VectorMapNet: End-to-end Vectorized HD Map Learning

Yicheng Liu [1 2] Tianyuan Yuan [2] Yue Wang [3] Yilun Wang [4] Hang Zhao [2 1]

Abstract

Autonomous driving systems require High-Definition (HD) semantic maps to navigate around urban roads. Existing solutions approach the semantic mapping problem by offline manual annotation, which suffers from serious scalability issues. Recent learning-based methods produce dense rasterized segmentation predictions to construct maps. However, these predictions do not include instance information of individual map elements and require heuristic post-processing to obtain vectorized maps. To tackle these challenges, we introduce an end-to-end vectorized HD map learning pipeline, termed VectorMapNet. VectorMapNet takes onboard sensor observations and predicts a sparse set of polylines in the bird's-eye view. This pipeline can explicitly model the spatial relation between map elements and generate vectorized maps that are friendly to downstream autonomous driving tasks. Extensive experiments show that VectorMapNet achieve strong map learning performance on both nuScenes and Argoverse2 dataset, surpassing previous state-of-the-art methods by 14.2 mAP and 14.6mAP. Qualitatively, VectorMapNet is capable of generating comprehensive maps and capturing fine-grained details of road geometry. To the best of our knowledge, VectorMapNet is the first work designed towards end-to-end vectorized map learning from onboard observations.

1. Introduction

Autonomous driving systems require an understanding of map elements on the road, including lanes, pedestrian crossing, and traffic signs, to navigate around the world. Such map elements are typically provided by pre-annotated High-Definition (HD) semantic maps in existing pipelines (Rong et al., 2020). However, these methods face scalability issues due to their heavy reliance on human labor for annotating HD maps. Additionally, they necessitate precise localization of the ego-vehicle to derive local maps from the global one, a process that could introduce meter-level errors.

[1]Shanghai Qi Zhi Institute [2]Tsinghua University [3]MIT [4]Li Auto. Correspondence to: Hang Zhao <ZhaoHang0124@gmail.com>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

In contrast, our focus lies in developing a learning-based approach for online HD semantic map learning. The aim is to use onboard sensors, including LiDARs and cameras, to estimate map elements on-the-fly. This methodology avoids the need for localization, allowing for prompt updates. Furthermore, learning-based methods can generate uncertainty or confidence indicators that downstream modules, such as motion forecasting and planning, can utilize to offset imperfect perception. These methods can leverage increasing data and model size, promptly reflect current conditions, and generalize from annotated maps to under-annotated or even non-annotated areas (please refer to Figure 6).

Most of HD semantic map learning methods (Li et al., 2021; Philion & Fidler, 2020; Roddick & Cipolla, 2020; Zhou & Krähenbühl, 2022) consider the task as a semantic segmentation problem in bird's-eye view (BEV), which rasterizes map elements into pixels and assigns each pixel with a class label. This formulation makes it straightforward to leverage fully convolutional networks. However, rasterized maps are not an ideal map representation for autonomous driving, for three reasons. First, rasterized maps lack instance information necessary to distinguish map elements with the same class label but different semantics, *e.g.* left boundary and right boundary. Second, it is hard to enforce spatial consistency within the predicted rasterized maps, *e.g.* nearby pixels might have contradicted semantics or geometries. Third, 2D rasterized maps are incompatible with most autonomous driving systems which consume instance-level 2D/3D vectorized maps for motion forecasting and planning.

To alleviate these issues and produce vectorized outputs, HDMapNet (Li et al., 2021) generates semantic, instance, and directional maps and vectorizes these three maps with a hand-designed post-processing algorithm. However, HDMapNet still relies on the rasterized map predictions, and its heuristic post-processing step restricts the model's scalability and performance.

