Bézier Everywhere All at Once: Learning Drivable Lanes as Bézier Graphs

Hugh Blayney[1] Hanlin Tian[2] Hamish Scott[1]
Nils Goldbeck[1] Chess Stetson[1] Panagiotis Angeloudis[2]
[1]dRISK.ai [2]Imperial College London
{hugh, hamish, nils, chess}@drisk.ai {h.tian22, p.angeloudis}@imperial.ac.uk

Abstract

Knowledge of lane topology is a core problem in autonomous driving. Aerial imagery can provide high resolution, quickly updatable lane source data but detecting lanes from such data has so far been an expensive manual process or, where automated solutions exist, undrivable and requiring of downstream processing. We propose a method for large-scale lane topology extraction from aerial imagery while ensuring that the resulting lanes are realistic and drivable by introducing a novel Bézier Graph *shared parameterisation of Bézier curves. We develop a transformer-based model to predict these Bézier Graphs from input aerial images, demonstrating competitive results on the* UrbanLaneGraph *dataset. We demonstrate that our method generates realistic lane graphs which require both minimal input, and minimal downstream processing. We make our code publicly available at* https://github.com/driskai/BGFormer

1. Introduction

Autonomous Vehicles (AVs) require knowledge of their surroundings to operate. So far, all systems not reliant on a safety driver have required pre-built High Definition maps (HD maps). HD maps can deliver a strong prior about the road and lane topology, e.g. so the AV can navigate amid occlusions. But creation of HD maps is time-consuming and expensive, typically requiring a fleet of road vehicles equipped with LiDAR and cameras followed by extensive manual curation and human annotation [18].

One of the core components of an HD map is precise lane geometry and topology data. Previous works have attempted to automate the generation of this data from onboard sensors [4, 5, 10] or - recently - from aerial imagery [3, 9, 30, 31]. Generating lane-level data from aerial imagery is of particular interest since this can be efficiently obtained at large scale using Unmanned Aerial Vehicles (UAVs) or satellites. In this way, AVs could be equipped with regularly updated HD maps that cover a large area.

However, these approaches are not without their challenges. Lane geometry and topology is typically predicted in the form of a *lane graph*: a graph in which nodes represent a discrete sampling of lane centre lines, with edges representing connectivity. Existing methods typically predict the position of each node in 2D space, but this frequently results in noisy, non-physical lane centre lines requiring extensive downstream processing and human oversight.

We improve on these methods by introducing a shared parameterisation of cubic Bézier curves in a graph structure that we refer to as a *Bézier Graph*. By associating Bézier direction vectors with nodes, we enforce a strong prior that lane direction should be continuous at boundaries.

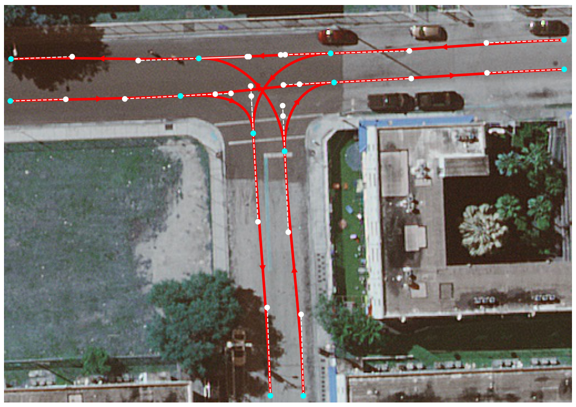


Figure 1. Our trained model detecting a lane Bézier Graph on an intersection. Bézier endpoints $\boldsymbol{P}_{0,3}$ shown in cyan, control points $\boldsymbol{P}_{1,2}$ in white, resulting curves in red.

For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) license to any Author Accepted Manuscript version arising. This work was funded by the Centre for Connected and Autonomous Vehicles, delivered by Innovate UK, part of UK Research and Innovation.

By associating cubic Béziers with edges, we efficiently represent high-fidelity lane geometry while encouraging realistic, smooth lanes. We develop an end-to-end transformer-based method to generate Bézier Graphs from aerial images, demonstrating competitive benchmark performance and increased drivability of the generated lane graphs.

To summarise, our main contributions are as follows:

- We introduce a novel *Bézier Graph* of lane topology and geometry, achieving smoother lanes and direction continuity throughout the network. We demonstrate a method for generating these from existing lane graphs.
- We develop BGFormer (Bézier Graph Transformer), an end-to-end transformer-based method to directly generate Bézier Graphs from aerial images.
- We evaluate our work on the UrbanLaneGraph [3] dataset, validating the advantages of our method. We highlight additional benefits, including drivability and faster inference.

2. Related Work

2.1. Road and Lane Detection

Lane graph detection is closely related to the problem of road graph detection. However, they work at different scales; road graphs will typically cover a much greater area but without distinguishing individual lanes. As such, road graphs are useful for coarse navigation and routing tasks, but not independently suitable for AV deployment.

Road graph detection methods typically utilise aerial images and, as noted by [25], typically follow one of two approaches: image segmentation followed by processing to obtain a graph, or direct graph detection. With more relevance to our work, we focus on the latter category. Several approaches iteratively construct road graphs by predicting adjacent nodes from the current node [1, 12]. Sat2Graph [2] additionally encodes an input image into a fixed dimensional context vector using an encoder-decoder structure. RNGDet and RNGDet++ [25, 27] are recent transformer-based approaches which iteratively predict sets of vertices.

Lane graph detection can be broadly split into two categories: methods using onboard sensors and methods using remote aerial imagery. In the former category, HDMapNet [11] uses onboard sensors to predicts a surrounding rasterised HD map, which is postprocessed into a vectorised map. VectorMapNet [16] learns end-to-end vectorised HD maps, similarly from a system of onboard cameras and LIDAR. CenterLineDet [26] uses an RNGDet-style [25] vertex buffer-and-update system to iteratively create vectorised lane centrelines from fused onboard sensor data.

A few papers in this area are of particular relevance as they also utilise Bézier curves. Several [8, 20] represent disconnected lanes using Bézier curves, but do not aggregate these into a graph. LaneGAP [13] detects separate lane paths, representing these as Bézier curves, then discretises these paths into a sequence of nodes before aggregating the nodes into a lane graph. By contrast, our method is the only one to *directly predict* a graph where edges encode Bézier curves and both positions and *directions* are shared across nodes, thus retaining the Bézier parameterisation in the graph representation. This affords a stronger enforcement of direction continuity and, uniquely, allows the Bézier curves representing lanes to smoothly branch into multiple other curves.

