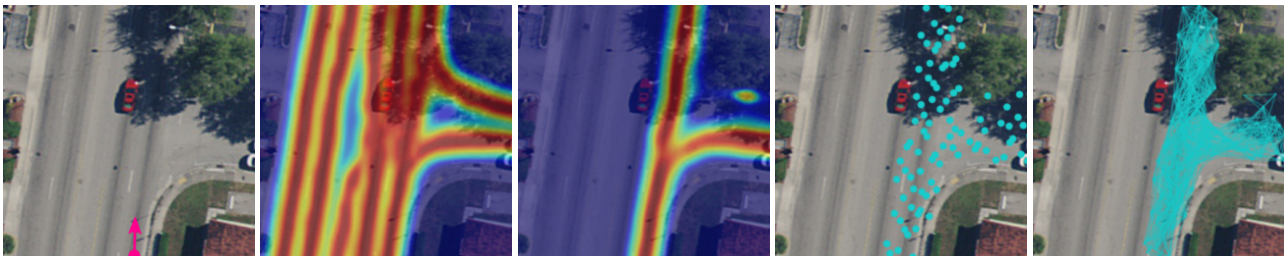
to recover from crop positions and orientations that are not well-aligned with the ground-truth graph.

S.1.1.5 Centerline Regression Data

The centerline regression targets are obtained by rendering an inverse signed distance function from the graph as an image with the same domain as the aerial image crop. Outputs of models trained on this are visualized in Fig. S.4 (lane centerline regression) and in Fig. S.3 (ego lane centerline regression).

S.1.2. Benchmark

We envision the previously described dataset as a reference for evaluating future approaches. To facilitate easier and quantitatively fair comparison between different approaches, we provide an easy-to-use graph prediction benchmarking API. For calculating all our metrics and generating visualizations, we leverage the networkx [13] Python library.



(a) (b) (c) (d) (e)

Figure S.2. Visualization of the data modalities that serve as the input to our LaneGNN model. From left to right: a) An aerial BEV image of a local scene, including the virtual agent pose for this crop. b) The regression output $\mathbb{S}_{lane}$ of our lane regression model. c) The regression output $\mathbb{S}_{lane}^{ego}$ of the agent-centric ego lane regressor. d) The lane graph node proposal that potentially holds the successor graph. e) The edge proposal list densely connects neighboring proposal nodes.

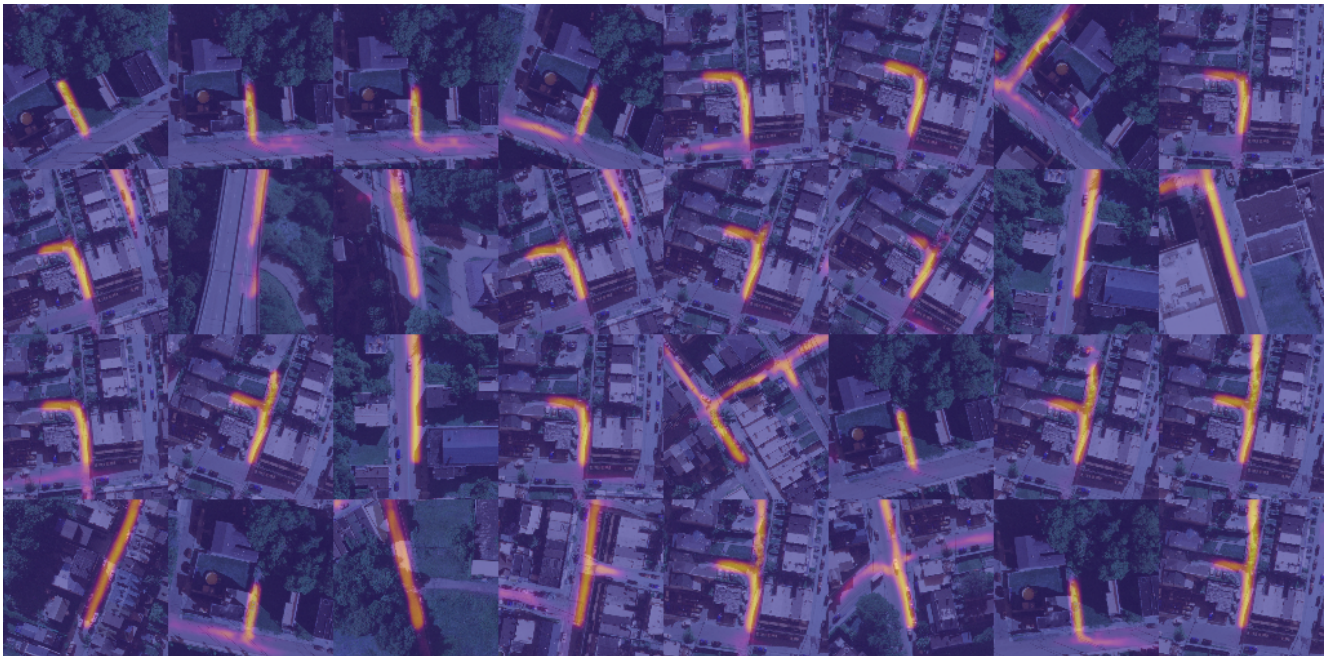


Figure S.3. Exemplary visualizations of our ego lane regression model on random crops from the test split of our dataset.

The overall information flow of our benchmarking system is visualized in Fig. S.5. Given an aerial image, the to-be-evaluated model predicts a graph `g_pred`. The `LaneGraphEvaluator` expects `g_pred` to be a networkx-type object; the conversion of the model output into networkx format may be implemented by the authors in a `to_networkx()` function. Given a predicted graph in networkx format, our evaluator queries the ground-truth graph annotations and crops the annotations to cover the same region as the predicted graph. Subsequently, all described metrics are evaluated and exported. Optionally, visualizations to rasterized or vector image formats may be produced. Finally, path-planning experiments on the predicted and ground-truth graphs may be conducted, evaluated, and visualized.

S.2. Evaluation Metrics Details

In the following, we detail the GEO and TOPO metrics, proposed by He *et al*. [15] which were used, among others metrics, to evaluate or experiments.

S.2.1. GEO Metric

The GEO metric aims to quantify the quality of the spatial position of vertices in the predicted graph, ignoring any topological properties (the existence of edges between the vertices). To this end, following He *et al*. [15], we densely interpolate the predicted graph $G_{pred} = \{V_{pred}, E_{pred}\}$ and the ground-truth graph $G_{gt} = \{V_{gt}, E_{gt}\}$ such that any two adjacent vertices have the same distance from each other. Subsequently, a 1:1 matching between V_{pred} and V_{gt} is computed, giving rise to the matching precision and matching

Figure S.4. Exemplary visualizations of our lane regression model on random crops from the test split of our dataset.