In this paper, we propose an end-to-end vectorized HD

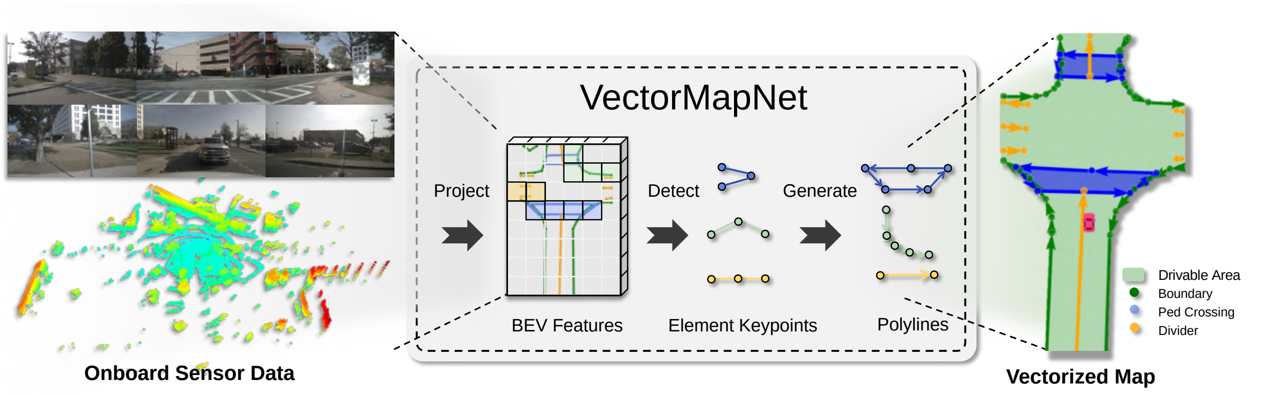

Figure 1: An overview of VectorMapNet. Sensor data is encoded to BEV features in the same coordinate as map elements. VectorMapNet detects the locations of map elements from BEV features by leveraging element queries. The vectorized HD map is built upon a sparse set of polylines that are generated from the detection results. Since our polylines are directional, we can infer drivable and walkable area of a map.

map learning model named VectorMapNet, an end-to-end framework that does not involve dense semantic pixels or sophisticated post-processing steps. Instead, it represents map elements as a set of polylines closely related to downstream tasks, *e.g*. motion forecasting (Gao et al., 2020). Therefore, the mapping problem boils down to predicting a sparse set of polylines from sensor observations. Specifically, we pose it as a detection problem and leverage recent set detection and sequence generation methods. First, VectorMapNet aggregates features generated from different modalities (*e.g*. camera images and LiDAR) into a common BEV feature space. Then, it detects map element locations based on learnable element queries and BEV features. Finally, we decode each element query into a polyline. An overview of VectorMapNet is shown in Figure 1.

Our experiments show that VectorMapNet achieves state-of-the-art performance on the public nuScenes dataset (Caesar et al., 2020) and Argoverse2 (Wilson et al., 2021), outperforming HDMapNet and another baseline by at least 14.2 mAP. Qualitatively, VectorMapNet builds a more comprehensive map than previous works and can capture fine details, *e.g*. jagged boundaries. Furthermore, we feed our predicted vectorized HD map into a downstream motion forecasting module, demonstrating the predicted map's compatibility and effectiveness. To summarize, the contributions of the paper are as follows:

- We present VectorMapNet, an end-to-end mapping approach that eliminates the need for map rasterization and post-processing by predicting vectorized outputs directly from sensor observations.
- We utilize polyline, a flexible primitive with variable lengths and encoded order, to accommodate the heterogeneous nature of map elements. This approach effectively formulates the construction of a polyline map as a detection issue, thereby introducing a new strategy to the mapping paradigm.
- We adapt detection transformer (DETR) models to locate deformable elements within a 3D space. Recognizing that prevalent centerpoint-based feature extraction methods fall short when dealing with map elements of varying sizes and shapes, we propose an innovative solution. Our novel method overcomes these limitations, delivering state-of-the-art performance in online semantic HD map learning tasks.