Lane detection using aerial imagery has seen less research. LaneExtraction [9] was the first to generate "routable" lane centrelines from aerial imagery, extracting lanes at non-intersection areas and then enumerating all possible turning lanes. LaneGNN [3] removed the distinction between intersection and non-intersection inference, estimating reachable lanes from a starting point using a semantic segmentation network; sampling nodes and edges to fill this segmentation and then training a Graph Neural Network (GNN) to filter the resulting graph. Zürn *et al.* [31], detected lane graphs from aerial images using only recorded vehicle tracklets as signal, omitting human annotation.

2.2. Transformers for Graphs in Computer Vision

Until the recent pioneering work of Relationformer [21], graph generation from images was specialised to particular research areas; for example, road and lane graph generation (see above), extracting biological "vessels" from 3D data [19, 22], scene graph generation from 2D images [17, 24].

Relationformer [21] presented a unified image-to-graph generation approach, using a DETR-style [6] transformer network to encode the input image and propose node "queries" located within the image. This work achieved state-of-the-art performance across a range of benchmarks and datasets, including several relating to road graph generation. However, direct application of this method to the challenge of lane graph prediction is made difficult by a significant increase in graph size; the lane graph crops studied here are significantly larger than the 200 "object" (node) tokens of the original architecture, an issue due to $\mathcal{O}(N^2)$ attention scaling. Our method can be seen as - alongside other improvements - addressing this scale issue with a more efficient Bézier Graph parameterisation of lane geometry.

Our work is the first to introduce Bézier Graphs and jointly parameterise Bézier curves over an entire lane network, thus distinguishing us from prior work that models disconnected lanes with independent Bézier curves, or discards the Bézier parameterisation through discretisation before merging into a graph. In addition, to the best of our knowledge our architecture is the first to detect lane graphs from aerial imagery in a fully end-to-end fashion, contrasting with existing multi-stage processes and resulting in a simpler, more efficient lane prediction pipeline.

3. Method

Our approach is comprised of three components: Sec. 3.1 describes our Bézier Graph parameterisation and fitting method, Sec. 3.2 introduces BGFormer, a transformer-based model for generating Bézier Graphs from aerial imagery, and Sec. 3.3 describes our algorithm for aggregating graphs into a consistent global Bézier Graph.

3.1. Bézier Graphs

A lane graph is defined as a directed graph $\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l)$ comprising a set of nodes $\mathcal{V}_l$ and directed edges $\mathcal{E}_l \subseteq \mathcal{V}_l \times \mathcal{V}_l$. Nodes are attributed with 2D positions $\mathbf{X}_l \in \mathbb{R}^{|\mathcal{V}_l| \times 2}$, edges represent valid movements between nodes in such a way that a single lane is followed i.e. edges representing lane changes are not included, but a node may have multiple successors where a lane branches. Lanes, as in the source Argoverse [7] dataset, are defined generally as a segment of road where cars drive in a single-file fashion in a single direction, and as such include the "implied" lanes for turning or at intersections.

A Bézier Graph is also defined as a directed graph $\mathcal{G}_b = (\mathcal{V}_b, \mathcal{E}_b)$, $\mathcal{E}_b \subseteq \mathcal{V}_b \times \mathcal{V}_b$. Nodes in this graph represent the beginnings and ends of cubic Bézier curves, and edges represent the curves themselves. In addition to position, nodes are attributed with direction unit vectors $\mathbf{D}_b \in \mathbb{R}^{|\mathcal{V}_b| \times 2}$, and edges with two *length* values, $\mathbf{E}_b \in \mathbb{R}_+^{|\mathcal{E}_b| \times 2}$.

Each edge in $\mathcal{G}_b$ represents a cubic Bézier curve in the following manner. Consider edge $(v_i, v_j) \in \mathcal{E}_b$, where nodes v_i, v_j have positions $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbb{R}^2$ and directions $\hat{\boldsymbol{d}}_i, \hat{\boldsymbol{d}}_j \in \mathbb{R}^2$ respectively. The edge has attributes $(\ell_1, \ell_2)^T \in \mathbb{R}_+^2$. The associated curve, parameterised by $t \in [0, 1]$ is defined as

$$\begin{aligned} \boldsymbol{B}(t) &= \sum_{r=0}^{3} \binom{n}{r} (1-t)^{n-r} t^r \boldsymbol{P}_r, \\ \boldsymbol{P}_0 &= \boldsymbol{x}_i, \qquad \boldsymbol{P}_1 = \boldsymbol{x}_i + \ell_1 \hat{\boldsymbol{d}}_i, \\ \boldsymbol{P}_2 &= \boldsymbol{x}_j - \ell_2 \hat{\boldsymbol{d}}_j, \qquad \boldsymbol{P}_3 = \boldsymbol{x}_j. \end{aligned} \tag{1}$$

Note this choice of $\mathbf{X}_b, \mathbf{D}_b, \mathbf{E}_b$ parameterisation encodes an important inductive bias: positions and directions are associated with *nodes* and thus shared between connected curves, ensuring smooth direction continuity where lanes merge and split. The direction lengths $\boldsymbol{\ell}$ are however associated with individual edges, allowing edges incident to the same node to have different control points $\boldsymbol{P}_{1,2}$. This is visualised in Fig. 1 in which the dashed white lines represent direction vectors, white dots represent control points $\boldsymbol{P}_{1,2}$.

In this new formulation, the edges of the graph are more expressive than the edges of the source lane graph, no longer only representing linear transitions. This additional expressiveness typically affords $|\mathcal{V}_b| \ll |\mathcal{V}_l|$ when representing the same underlying road network.

We introduce an algorithm for converting a standard lane graph to a Bézier Graph via an optimisation routine that aims to fit Bézier curves to paths in the input graph. The aim of the routine is to choose a subset $\mathcal{V}_b \subseteq \mathcal{V}_l$ and optimise the direction parameters $\mathbf{D}_b$ and the length parameters $\mathbf{E}_b$ such that the curves closely represent the input graph. That is, every edge in the input graph is closely matched by a segment of a curve in the Bézier Graph and vice versa. The choice of subset should additionally be motivated by the underlying road network geometry. This is important to allow the network of Sec. 3.2 to learn to predict the positions of the nodes from an image.