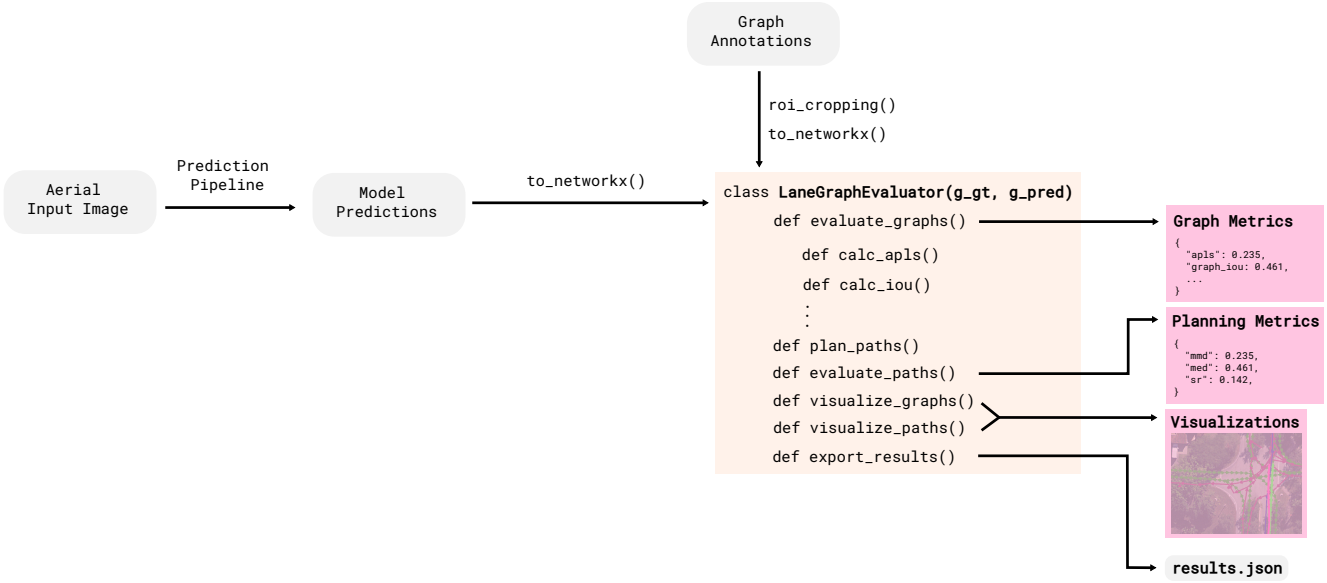


Figure S.5. Basic information flow for our graph evaluation benchmark. We provide basic routes for evaluating and visualizing lane graph predictions and annotations. Our evaluator expects the predicted graph(s) `g_pred` and the ground-truth graph annotations `g_gt` as `networkx` objects.

recall, denoted as GEO precision and GEO recall.

S.2.2. TOPO Metric

Building on top of the GEO metric, the TOPO metric takes vertex connectivity through edges into account as well. Given a vertex pair (V_{gt}, V_{pred}) obtained from the GEO metric, subgraphs $S_{gt}^{V_{gt}}$ and $S_{pred}^{V_{pred}}$ are created by walking a maximum distance D on the graphs. We select $D = 50\,\mathrm{m}$. The so-created sub-graphs may be compared according to the GEO metric and averaged over all sub-graphs to obtain the overall TOPO metric. The graph-walking procedure allows penalizing missing links or false-positive graph branches as they will produce poorly aligned sub-graphs S_{gt} and S_{pred}.

S.3. Target Graph Sampling for Lane Graph Learning

As described in the main manuscript, our LaneGNN model learns to predict node and edge scores for a proposal graph. The ground-truth graph according to the dataset annotations, however, cannot without modification be used as a learning target since its node positions do not correspond to the node points obtained from the Halton-sequence-based node sampling mechanism (see S.3.1). In the following, we describe how a target graph used for learning the LaneGNN model can be obtained from lane graph annotations.

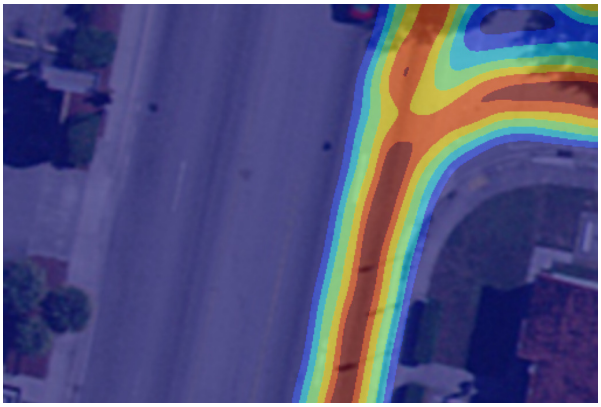


Figure S.6. The ego lane regressor model output, binned into 6 discrete thresholds from 0 to 1 for visualization purposes. The region of sampled node proposals for our LaneGNN model is sensitive to the choice of threshold cutoff value.

S.3.1. Node Sampling

In order to generate uniformly distributed 2D node positions in the rectangular image domain, we use Halton sequences. Their advantage over a per-dimension uniform random sampling of points in a 2D rectangle is their comparatively low discrepancy, leading to an approximately equal spacing of neighboring points across the 2D plane, rendering them the preferred choice for sampling random uniformly but non-regular distributed node proposals. For the experiments in our manuscript, we generate $N_{node} = 400$ Halton points for each sample. Finally, we filter the generated node positions based on the obtained ego centerline regression mask. Thus, the effective number of nodes is vastly decreased on average. We visualize the sampled Halton points for an exemplary scene in Fig. S.2d.

S.3.2. Edge Sampling

Due to the large number of possible edges in the number of nodes $|E| = \mathcal{O}(N_{node}^2)$, we sample edge proposals between pairs of nodes that have a Euclidean distance $d \in [d_{min}, d_{max}]$ such that only neighboring node are assigned a proposal edge. We visualize the sampled edges for an exemplary scene in Fig. S.2e. Furthermore, we only include edges in the edge proposal list that do not span over regions with low ego centerline regression output.

S.3.3. Node and Edge Scoring

Target node scores are a function of the proposal node distance to the ground-truth graph. More precisely, we score the node according to $s(n_i) = \big(1 - d_{L2}(n_i, G_{gt})\big)^8$ where $d_{L2}(\cdot, \cdot)$ denotes the euclidean distance. We also assign a binary *is-endpoint* label to each node. As described in the main manuscript, in addition to predicting node scores, our LaneGNN model also entails a node classification head, discriminating between lane endpoint nodes and non-endpoint nodes. This distinction allows for efficient proposal graph pruning during model inference. The positive *is-endpoint* label is assigned to the closest node in the proposal graph the ground-truth graph endpoint for each endpoint in the ground-truth graph.

For edge scoring, we empirically found that a binary scoring function (in contrast to a continuous scoring function for the node scores) leads to favorable graph learning performance. To produce this scoring, we leverage a two-step process: First, we evaluate a similar function to the node scoring function but add an angle penalty to the scoring function which penalizes a difference in the relative angle between the proposal edge and the closest edge e_{gt}^{prox} in the ground-truth graph. Concretely: $s(e_{ij}) = \big(1 - d_{\angle}(e_{ij}, e_{gt}^{prox}) d_{L2}(e_{ij}, G_{gt})\big)^8$. Second, we find minimum-cost paths from the lane starting node to the endpoint node(s) through the proposal graph, where the edge traversal cost is the reciprocal of the edge score. All edges along the minimum-cost paths from the start node to the end nodes are assigned a score of one while all remaining edges are assigned a score of zero.

S.4. Model Details

S.4.1. Image Skeletonization Baseline

Our image skeletonization baseline is obtained by first thresholding the predicted ego centerline regression model output to generate a binary image of high-likelihood successor lane graph regions. We subsequently skeletonize [32] the binary image and obtain a 1-pixel wide representation of this region (On-pixels in the binary image). Finally, we convert this representation into a graph by considering all On-pixels that have $n = 1$ neighbor pixels (lane starting point or endpoint) or $n >= 3$ neighbor pixels (lane split point), forming graph nodes. To obtain the graph edges, we add a graph edge for all pairs of nodes that are connected by regions of On-pixels that do not contain any other nodes between the two nodes in consideration.

Table S.1. Details on the used architecture of the LaneGNN model $\mathcal{M}$. All ReLU-activated MLP layers except for the ones under *Classification* which are ReLU-activated up to the last layer followed by Sigmoid.

Stage	Layer	Transformation				Parametrization
Encoding	Aerial edge features	$f^{e,bev}_{enc}(\mathbf{X}_{e,bev})$	$=$	$\mathbf{H}^{(0)}_{e,bev}$	$\in \mathbb{R}^{B\times E\times 64}$	ResNet-18 (non-pretrained)
	Geometric edge features	$f^{e,geo}_{enc}(\mathbf{X}_{e,geo})$	$=$	$\mathbf{H}^{(0)}_{e,geo}$	$\in \mathbb{R}^{B\times E\times 16}$	$\mathrm{MLP}(4, 8, 16)$, ReLU
	Node features	$f^{v}_{enc}(\mathbf{X})$	$=$	$\mathbf{H}^{(0)}_{v}$	$\in \mathbb{R}^{B\times N\times 16}$	$\mathrm{MLP}(2, 8, 16)$, ReLU
Fusion	Edge Feature Fusion	$f^{e}_{fuse}([\mathbf{H}^{(0)}_{e,bev}, \mathbf{H}^{(0)}_{e,geo}])$	$=$	$\mathbf{H}^{(0)}_{e}$	$\in \mathbb{R}^{B\times E\times 32}$	$\mathrm{MLP}(16+64, 64, 32)$, ReLU
Message Passing $l=1\ldots L$	Edge feature update	$f_e(\mathbf{H}^{(l-1)}_i, \mathbf{H}^{(l-1)}_{ij})$	$=$	$\mathbf{H}^{(l)}_{e}$	$\in \mathbb{R}^{B\times E\times 32}$	$\mathrm{MLP}(16+16+32, 64, 32)$, ReLU
	Node feature update	$f^{pred}_v(\mathbf{H}^{(l-1)}_i, \mathbf{H}^{(l-1)}_{ij}, \mathbf{H}^{(0)}_i)$	$=$	$\mathbf{H}^{(l)}_{v,pred}$	$\in \mathbb{R}^{B\times E\times 32}$	$\mathrm{MLP}(16+16+32, 64, 32)$, ReLU
		$f^{succ}_v(\mathbf{H}^{(l-1)}_i, \mathbf{H}^{(l-1)}_{ij}, \mathbf{H}^{(0)}_i)$	$=$	$\mathbf{H}^{(l)}_{v,succ}$	$\in \mathbb{R}^{B\times E\times 32}$	$\mathrm{MLP}(16+16+32, 64, 32)$, ReLU
		$f_v(\mathbf{H}^{(l)}_{i,pred}, \mathbf{H}^{(l)}_{i,succ})$	$=$	$\mathbf{H}^{(l)}_{v}$	$\in \mathbb{R}^{B\times E\times 32}$	$\mathrm{MLP}(16+16+32, 64, 32)$, ReLU
Classification	Edge Activation Scores	$f^{e}_{cls}(\mathbf{H}^{(L)}_e)$	$=$	$\mathbb{E}_e$	$\in \mathbb{R}^{B\times E\times 1}$	$\mathrm{MLP}(32, 16, 8, 1)$, ReLU + Sigmoid
	Node Activation Scores	$f^{v}_{cls}(\mathbf{H}^{(L)}_v)$	$=$	$\mathbb{S}_v$	$\in \mathbb{R}^{B\times E\times 1}$	$\mathrm{MLP}(16, 8, 4, 1)$, ReLU + Sigmoid
	Terminal Node Scores	$f^{v,t}_{cls}(\mathbf{H}^{(L)}_v)$	$=$	$\mathbb{T}_v$	$\in \mathbb{R}^{B\times N\times 1}$	$\mathrm{MLP}(16, 8, 4, 1)$, ReLU + Sigmoid

Table S.2. Training details for the three models contained in the full LaneGNN model $\mathcal{M}$.

	Batch Size	Epoch	Learning Rate	Weight Decay
Lane Regressor	8	50	10^{-3}	10^{-3}
Ego Lane Regressor	8	50	10^{-3}	10^{-3}
Graph Neural Network	2	100	10^{-3}	10^{-4}

S.4.2. Centerline Regression Architectures

For the *centerline regression* and the *ego centerline regression* models, we use the same model architectures. In both cases, we use a PSPNet architecture [33] with a ResNet-152 backbone. The number of input channels is 3 (RGB color channels) for the centerline regression model while it is 4 (RGB + 1 channel output of centerline regression model) for the ego centerline regression model. Additional training parameters can be found in Tab. S.2.

S.4.3. LaneGNN Architecture

The graph neural network architecture detailed in Table S.1 is trained for 100 epochs using a learning rate of 10^{-3} and a batch size of 2. The used optimizer is Adam with weight decay $\lambda = 10^{-4}$ and $\beta = (0.9, 0.999)$. Empirically, we found that the network is relatively insensitive to variations in the learning rate, batch size, and the number of message passing steps $L = 6$. This is further demonstrated in Tab. S.3.

S.5. Graph Aggregation

In order to aggregate our successive predictions into a globally consistent solution we take the parallelizable approach of running multiple `drive()`-instances (see Algo. 1) each starting at a different initial pose $\mathbf{p}_i = (x_i, y_i, \gamma_i)$ up to a certain maximum number of steps or a maximum number of branches is reached. Each initial pose $\mathbf{p}_i$ could either be the actual pose obtained from localization when driving or a pose inferred from a segmentation mask of the aerial image that also includes yaw regression. Next, we follow the procedure described in Algo. 1, which starts *driving* on the birds-eye view image. Sequentially, we predict a successor lane graph G_{pred} that is pruned via multiple runs of Dijkstra's algorithm using the predicted terminal node scores $\mathbb{T}_v$ as target nodes while neglecting already traversed corridors between runs.