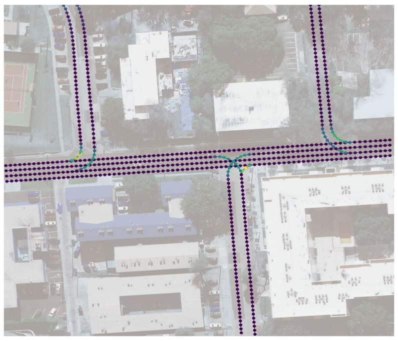
The subset $\mathcal{V}_b$ is determined using a two-step process shown in Fig. 2b. First, we select all nodes which have in or out degree not equal to 1; these are lane endpoints, including locations where lanes merge or split. We connect these nodes with edges where there exists a path between them that does not contain any of the selected subset of nodes. Then, we further split these edges where there exist large changes of curvature - this avoids long paths with multiple turns which are inadequately fit with cubic Bézier curves.

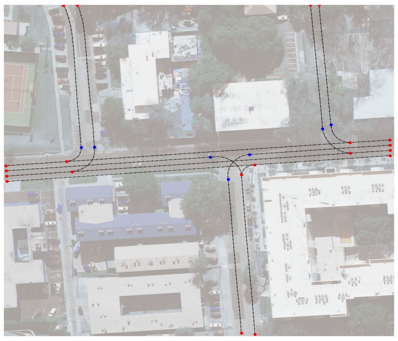
The direction and length parameters $\mathbf{D}_b, \mathbf{E}_b$ are optimised in the following way. For each node in input graph $\mathcal{G}_l$ we can associate a corresponding edge in $\mathcal{E}_b$ due to the surjective mapping described in the previous paragraph. We associate each node with a Bézier parameter $t \in [0, 1]$; following the example of [15], t is calculated as the fraction of the cumulative length of the source graph path at which the original node lies. Note this mapping $\mathcal{V}_l \to \mathcal{E}_b \times [0, 1]$ needs only be computed once, before optimisation. For given values of $\mathbf{D}_b, \mathbf{E}_b$, we then compute a corresponding location in the Bézier Graph, yielding a matrix of predicted positions $\mathbf{F}_{\mathcal{G}_l}(\mathbf{D}_b, \mathbf{E}_b) \in \mathbb{R}^{|\mathcal{V}_l| \times 2}$. We choose $\mathbf{D}_b, \mathbf{E}_b$ by minimising the L_2 distance $||\mathbf{X}_l - \mathbf{F}_{\mathcal{G}_l}(\mathbf{D}_b, \mathbf{E}_b)||_2$. Note this requires joint minimisation of the Bézier parameters across the entire graph; for this we used gradient descent, and discovered that the Adam optimiser was highly efficient for this task. An example result of this Bézier fit can be seen in Fig. 2c. We describe the optimisation procedure in greater detail in the supplementary material, and a JAX implementation can be found in our provided code.

3.2. Bézier Graph Transformer

We learn to generate Bézier Graphs in a supervised fashion, given input aerial image $\mathbf{I}$ and associated target $\mathcal{G}_b$. Our architecture is based on Relationformer [21], but with adaptations to predict Bézier Graphs. The model is described below, with architecture overview in Fig. 3.

Transformer Backbone: Our transformer encoder-decoder uses the deformable-DETR [29] style architecture of Relationformer: a convolutional backbone is used to first extract image features which are passed into a transformer encoder, generating an image encoding. A transformer de-


(a) Ground truth graph, where node colour represents curvature. High curvature is yellow, low is dark blue.


(b) Assignment of Bézier endpoints. Lane endpoints shown in red, additional endpoints due to curvature changes shown in blue.

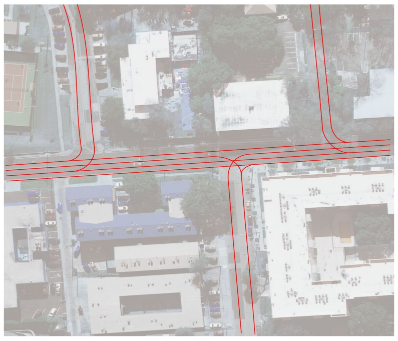

(c) Result of the Bézier fit.

Figure 2. The Bézier Graph fitting process, starting from a ground truth lane graph.

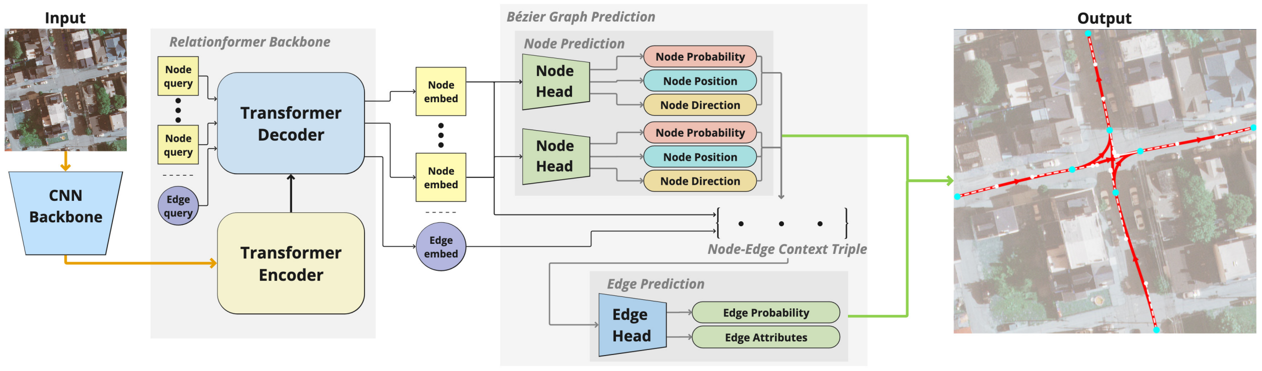


Figure 3. Illustration of the BGFormer model. Input image embeddings are generated by a CNN backbone and transformer encoder. A transformer decoder combines node and edge queries with the encoder output through self- and cross-attention, generating a set of node embeddings and a single edge embedding. A node head acts on each node embedding to predict its probability, position and direction. Predicted node positions and directions are concatenated with the embedding vectors to produce node context vectors, and pairs of these are concatenated with the edge embedding to produce context triples. An edge head acts on these triples to predict edge probability and attributes. The predicted values are combined to yield the final Bézier Graph.

coder applies cross-attention to N learnable queries and the encoder output, generating N query embeddings. The first $N-1$ of these embeddings are treated as node tokens $\boldsymbol{n}_i, 1 \leq i \leq N-1$, and the remaining embedding is treated as a single edge token $\boldsymbol{e}$.