Before merging the predicted graph G_{pred} into G_{agg}, we transform each predicted graph into global coordinates and employ multiple iterations of Laplacian smoothing:

$$\mathbf{X} \leftarrow (\mathbf{I} - \gamma \mathbf{L})\, \mathbf{X}\,, \tag{1}$$

which smoothens the original node positions $\mathbf{X}$ based on the graph Laplacian $\mathbf{L}$ of the undirected representation of G_{pred}. The graph Laplacian is given by

$$\mathbf{L} = \mathbf{D} - \mathbf{A}, \tag{2}$$

where $\mathbf{D}$ is the degree matrix and $\mathbf{A}$ is the adjacency matrix. Hereby, the position of each node is influenced by its first-degree neighbors. The scalar γ denotes a smoothing intensity parameter while $\mathbf{I}$ is the identity matrix. The smoothing evens out position irregularities obtained from the initial Halton sampling. In the following, we aggregate the smoothed graph G_{pred} with a consistent representation of all prior predictions G_{agg} (see Sec. S.5.1) and traverse it by steadily exploring the edge that shows the highest successor-tree weight up to a depth of 10. The depth is limited because

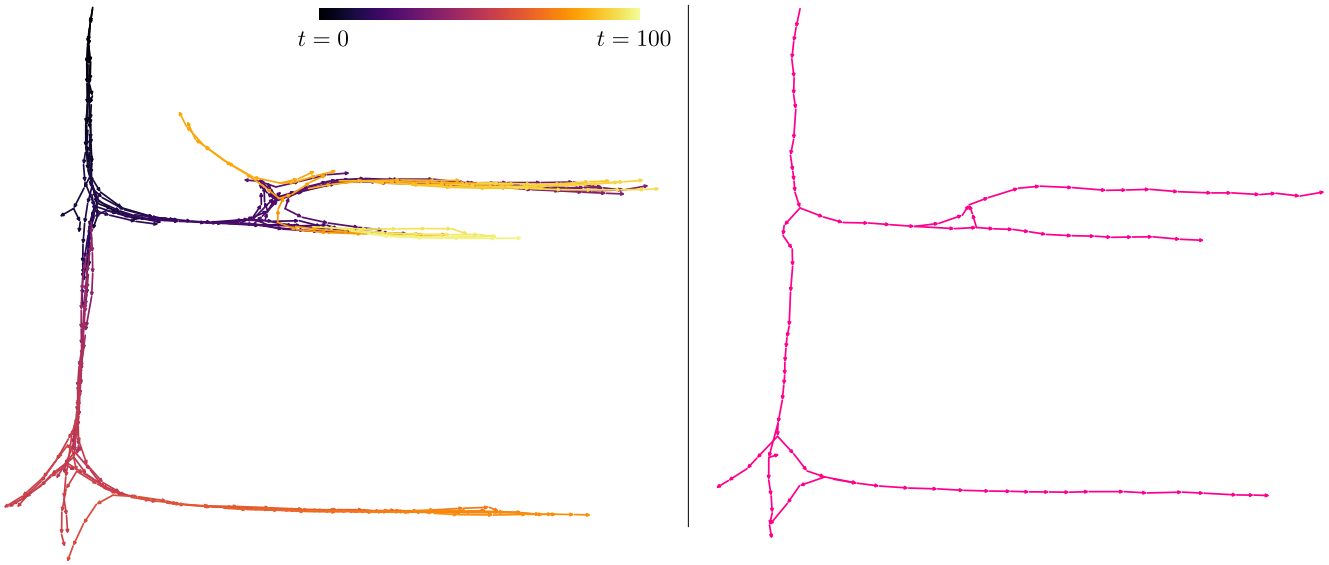


Figure S.7. Left: Visualization of consecutive successor graph predictions for $t \in [0, 1, ..., 100]$ time-steps. Right: The aggregated graph.

potentially existing loop closures of the lane graph would induce infinite tree weights. As predicted intersections provide multiple future branches of which only one is explored at a time, we queue the remaining ones for later exploration. As soon as the next edge is selected, we *step forward* by one edge length, which however is subject to hyperparameter optimization. After updating the pose, our approach is able to make predictions on new grounds while further updating and improving the aggregated lane graph representation G_{agg}. This is illustrated over 100 time steps in Fig. S.7.

Depending on the mode of operation, we are able to terminate a current branch as soon as a pose shows significant similarity to an already visited pose stored in a list. In addition, we also terminate a branch if a certain maximum number of steps is traveled across a number of branches combined or the number of explored branches exceeds a certain value. Furthermore, every single branch can also be terminated after a designated number of steps. If a branch suddenly terminates, we filter the currently aggregated graph G_{agg} by node out-degrees of two and larger to obtain split points with unvisited edges/poses that provide grounds for further exploration. The edge with the largest successor-tree weight is investigated next if exceeds a certain threshold. We have listed all used parameters in Sec. S.7.

Each and every aggregated output G_{agg} produced by one of the `drive()`-instances are finally aggregated one by one using the function `aggregate()` provided with Algo. 2, which is able to merge arbitrary graphs. In general, it would also be possible to aggregate multiple returns of `drive()` in an agglomerative fashion to increase inference speed for HD map creation. If necessary as a postprocessing step, we delete all but one parallel branch with the same source and end node if they do not show any intermediate branching and contain less than six consecutive edges.

Overall, we observe an average runtime of $1\,\mathrm{s}$ per model forward-pass and aggregation step on our development machine (NVIDIA RTX 3090, AMD EPYC CPU @ 3.65GHz). With parallel execution, processing a map tile ($10^6\,\mathrm{m}^2$) requires $\sim$7 min.

S.5.1. Lateral Aggregation Scheme

In the context of lane graphs, particular longitudinal node coordinates are merely a consequence of the chosen sampling distance. The proposed lateral graph aggregation scheme disregards deviations in longitudinal node position. Thus, we merge newly predicted graphs into an existing graph while updating it solely based on lateral deviations. Before doing so, we remove unvalidated splits and merges from the already existing aggregated graph by disregarding all splits or merges that do not exhibit a successor-edge tree of at least length three or do not show sufficient successor-tree (splits) or predecessor-tree (merges) weights. To do so, we construct local temporary aggregation graphs $LocalAggGraph(\mathbf{A})$ for each newly predicted node $\mathbf{A} \in G_{pred}$ as depicted in Fig. S.8. This is done to decrease the number of distance calculations necessary, which is crucial for large graphs G_{agg} containing thousands of nodes. Thus, each $LocalAggGraph(\mathbf{A})$ represents the region of G_{agg} in immediate vicinity of node $\mathbf{A} \in V_{pred}$ whereas $G_{pred} = (V_{pred}, E_{pred})$. In the next step, we obtain the nearest edge contained in G_{agg} as well as the lateral distance a to that edge. Similarly, we obtain the closest and second closest nodes **I** and **II** incident to that edge. If the lateral distance $a < a_{thresh}$, we see the grounds for updat-