Node Head: We use a multilayer perceptron (MLP) to regress node positions and directions:

$$[\tilde{\boldsymbol{x}}_i', \tilde{\boldsymbol{d}}_i] = \text{MLP}^n(\boldsymbol{n}_i). \tag{2}$$

Since the target output position is normalised to $[0,1]$ we apply a sigmoid non-linearity to obtain the final predicted position, $\tilde{\boldsymbol{x}}_i = \sigma\left(\tilde{\boldsymbol{x}}_i'\right)$. For predicting the probability of a given token representing a node as opposed to the no object class, we use a single linear layer,

$$\tilde{p}_i = \sigma\left(\mathbf{W}\boldsymbol{n}_i + \boldsymbol{b}\right). \tag{3}$$

Edge Head: Our edge head also uses an MLP followed by a sigmoid non-linearity. This predicts the existence and attributes of an edge between any two given nodes i, j. For each pair i, j it takes in a *context triple* consisting of context vectors for nodes i and j, and the edge context token $\boldsymbol{e}$. The node context vectors are their corresponding object tokens $\boldsymbol{n}_i, \boldsymbol{n}_j$ concatenated with their node-head predicted attributes. The edge head outputs a predicted edge probability p_{ij} and normalised edge length values $\tilde{\boldsymbol{\ell}}_{ij}$:

$$\boldsymbol{c}_i = [\boldsymbol{n}_i, \tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{d}}_i], \quad \boldsymbol{c}_j = [\boldsymbol{n}_j, \tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{d}}_j], \tag{4}$$

$$[p_{ij}, \tilde{\boldsymbol{\ell}}_{ij}] = \sigma\left(\text{MLP}^e([\boldsymbol{c}_i, \boldsymbol{e}, \boldsymbol{c}_j])\right). \tag{5}$$

Similar to the node positions, these edge lengths are normalised to $[0,1]$ and then multiplied by the image dimension to obtain the final predicted edge lengths.

3.2.1 Loss Function and Graph Matching

To train the prediction model, we require a matching between the predicted - unordered - model outputs and the nodes and edges of the ground truth Bézier Graph. For this we take the DETR [6] approach of using the Hungarian algorithm, matching the predicted and ground truth nodes according to the predicted class probabilities and node attributes, and deriving the edge matching from this.

For node attributes, we use the L_1 regression loss: we denote this $\mathcal{L}_p^n$ for the component corresponding to node position loss and $\mathcal{L}_d^n$ for the node direction. Note that these are computed on the nodes which are assigned the "node" class by the Hungarian matcher, as these are the only nodes which have a ground truth position and direction. For the node classification loss $\mathcal{L}_c^n$, we use the standard Deformable DETR Focal Loss [14, 29].

To compute edge loss, we follow the example of Relationformer, considering only edges between nodes which have been matched to ground truth nodes. Denote the matching of node i by c_i where $c_i = \emptyset$ indicates that c_i is not matched to a ground truth node. Define the set of possible edges between these matched nodes as E_{poss}:

$$E_{\text{poss}} = \{(i,j)\,|1 \le i,j \le N-1 \wedge (c_i \neq \emptyset) \wedge (c_j \neq \emptyset)\}\,. \tag{6}$$

From this we define the subset which exist in the ground truth set $\mathcal{E}_b$ as E_{valid}:

$$E_{\text{valid}} = E_{\text{poss}} \cap \mathcal{E}_b. \tag{7}$$

From this, we can define the set of *background* edges - those which are possible but which don't exist in the ground truth:

$$E_{\text{background}} = E_{\text{poss}} \setminus E_{\text{valid}}. \tag{8}$$

Following relationformer, we define the edge probability loss $\mathcal{L}_{\text{prob}}^e$ as the binary cross entropy (BCE) loss between the predicted edge probabilities and the ground truth edge probabilities, taking as our edge samples all of E_{valid} and a random sample of $E_{\text{background}}$ such that we sample three background edges for every valid edge. We define the edge attribute loss $\mathcal{L}_a^e$ as the MSE loss between $\boldsymbol{\ell}_{ij}$ and the ground truth edge lengths, computed only over E_{valid}.

The total loss is then computed as a weighted sum of all of the aforementioned losses:

$$\mathcal{L} = \lambda_p^n \mathcal{L}_p^n + \lambda_d^n \mathcal{L}_d^n + \lambda_c^n \mathcal{L}_c^n + \lambda_{\text{prob}}^e \mathcal{L}_{\text{prob}}^e + \lambda_a^e \mathcal{L}_a^e. \tag{9}$$

3.3. Global graph aggregation

In order to predict lane graphs over larger areas than can be predicted directly, we first predict local graphs from crops of the image and then aggregate these into a global graph. In this section we introduce a graph aggregation algorithm that iteratively predicts a Bézier Graph for each crop, post-processes the model output and uses the Hungarian matching algorithm to produce a globally consistent graph. Pseudocode for our method is shown in Algorithm 1.

Algorithm 1 Bézier Graph global aggregation

```
Require: Images = $\{I_0, \dots, I_n\}$
 1: $G_{\text{agg}} \leftarrow (\emptyset, \emptyset)$
 2: for $I_i \in$ Images do
 3:     $G_{\text{local}} \leftarrow$ predict$(I_i)$
 4:     post_process$(G_{\text{local}})$
 5:     agg_nodes $\leftarrow$ nodes_on_boundary$(G_{agg})$
 6:     local_nodes $\leftarrow$ nodes_on_boundary$(G_{local})$
 7:     matches $\leftarrow$ match_and_threshold(agg_nodes, local_nodes)
 8:     $G_{\text{agg}} \leftarrow$ aggregate$(G_{\text{agg}}, G_{\text{local}},$ matches$)$
 9: end for
10: return $G_{\text{agg}}$
```

Given a large input image, we subdivide the image into tiles of size 512×512 pixels which mutually overlap by 14 pixels. We initialize empty global Bézier Graph and iteratively predict a Bézier Graph for each tile, apply post-processing and aggregate it into the global graph. This comparatively simple "tiling" method as compared to LaneGNN [3] is enabled by our model predicting *all* lanes in a crop, rather than only a successor lane graph.

Post-processing comprises two steps that prune the predicted Bézier Graph for an individual tile. The first step removes singleton nodes: those with no edges, which are meaningless in the context of a lane graph. The second step is to remove edges that form triangles of the following form: let i, j and k be predicted nodes such that there exist predicted edges ij and jk. If there is also an edge ik then we delete this edge from the predicted graph. This is because these edges tend to 'cut the corner' of the true lane predicted by the model, and are likely to be mistakes.

After post-processing the predicted graph for the current tile we aggregate this into the current global graph. We gather all nodes from the global and tile graphs within a region either side of the tile boundary. We then apply the Hungarian algorithm to find a matching between the tile and aggregated sets of nodes according to the following cost function:

$$c(i,j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 + \kappa \mathcal{I}_{[\boldsymbol{d}_i \cdot \boldsymbol{d}_j \le 0]}. \tag{10}$$

The first term computes the Euclidean distance between node i and node j. The second term adds a fixed cost κ if the direction vectors are misaligned. Given the result of the matching we merge any pairs of matched nodes with a cost below a fixed threshold κ_c. Nodes are merged by taking the average of the predicted position and direction vectors. This represents another significant advantage of the Bézier Graph formulation: a principled method of averaging over lane *direction* at aggregation, enforcing direction continuity. All remaining nodes and edges in the predicted tile graph are added into the global graph. Figs. 5a and 5b show the output of our method before and after merging.

4. Experimental Results

4.1. Lane Prediction Tasks

We evaluate on the two lane prediction tasks introduced alongside the UrbanLaneGraph dataset [3]:

- **Successor Lane Graph Prediction** (Succ-LGP): Given an image crop and a starting location, predict the lane graph which is reachable from that location.
- **Full Lane Graph Prediction** (Full-LGP): Given a larger image, predict the full lane graph within that image.

We approach the Full-LGP task with minimal additional assumptions; LaneGNN from the original UrbanLaneGraph paper [3] assumes that there exists a lane starting in the bottom center of input image crops, and therefore requires initial locations obtained from a separately trained model. By contrast, as detailed in Sec. 3.3, we employ a strategy of tiling the entire image with crops and aggregating to form the full lane graph, requiring no prior starting positions.

However, this puts us at a *disadvantage* at the more constrained task of Succ-LGP. For this task, all evaluation examples have a lane beginning in the bottom center of the image which is implicit in the training of LaneGNN but which our standard training method does not assume.

Therefore, we train two variants of our model to target the two tasks and ensure we are not at a disadvantage for Succ-LGP. The first, Full-LGP, is trained on 512×512 crops of the training set, where all ground truth lanes within those crops are included. The second, Succ-LGP, is trained on 256×256 crops where there always exists a lane beginning in the bottom center of the image, and the only lanes included are those which are reachable from that point.

We evaluate both on the `eval` files provided by UrbanLaneGraph. Succ-LGP is evaluated on held-out 256×256 tiles, Full-LGP on 5000×5000 images.

4.2. Bézier Graph Fit

In Tab. 1 we provide quantitative results for the closeness of the Bézier Graph fitting procedure evaluated on the training tiles for each of the tasks introduced in section Sec. 4.1.

The small Hausdorff distances indicate that the Bézier Graphs are able to effectively represent the underlying lane topology across all cities in the dataset. We show also that Bézier Graphs result in $\sim 90\%$ reductions in graph size, demonstrating that this representation is highly efficient.

4.3. Evaluation Metrics

We follow the evaluation procedure of [3] and use the following metrics to evaluate predicted graphs:

- **GEO/TOPO** [9]: the GEO metric measures geometric and topological similarity between graphs by matching nodes within a radius of 8 pixels. The TOPO metric computes the GEO metric over the ego subgraph around each node so that graph connectivity is taken into account.

	City	Hausdorff (pixels)	$\|\mathcal{V}\|$ (% reduction) Max	Mean
Successor	Austin	1.1 ± 1.0	12 (83)	4 ± 2 (84 ± 6)
	Detroit	1.2 ± 1.1	17 (82)	4 ± 2 (84 ± 6)
	Miami	1.2 ± 1.3	16 (83)	4 ± 2 (84 ± 7)
	Palo Alto	1.3 ± 1.4	21 (72)	5 ± 2 (84 ± 6)
	Pittsburgh	1.2 ± 1.4	11 (84)	4 ± 2 (84 ± 7)
	Washington	1.2 ± 1.3	14 (82)	4 ± 2 (84 ± 6)
Full	Austin	3.3 ± 3.4	60 (91)	14 ± 8 (92 ± 4)
	Detroit	3.4 ± 3.5	60 (91)	17 ± 10 (92 ± 5)
	Miami	3.2 ± 3.2	64 (90)	15 ± 9 (92 ± 5)
	Palo Alto	4.0 ± 5.5	77 (92)	16 ± 10 (93 ± 4)
	Pittsburgh	3.7 ± 5.1	49 (91)	13 ± 8 (92 ± 5)
	Washington	3.1 ± 3.3	52 (92)	15 ± 9 (93 ± 5)

Table 1. Metrics on the Bézier Graph fit for each training tile. Succ-LGP uses 256x256 tiles, Full-LGP uses 512x512. Hausdorff (pixels) refers to the *mean* value (averaged over all training tiles for that city and experiment) of the *maximum* Hausdorff distance between the Bézier fit and the ground truth. $|\mathcal{V}|$ denotes number of nodes, percentage reduction compared to ground truth in brackets.

- **APLS** [23]: Average Path Length Similarity is the mean difference in shortest path lengths between nodes in the two graphs scaled from 0 to 1.
- **SDA** [3]: Split Detection Accuracy measures the predictive accuracy on lane splits in the graph.
- **Graph IOU** [30]: computes the intersection over union between the two graphs in pixel space assuming a lane width of 10 pixels.