Algorithm 1: drive($\mathbf{p}_{\text{init}}, \mathbf{I}_{\text{sat}}$)

$G_{agg} \leftarrow$ InitializeEmptyAggGraph()
$\mathbf{p} \leftarrow$ InitializePoseOnMap($\mathbf{p}_{init}$)
$\mathbf{p} \leftarrow$ PadSatImageSymmetrically($\mathbf{I}_{bev}$)
$stepCounter = 0$
$branchAlive = True$
$branchAge = 0$
$branchCounter = 0$
$numFutBranches = 1$
do
  if $stepCounter > maxSteps \vee branchCounter > maxNumBranches \vee branchAge > maxBranchAge$ **then**
    break
  if *branchAlive* **then**
    $branchAge = branchAge + 1$
  else
    $succEdges, numFutBranches \leftarrow$ GetUntraversedEdgesAtSplits(G_{agg})
    $\mathbf{p} \leftarrow$ GetNewPoseOfMaxScoreEdge($succEdges$)
  $G_I, \mathbb{S}^{ego}_{lane}, \mathbb{S}_{lane} \leftarrow$ ConstrAttribGraphManifold($\mathbf{I}_{bev}, \mathbf{p}$)
  $\mathbb{E}_e, \mathbb{S}_v, \mathbb{T}_e \leftarrow$ PredictSuccessorGraph(G_I)
  $\hat{G} \leftarrow$ PruneTraverseLaneGraph($G_I, \mathbb{E}_e, \mathbb{S}_v, \mathbb{T}_e$)
  $G_{pred} \leftarrow$ TransformLaneGraphToGlobalCoords($\hat{G}$)
  $G_{pred} \leftarrow$ ApplyLaplacianSmoothing(G_{pred})
  $G_{agg} \leftarrow$ Aggregate(G_{agg}, G_{pred})
  $branchAlive, \mathbf{p} \leftarrow$ StepForwAlongCurrBranch($\mathbf{p}, G_{agg}$)
while $numFutBranches \vee branchAlive$;
return G_{agg}

ing the node positions of **I** and **II** using a weighting-based scheme involving the weights of the involved aggregated nodes (obtained from previous aggregations) as well as the initial weight of each newly predicted node **A**. To do so, we calculate the angles α, β and, γ as given by Eq. 3 in order to further calculate the lengths b_1 and b_2 as shown in Fig. S.8:

$$\alpha = \arccos\left(\frac{a}{c_1}\right), \quad \beta = \arctan\left(\frac{d_y^{\mathbf{AI}}}{d_x^{\mathbf{AI}}}\right), \quad \gamma = \frac{\pi}{2} - \alpha - \beta \quad (3)$$

$$b_1 = c_1 \sin(\alpha), \qquad b_2 = c_2 \sin\left(\arccos\left(\frac{a}{c_2}\right)\right). \quad (4)$$

These lengths are used to measure the relative influence of **A** onto **I** as well as **II**, respectively. We create temporary nodes $\mathbf{A}'$ and $\mathbf{A}''$ to ease merging (Eq. 5) in the next step:

$$\mathbf{A}' = \mathbf{A} + b_1 \begin{bmatrix}\cos(\gamma)\\ \sin(\gamma)\end{bmatrix}, \quad \mathbf{A}'' = \mathbf{A} + b_2 \begin{bmatrix}\cos(\gamma)\\ \sin(\gamma)\end{bmatrix}. \quad (5)$$

Finally, we update the position of **I** and **II** using a weighting-based approach as described with Eq. 6 and Eq. 7 in the following:

$$\mathbf{I}^* = \frac{1}{\omega_{agg,I} + \omega_{A,I}} \left(\omega_{agg,I} \begin{bmatrix}I_x\\ I_y\end{bmatrix} + \omega_{A,I} \begin{bmatrix}A'_x\\ A'_y\end{bmatrix}\right), \quad (6)$$

$$\mathbf{II}^* = \frac{1}{\omega_{agg,II} + \omega_{A,II}} \left(\omega_{agg,II} \begin{bmatrix}II_x\\ II_y\end{bmatrix} + \omega_{A,II} \begin{bmatrix}A''_x\\ A''_y\end{bmatrix}\right), \quad (7)$$

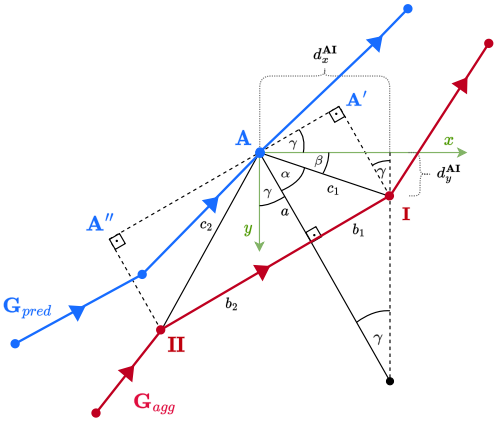


Figure S.8. Geometric visualization of our lateral graph aggregation scheme. We visualize the predicted graph G_{pred} in blue and the graph to be merged into G_{agg} in red.

Figure S.9. Additional qualitative results of our LaneGNN model. In the top row, we illustrate success cases. In the bottom row, we show interesting failure cases. The depicted graphs are not Laplace-smoothed.

where $\omega_{A,I}$ and $\omega_{A,II}$ are defined using c_1, c_2 as follows:

$$\omega_{A,I} = 1 - \frac{c_1}{c_1 + c_2}, \quad \omega_{A,II} = 1 - \frac{c_2}{c_1 + c_2}. \quad (8)$$

We collect all aggregation information using a merging map. All other nodes $m \in V_{pred}$ that were not mapped to any other node $k \in V_{agg}$ are added as new nodes to G_{agg}. Similarly, we add edges $(m, l) \in E_{pred}$ to E_{agg} in a slightly different fashion depending on whether one of the involved nodes has been already mapped to a node $k \in V_{agg}$. Finally, our `aggregate()` function returns G_{agg}.

S.6. Extended Results: Successor-LGP

In the following section, we perform additional ablation studies for our LaneGNN model on the Successor-LGP task.