4.4. Successor Lane Graph Prediction

In this section, we evaluate our model's performance on the Succ-LGP task, using a node threshold of 0.5 and an edge threshold of 0.3. Quantitive results are shown in Tab. 2, and qualitative results are shown in Fig. 4.

Our model improves on many of the metrics, with particular improvements in APLS and SDA, but achieves worse recall performance. We believe these results can be explained by looking at the trends apparent in Fig. 4: our model is less likely to include incorrect lane splits, which appears to be an issue with the LaneGNN model. This is reflected in improved SDA results. However, our model frequently omits entire lanes from its prediction, reducing recall values. This can be explained somewhat by the model architecture; our predictions tend to consist of relatively few nodes and when a single node is omitted, an entire lane will be omitted. LaneGNN, on the other hand, follows a "subtractive" approach, initially sampling nodes and edges to fill a segmentation of reachable lanes and then using a GNN to remove edges from this graph. This results in significantly fewer nodes being omitted, but likely contributes to the incorrect lane split issue.

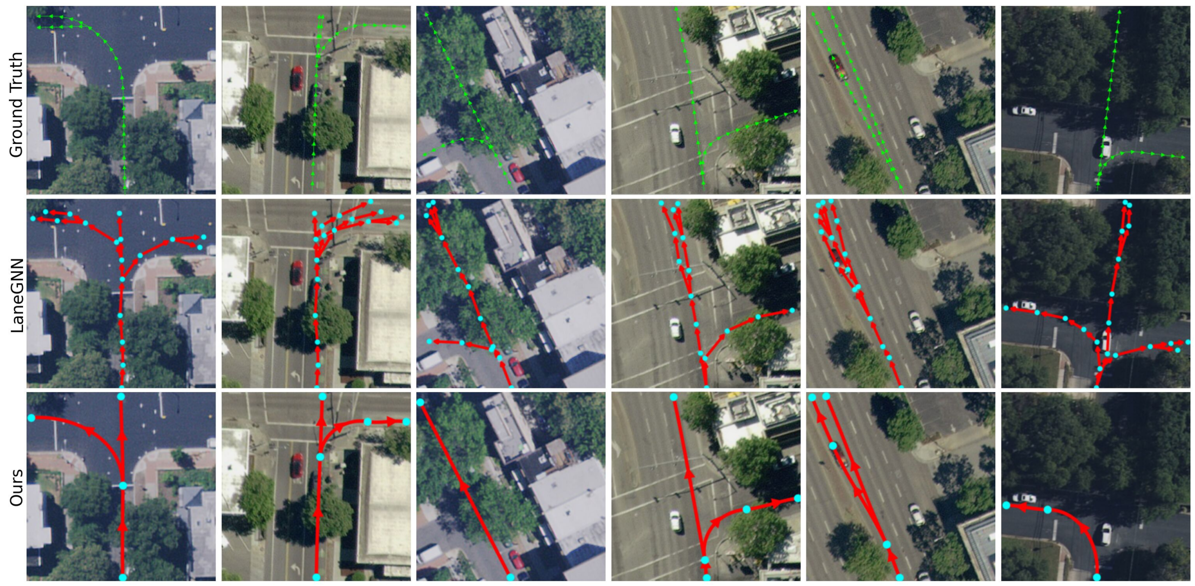


Figure 4. Qualitative comparison of the Succ-LGP variant of our model with LaneGNN [3]. Top row shows the ground truth data, second row shows LaneGNN (nodes in cyan, edges in red), bottom row shows our method (Bézier endpoints in cyan, curves in red).

Model	SDA$_{20}$ ↑	SDA$_{50}$ ↑	Graph IoU ↑	APLS ↑	TOPO P/R/F$_1$ ↑	GEO P/R/F$_1$ ↑
LaneGraphNet [28]	0.0	0.0	0.063	0.179	0.0 / 0.0 / 0.0	0.0 / 0.0 / 0.0
LaneGNN [3]	0.227	0.377	**0.347**	0.202	0.600 / **0.699** / **0.646**	0.599 / **0.695** / 0.643
BGFormer (ours)	**0.492**	**0.549**	0.312	**0.772**	**0.603** / 0.501 / 0.547	**0.762** / 0.576 / **0.656**

Table 2. Succ-LGP results. See Sec. 4.3 for a description of the metrics used. P/R/F$_1$ denotes precision/recall/F$_1$ score.

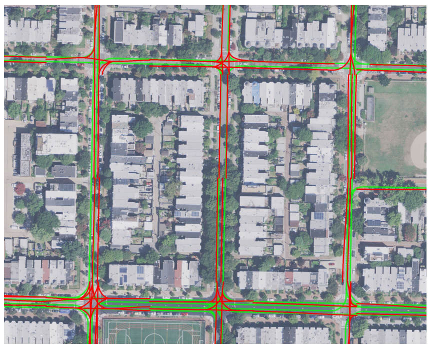
4.5. Full Lane Graph Prediction

For the Full-LGP task, evaluation is conducted on the 11 5000×5000 `eval` aerial images provided by the Urban-LaneGraph dataset, covering 6 different cities and a wide variety of interconnected road and lane geometries; the resulting metrics are presented in Tab. 3. A qualitative comparison of our method and LaneGNN is visualised in Fig. 5. To ensure a fair inference time comparison, all images and models were evaluated on a machine with an AMD EPYC 7282 16-Core Processor, NVIDIA A6000 GPU and 64GB of RAM.

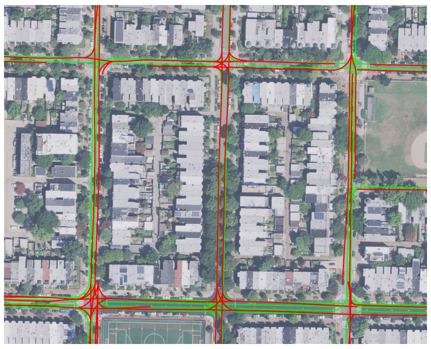
In addition to the processing described in Sec. 3.3, we experiment with one other post-processing variation. We observed that our model occasionally predicted tiles entirely empty of lanes, damaging lane connectivity and recall metrics. We therefore employ a simple method to combat this, applying the model to several different rotations of each tile and selecting an optimal prediction from the set of rotated predictions. In our experiments, we use integer multiples of $90°$ rotations, and select the prediction with $|\mathcal{E}|$ closest to the mean $|\mathcal{E}|$ across these four rotations. We report results with and without this step in Tab. 3, showing that its inclusion consistently improves both precision and recall scores.

Curiously, Tab. 3 shows that the trends observed in Succ-LGP are reversed in the Full-LGP experiments; BGFormer scores consistently higher in recall metrics, and lower in precision metrics, when compared to LaneGNN. This is likely due to the differences in tiling and aggregation; LaneGNN employs a "driving" method, meaning that a single missed node can result in an entire downstream lane being missed, discouraging high recall scores. BGFormer achieves significantly higher APLS scores, likely due to its more robust aggregation strategy discouraging disconnected graphs, as evidenced in Fig. 5.

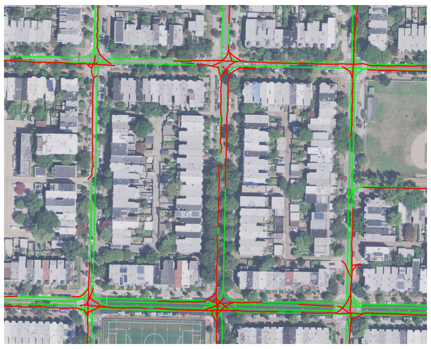
The visual comparison of Fig. 5 reveals further distinctions between the approaches. The lanes of BGFormer are smoother and appear closer to a physically drivable network, particularly when predicting curved paths. This capability leads to a graph structure that closely mirrors actual lane layouts, enhancing the reliability of the model for navigation purposes. To further test this "drivability" improvement we define a simple additional metric: the KL divergence between the predicted and ground truth curvature distributions across the aggregated graphs. We plot this metric for each of the 11 evaluation images in Fig. 6. We observe that BGFormer achieves a lower KL divergence for every one of these evaluation graphs, with an average of



(a) BGFormer before merge



(b) BGFormer after merge



(c) LaneGNN [3]

Figure 5. Qualitiative comparison of example Full-LGP results. Our output - using the BGFormer model with the greatest F_1 score from Tab. 3 - is shown in Fig. 5b, LaneGNN in Fig. 5c. Predicted lanes are depicted in red, ground truth lanes in green. For clarity we show a crop of the image, the corresponding full 5000×5000 prediction can be found in the supplementary material.

Model	R / T	$SDA_{20} \uparrow$	$SDA_{50} \uparrow$	IoU $\uparrow$	APLS $\uparrow$	TOPO P/R/F_1 $\uparrow$	GEO P/R/F_1 $\uparrow$	Time (s) $\downarrow$
LaneGNN [3]	- / -	0.047	0.108	0.112	0.065	**0.554**/0.303/0.392	**0.688**/0.382/0.491	9783.83
BGFormer (ours)	✗ / H	0.054	0.112	0.125	0.187	0.446/0.441/0.443	0.532/0.521/0.526	108.15
	✗ / L	0.063	**0.126**	0.128	0.212	0.427/0.446/0.436	0.520/0.536/0.528	**101.42**
	✓ / H	0.052	0.106	0.132	0.208	0.451/0.461/**0.456**	0.536/0.543/**0.539**	401.87
	✓ / L	**0.064**	0.123	**0.137**	**0.229**	0.431/**0.464**/0.447	0.523/**0.554**/0.538	416.49

Table 3. Full Lane Graph Prediction results. Our BGFormer model is evaluated with different settings, shown in the "R / T" column. R values denote whether rotation is applied (✓) or not (✗) to each tile during processing - see the description in Sec. 4.5 for more details. T values denotes the node and edge detection threshold used: H (high) for 0.6/0.5 respectively, L (low) for 0.5/0.3 respectively.

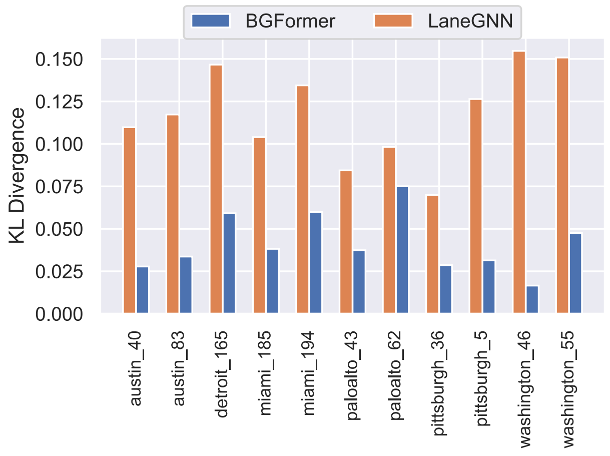


Figure 6. KL divergence between the predicted and ground truth curvature distributions across the aggregated evaluation graphs. LaneGNN in orange, BGFormer in blue.

0.04 ± 0.02 as compared to 0.12 ± 0.03 for LaneGNN, indicating that the predicted distribution of curvatures is consistently closer to the ground truth distribution. Several individual distributions are shown in the supplementary material.

Finally, we highlight our relatively simple end-to-end model architecture. By comparison, LaneGNN requires four independently trained models: a separate pre-trained LaneExtraction [9] model to provide starting positions and directions on the 5000×5000 tiles, two image models to segment all lanes and ego reachable lanes respectively, and a GNN model to filter down the sampled graph. We highlight also that the Bézier formulation naturally results in a smooth lane graph, avoiding the spatial smoothing used by other models. The comparitive simplicity of our method and aggregation scheme contributes to the $>95\%$ reduction in Full-LGP inference times shown in Tab. 3.

5. Conclusion

In this paper, we presented a novel method for representing lane networks in a Bézier Graph, and an end-to-end transformer-based method for generating lane Bézier Graphs from aerial images. We demonstrated that our model produced physically realistic, drivable lanes, reducing the need for downstream postprocessing.

Future work could further adapt the prediction heads; one specific area of improvement would be to modify the architecture to eliminate the 'corner cutting' edges (Sec. 3.3), removing the need for a post-processing step. A further direction for future work would be combining the present techniques with onboard sensors, to achieve the best of the Bézier Graph drivable priors with real-time detection.