Algorithm 2: aggregate($\mathrm{G_{pred}}, \mathrm{G_{agg}}$)

Add weight to each node $m \in V_{pred}$ based on weight equal to path length to starting pose
Compute edge angles and mean node angles based on predecessor and successor edge angles for G_{pred} and G_{agg}
$G_{agg} \leftarrow removeUnvalidatedSplitsMerges(G_{agg})$
$\mathbf{D}^{L2}_{agg,pred}$ Compute pair-wise Euclidean distance between V_{agg} and V_{pred}.
$\mathbf{D}^{\angle}_{agg,pred}$ Compute pair-wise mean angle distance between V_{agg} and V_{pred}.
$\mathbf{B}_{agg,pred} \leftarrow \left(\mathbf{D}^{L2}_{agg,pred} < \lambda\right) \wedge \left(\mathbf{D}^{L2}_{agg,pred} < \psi\right)$
for $\boldsymbol{A} \in V_{pred}$ **do**
    $LocalAggGraph(\mathbf{A}) = EmptyGraph()$
    for $k \in V_{agg}$ **do**
        if $\mathbf{B}_{agg,pred}(\boldsymbol{A}, k) == 1$ **then**
            $LocalAggGraph(\mathbf{A}) \leftarrow$ AddAggEdges(k)
        if $LocalAggGraph(\boldsymbol{A})$ *not empty* **then**
            $(\mathbf{II}, \mathbf{I}), a \leftarrow$ getNearestEdgeAndLatDist($\mathbf{A}, LocalAggGraph(\mathbf{A})$)
            $\mathbf{I}, \mathbf{II} \leftarrow$ getNearestNodes($m, LocalAggGraph(\mathbf{A}), (\mathbf{II}, \mathbf{I})$)
            if $a < a_{thresh}$ **then**
                Compute $\mathbf{I}^*$ and $\mathbf{II}^*$ as detailed in Eq. 6 and Eq. 7
                $G_{agg} \leftarrow$ UpdateAggGraph($\mathbf{I}^*, \mathbf{II}^*$)
for $m \in V_{pred}$ ***if*** m *unmapped to* G_{agg} **do**
    $G_{agg} \leftarrow$ AddUnmappedNode(G_{agg}, m)
for $(m, l) \in E_{pred}$ **do**
    if m *not mapped to* $G_{agg} \wedge l$ *not mapped to* G_{agg} **then**
        $G_{agg} \leftarrow$ AddUnconstrainedEdge$(G_{agg}, (m, l))$
    if m *mapped to* $G_{agg} \wedge l$ *not mapped to* G_{agg} **then**
        $G_{agg} \leftarrow$ AddLeadingEdge$(G_{agg}, (m, l))$
    if m *not mapped to* $G_{agg} \wedge l$ *mapped to* G_{agg} **then**
        $G_{agg} \leftarrow$ AddTrailingEdge$(G_{agg}, (m, l))$
return G_{agg}

S.6.1. Quantitative Results

S.6.1.1 Additional LaneGNN Ablation

In addition to our observation regarding low sensitivity to minor parameter variations (see Sec. S.4.3), we present a parameter study in Tab. S.3 for the LaneGNN architecture as well as the training parameters. The model is trained using both 200 as well as 400 node samples, which is a hyper-parameter that is chosen in the preprocessing stage. Note that the effective number of nodes covering the drivable corridor can be much lower in reality due to ego lane segmentation masking. We observe significantly higher recalls while the precision is roughly the same when increasing from 200 to 400 nodes. The remaining metrics, however, indicate a more drastic performance difference with plummeting values across $APLS, SDA_{20}, SDA_{50}$ and $GraphIoU$ for fewer nodes.

In addition to the number of nodes, we show that the inclusion of the node loss term improves recall significantly while also showing higher values across all other metrics. The node regression outputs $\mathbb{S}_v$ are not used during graph traversal but still induce significant performance improvements as they guide the network towards reasonable corridors. The batch size shows the best performance for a value of 2 or 4 depending on the evaluated metric while batch size 1 produces drastically worse outputs. In addition to the batch size, we have observed that a number of 6 to 8 message passing steps produces high precision as well as high recall. A GNN depth of 0 essentially renders the network a classic edge classifier without leveraging the graph structure, which results in drastically reduced performance. Our experiments show that the produced output graph is often not even traversable (using our pruning approach) under that limitation.

Lastly, we train three different GNN architectures: a small-, medium- and, large-size LaneGNN. Our findings support our chosen network architecture presented in the main paper and underpin that a larger network does not necessarily perform better given the test sets across all 4 cities. Overall, we observe that the parameter set used in the main paper (underlined values) yields maximum performance across the metrics evaluated.

S.6.1.2 City-wise Test Set Results

In the following, we evaluate the used LaneGNN architecture for different training sets as well as different splits of the test set each consisting of different combinations of cities. For each respective city-split, we show the performance of

Table S.3. Additional ablations of the LaneGNN model $\mathcal{M}$, measured across the test sets of all 4 cities combined. Underlined parameter values denote our presented approach used in the main paper, whereas bold values represent respective **best** values for each distinct ablation. The tuples under feature dimensions represent the main LaneGNN architecture dimensions (`map_feat_dim`, `node_dim`, `edge_dim`, `msg_dim`, `edge_geo_dim`), please see Table S.1.

Parameter		TOPO P/R ↑	GEO P/R ↑	APLS ↑	SDA_{20} ↑	SDA_{50} ↑	Graph IoU ↑
Number of Nodes	200	**0.622**/0.561	**0.622**/0.560	0.077	0.094	0.162	0.172
	400	0.600/**0.699**	0.599/**0.695**	**0.202**	**0.227**	**0.377**	**0.347**
Node Loss Term	✗	0.502/**0.702**	0.501/**0.699**	0.144	0.149	0.288	0.335
	✓	**0.600**/0.699	**0.599**/0.695	**0.202**	**0.227**	**0.377**	**0.347**
Batch Size	1	0.421/0.557	0.421/0.560	0.170	0.110	0.222	0.296
	2	**0.600**/0.699	**0.599**/0.695	**0.202**	**0.227**	0.377	0.347
	4	0.581/**0.701**	0.579/**0.696**	0.162	0.220	**0.387**	**0.349**
GNN Depth	0	0.163/0.168	0.163/0.166	0.057	0.017	0.051	0.098
	2	0.517/0.688	0.517/0.684	0.144	0.152	0.310	0.331
	4	0.526/0.684	0.525/0.680	0.145	0.162	0.320	0.330
	6	**0.600**/0.699	**0.599**/0.695	**0.202**	**0.227**	**0.377**	0.347
	8	0.570/**0.710**	0.568/**0.704**	0.163	0.195	0.350	**0.349**
Feature Dimensions	(16, 8, 16, 16, 8)	0.599/0.655	0.598/0.650	0.114	0.072	0.178	0.234
	(64, 16, 32, 32, 16)	0.600/**0.699**	0.599/**0.695**	**0.202**	**0.227**	**0.377**	**0.347**
	(128, 32, 48, 48, 16)	**0.616**/0.695	**0.613**/0.688	0.162	0.180	0.310	0.332

Table S.4. Additional ablations of the LaneGNN model $\mathcal{M}$ for the *Successor-LGP* task trained on respective cities as detailed and evaluated on various combinations of city-respective test sets. PAO, ATX, MIA, and PIT represent the cities of Palo Alto, Austin, Miami, and Pittsburgh, respectively. Bold entries denote the maximum value for the given evaluation set under different training sets. Underlined values denote the numbers presented in the main paper.

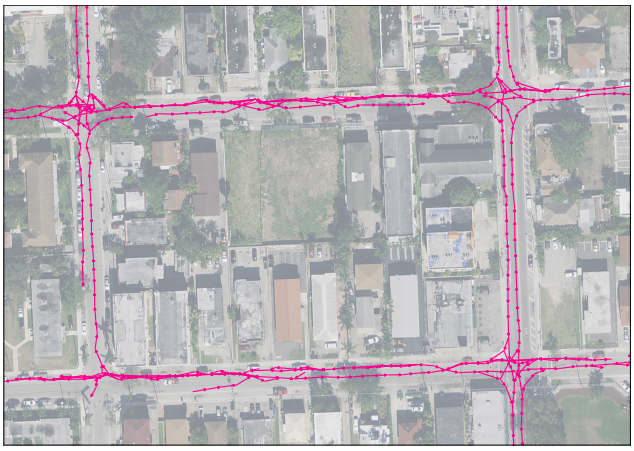
Train Set				Eval Set				TOPO P/R ↑	GEO P/R ↑	APLS ↑	SDA_{20} ↑	SDA_{50} ↑	Graph IoU ↑
PAO	ATX	MIA	PIT	PAO	ATX	MIA	PIT						
✓				✓				0.584/**0.744**	0.582/**0.739**	0.177	**0.220**	**0.367**	**0.378**
✓	✓	✓	✓	✓				**0.666**/0.702	**0.663**/0.696	**0.206**	0.199	0.337	0.366
	✓				✓			0.468/**0.726**	0.468/**0.727**	0.123	**0.227**	0.304	**0.327**
✓	✓	✓	✓		✓			**0.579**/0.660	**0.578**/0.660	**0.194**	0.206	**0.361**	0.301
		✓				✓		0.534/**0.687**	0.532/**0.683**	0.145	**0.217**	0.354	**0.330**
✓	✓	✓	✓			✓		**0.643**/0.672	**0.638**/0.666	**0.193**	0.198	**0.371**	0.315
			✓				✓	0.530/**0.722**	0.532/**0.714**	0.143	**0.151**	**0.307**	**0.336**
✓	✓	✓	✓				✓	**0.667**/0.674	**0.667**/0.665	**0.191**	0.115	0.243	0.333
✓				✓	✓	✓	✓	**0.603**/0.675	**0.601**/0.670	**0.203**	0.206	0.376	0.339
	✓			✓	✓	✓	✓	0.549/0.686	0.548/0.681	0.200	0.185	0.338	0.338
		✓		✓	✓	✓	✓	0.578/0.663	0.577/0.658	0.202	0.174	0.348	0.330
			✓	✓	✓	✓	✓	0.577/0.643	0.574/0.635	0.171	0.073	0.179	0.281
✓	✓	✓	✓	✓	✓	✓	✓	0.600/**0.699**	0.599/**0.695**	0.202	**0.227**	**0.377**	**0.347**

a LaneGNN instance trained on either only the respective city or on all cities (see Tab. S.4, upper half). In general, we observe that the specifically trained models do not necessarily perform better on the test-sets of their respective cities. In comparison, the LaneGNN model trained on all cities consistently produces higher precision while showing comparably small reductions in recall. This allows us to state that larger training sets across all cities do not necessarily harm the performance on specific cities when evaluating.

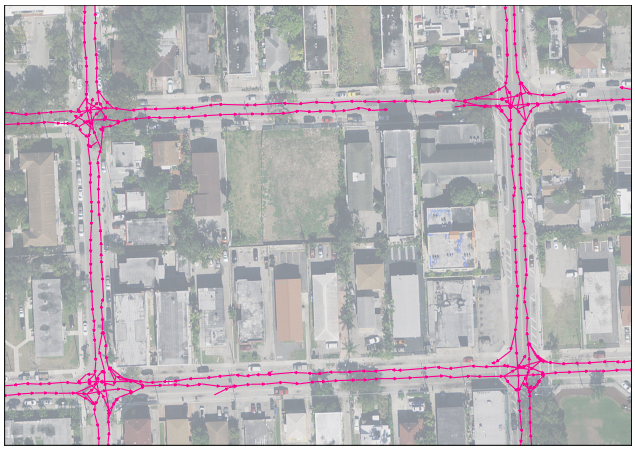
In addition to these findings, we also evaluate different LaneGNN instances trained on city-wise training sets and measure their performance across all cities combined. We observe no major performance drops when testing a city-specific model on all cities except for the model trained on Pittsburgh (see Tab. S.4, lower half). This essentially means that we can use a model trained on one city and still perform reasonably well in other cities. Nonetheless, the model trained on all cities still shows the average best performance across all cities.

S.6.2. Qualitative Results

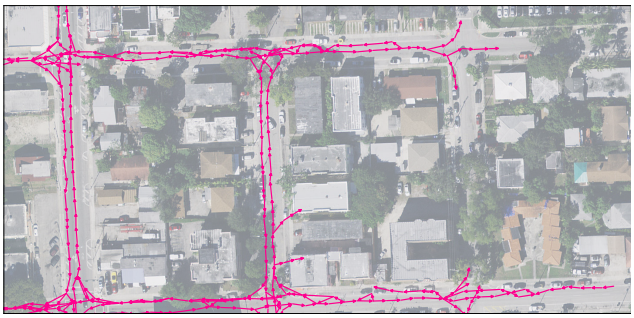
In Fig. S.9, we visualize the success and failure cases of our LaneGNN model for the Successor-LGP task. Challenging scenes typically entail complex topological structures, such as roundabouts, multi-lane streets, and high-contrast illumination scenarios. In the depicted high-contrast scene,



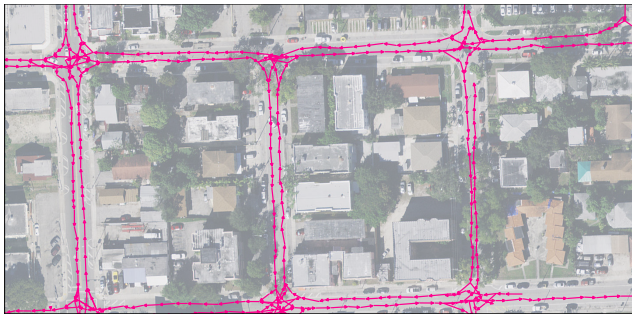
(a) Region 1, naive aggregation



(b) Region 1, our aggregation



(c) Region 2, naive aggregation



(d) Region 2, our aggregation

Figure S.10. Visualizations of the naive and our aggregation scheme for two large-scale areas within the testing region of the city of Miami.

the lane turning right is not detected, leading to a missing branch in the predicted lane graph. The roundabout depicted in the bottom center leads to a topologically correct predicted successor graph, but the predicted waypoints of the left-turning lane do not align with the position of the roundabout center. Finally, in the bottom right scene, an additional left-turning lane going in the opposite direction is predicted by our LaneGNN model. Our aggregation scheme can remove these false-positive graph branches if they are not consistently predicted for multiple successive LaneGNN forward passes.