References

[1] Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden, and David DeWitt. Roadtracer: Automatic extraction of road networks from aerial images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4720–4728, 2018. 2

[2] Davide Belli and Thomas Kipf. Image-conditioned graph generation for road network extraction. *arXiv preprint arXiv:1910.14388*, 2019. 2

[3] Martin Büchner, Jannik Zürn, Ion-George Todoran, Abhinav Valada, and Wolfram Burgard. Learning and aggregating lane graphs for urban automated driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13415–13424, 2023. 1, 2, 5, 6, 7, 8

[4] Yigit Baran Can, Alexander Liniger, Danda Pani Paudel, and Luc Van Gool. Structured bird's-eye-view traffic scene understanding from onboard images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15661–15670, 2021. 1

[5] Yigit Baran Can, Alexander Liniger, Danda Pani Paudel, and Luc Van Gool. Topology preserving local road network estimation from single onboard camera image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17263–17272, 2022. 1

[6] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 2, 5

[7] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8748–8757, 2019. 3

[8] Zhengyang Feng, Shaohua Guo, Xin Tan, Ke Xu, Min Wang, and Lizhuang Ma. Rethinking efficient lane detection via curve modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17062–17070, 2022. 2

[9] Songtao He and Hari Balakrishnan. Lane-level street map extraction from aerial imagery. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2080–2089, 2022. 1, 2, 6, 8

[10] Namdar Homayounfar, Wei-Chiu Ma, Justin Liang, Xinyu Wu, Jack Fan, and Raquel Urtasun. Dagmapper: Learning to map by discovering lane topology. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2911–2920, 2019. 1

[11] Qi Li, Yue Wang, Yilun Wang, and Hang Zhao. Hdmapnet: An online hd map construction and evaluation framework. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4628–4634. IEEE, 2022. 2

[12] Zuoyue Li, Jan Dirk Wegner, and Aurélien Lucchi. Topological map extraction from overhead images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1715–1724, 2019. 2

[13] Bencheng Liao, Shaoyu Chen, Bo Jiang, Tianheng Cheng, Qian Zhang, Wenyu Liu, Chang Huang, and Xinggang Wang. Lane graph as path: Continuity-preserving path-wise modeling for online lane graph construction. *arXiv preprint arXiv:2303.08815*, 2023. 2

[14] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 5

[15] Yuliang Liu, Chunhua Shen, Lianwen Jin, Tong He, Peng Chen, Chongyu Liu, and Hao Chen. Abcnet v2: Adaptive bezier-curve network for real-time end-to-end text spotting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):8048–8064, 2021. 3

[16] Yicheng Liu, Tianyuan Yuan, Yue Wang, Yilun Wang, and Hang Zhao. Vectormapnet: End-to-end vectorized hd map learning. In *International Conference on Machine Learning*, pages 22352–22369. PMLR, 2023. 2

[17] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual relationship detection with language priors. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 852–869. Springer, 2016. 2

[18] Lu Mi, Hang Zhao, Charlie Nash, Xiaohan Jin, Jiyang Gao, Chen Sun, Cordelia Schmid, Nir Shavit, Yuning Chai, and Dragomir Anguelov. Hdmapgen: A hierarchical graph generative model of high definition maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4227–4236, 2021. 1

[19] Arttu Miettinen, Antonio G Zippo, Alessandra Patera, Anne Bonnin, Sarah H Shahmoradian, Gabriele EM Biella, and Marco Stampanoni. Micrometer-resolution reconstruction and analysis of whole mouse brain vasculature by synchrotron-based phase-contrast tomographic microscopy. *bioRxiv*, pages 2021–03, 2021. 2

[20] Limeng Qiao, Wenjie Ding, Xi Qiu, and Chi Zhang. End-to-end vectorized hd-map construction with piecewise bezier curve. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13218–13228, 2023. 2

[21] Suprosanna Shit, Rajat Koner, Bastian Wittmann, Johannes Paetzold, Ivan Ezhov, Hongwei Li, Jiazhen Pan, Sahand Sharifzadeh, Georgios Kaissis, Volker Tresp, et al. Relationformer: A unified framework for image-to-graph generation. In *European Conference on Computer Vision*, pages 422–439. Springer, 2022. 2, 3

[22] Mihail Ivilinov Todorov, Johannes Christian Paetzold, Oliver Schoppe, Giles Tetteh, Suprosanna Shit, Velizar Efremov, Katalin Todorov-Völgyi, Marco Düring, Martin Dichgans, Marie Piraud, et al. Machine learning analysis of whole mouse brain vasculature. *Nature methods*, 17(4):442–449, 2020. 2

[23] Adam Van Etten, Dave Lindenbaum, and Todd M Bacastow. Spacenet: A remote sensing dataset and challenge series. *arXiv preprint arXiv:1807.01232*, 2018. 6

[24] Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5410–5419, 2017. 2

[25] Zhenhua Xu, Yuxuan Liu, Lu Gan, Yuxiang Sun, Xinyu Wu, Ming Liu, and Lujia Wang. Rngdet: Road network graph detection by transformer in aerial images. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–12, 2022. 2

[26] Zhenhua Xu, Yuxuan Liu, Yuxiang Sun, Ming Liu, and Lujia Wang. Centerlinedet: Centerline graph detection for road lanes with vehicle-mounted sensors by transformer for hd map generation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3553–3559. IEEE, 2023. 2

[27] Zhenhua Xu, Yuxuan Liu, Yuxiang Sun, Ming Liu, and Lujia Wang. Rngdet++: Road network graph detection by transformer with instance segmentation and multi-scale features enhancement. *IEEE Robotics and Automation Letters*, 2023. 2

[28] Yiyang Zhou, Yuichi Takeda, Masayoshi Tomizuka, and Wei Zhan. Automatic construction of lane-level hd maps for urban scenes. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6649–6656. IEEE, 2021. 7

[29] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020. 3, 5

[30] Jannik Zürn, Johan Vertens, and Wolfram Burgard. Lane graph estimation for scene understanding in urban driving. *IEEE Robotics and Automation Letters*, 6(4):8615–8622, 2021. 1, 6

[31] Jannik Zürn, Ingmar Posner, and Wolfram Burgard. Autograph: Predicting lane graphs from traffic observations. *arXiv preprint arXiv:2306.15410*, 2023. 1, 2