S.7. Extended Results: Full-LGP

In order to generate the test-set results provided in the main paper, we initialized our `drive`($\mathbf{p}_{init}$) function (Algo. 1) starting at the predicted lane start points as provided by the yaw-segmentation output of LaneExtraction [15]. This results in 178 initial poses parametrized by $\mathbf{p}_i = (x_i, y_i, \gamma_i)$ used for parallel execution of `drive()` as described in Algo. 1. Our numbers provided in the main paper are generated using the parameters detailed below.

We make use of thresholding $\mathbf{S}^{ego}_{lane}$ at a value of 0.15, which produces relatively high recalls and thus more connections at intersections. This is complemented by only considering edges with an edge score of 0.5 or higher for graph traversal. For aggregation, we use a radius and distance threshold of 80 px and 0.5 radians to filter close nodes and edges of G_{agg} for constructing $LocalAggGraph(\mathbf{A})$ instances to speed up the aggregation process. For the actual merging of nodes (Eq. 6 and Eq. 7), we choose a lateral distance $a_{thresh} = 20$ that is used for `aggregate()` within one `drive()` instance as well as aggregation of multiple `drive()` outputs (see Algo. 2). Regarding the `drive()` function, the maximum overall number of steps is 36 over a

Table S.5. Additional ablations of the aggregation scheme used in the Full-LGP task. We compare the presented naïve and full aggregation scheme with the reduced variants of the full pipeline (no smoothing of G_{pred}, no removal of invalidated splits and merges, and a smaller lateral aggregation threshold).

Model	APLS ↑	TOPO P/R ↑	GEO P/R ↑	Graph IoU ↑
w/o smoothing	0.105	0.452/0.671	0.631/0.726	0.377
w/o remove s/m	0.102	0.480/0.658	0.634/0.698	0.366
$a_{thresh} = 10$	**0.108**	0.458/**0.677**	0.581/**0.738**	0.354
Naïve	0.101	0.366/0.654	0.523/0.727	0.376
Ours	0.103	**0.481**/0.670	**0.649**/0.689	**0.384**

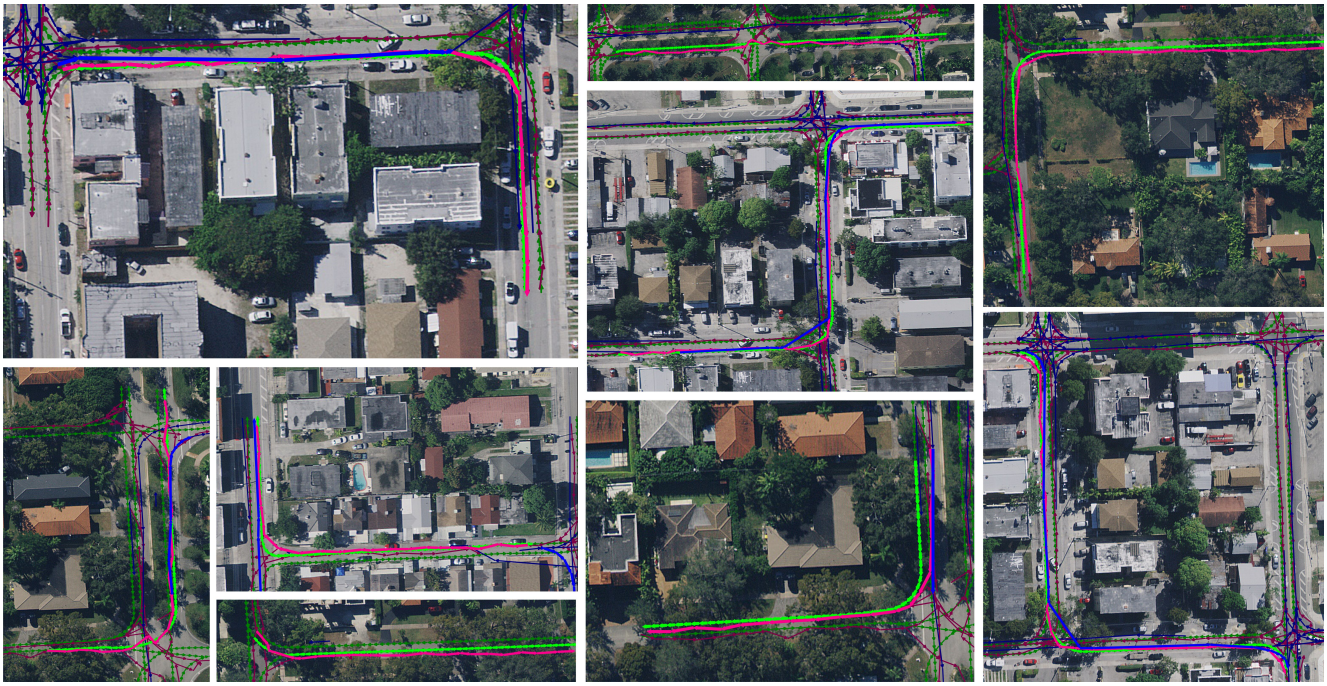


Figure S.11. Illustrative path planning experiment results. We visualize the ground-truth graph in thin green and the path planned on the ground-truth graph in bold **green**. We employ the same visualization scheme for the graph from LaneExtraction, and our aggregation scheme.

maximum of 4 branches with a maximum branch-age of 12 each. As described before we smooth the predicted graphs and also remove splits and merges.

In addition, we present results on three additional parameter settings for the *Full-LGP* task in Tab. S.5. We observe slight performance decreases in precision while the recall is similar or higher when not smoothing predicted graphs. Without removing unvalidated splits and merges, our performance shows slight decreases for GraphIoU, GEO precision, and TOPO P/R. Finally, we lower a_{thresh} to 10, which increases recall but lowers precision, specifically the GEO metric, while the Graph IoU also drops by three percent. As in the main paper, we also list our naïve aggregation scheme for comparison. The results are further illustrated in Fig. S.10.

S.8. Extended Results: Planning

As described in the main manuscript, we evaluate the quality of the generated lane graph on a planning task. Fig S.11 illustrates exemplary planned paths obtained on the graphs from the ground-truth annotations, LaneExtraction, and our aggregation scheme, respectively. We observe that for most scenarios, the planned paths from our aggregation scheme closely match the paths planned on the ground-truth graph. Even for long routes with multiple turns, we report a close match between our path and the ground-truth path. We notice some inaccuracies in our planned path where the trajectory deviates from the correct lane and contains slightly misaligned path waypoint positions.

While some of the paths from LaneExtraction show encouraging results, we observe that for many routes, no close match between the LaneExtraction path and the ground-truth path can be observed, also mirrored in the quantitative evaluation of the planning task in the main manuscript. We hypothesize that the LaneExtraction graphs are not always complete and have missing links for occluded or topologically complex regions, resulting in the nonexistence of a path through the graph from start to goal nodes.