2. Related Works

Semantic map learning. Annotating semantic maps attracts plenty of interests thanks to autonomous driving. Recently, semantic map learning is formulated as a semantic segmentation problem (Mattyus et al., 2015) and is solved by using aerial images (Máttyus et al., 2016), LiDAR points (Yang et al., 2018), and HD panorama (Wang et al., 2016). The crowdsourcing tags (Wang et al., 2015) are used to improve the performance of fine-grained segmentation. Instead of using offline data, recent works focus on understanding BEV semantics from onboard camera images (Lu et al., 2019; Yang et al., 2021), and videos (Can et al., 2020). Only using onboard sensors as model input is particularly challenging as the inputs and target map lie in different coordinate systems. Recently, several cross-view learning approaches (Philion & Fidler, 2020; Pan et al., 2020; Li et al., 2021; Zhou & Krähenbühl, 2022; Wang et al., 2022; Chen et al., 2022) leverage the geometric structure of scenes to mitigate the mismatch between sensor inputs and BEV representations. Some methods (Casas et al., 2021; Sadat et al., 2020) use pixel-level semantic maps to solve downstream tasks, but the entire downstream pipeline needs to be redesigned to accommodate these rasterized map inputs. Beyond pixel-level semantic maps, our work extracts a consistent vectorized map around ego-vehicle from surrounding cameras or LiDARs, which suits for existing downstream

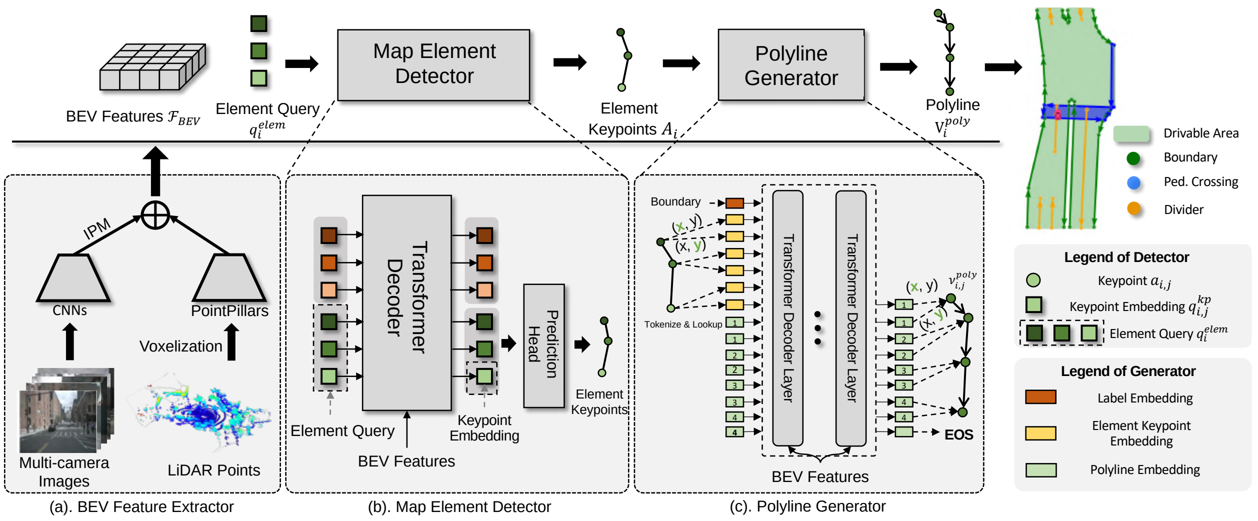

Figure 2: The network architecture of VectorMapNet. The top row is the pipeline of VectorMapNet generating polylines from raw sensor inputs. The bottom row illustrates detailed structures and inference procedures of three primary components of VectorMapNet: BEV feature extractor, map element detector, and polyline generator. Numbers in polyline embeddings indicate predicted vertex indexes.

tasks like motion forecasting (Gao et al., 2020; Zhao et al., 2020; Liu et al., 2021) without further post-processing.

Lane detection. Lane detection aims to separate lane segments from road scenes precisely. Most lane detection algorithms (Pan et al., 2018; Neven et al., 2018) use a pixel-level segmentation technique combined with sophisticated post-processing. Another line of work leverages the predefined proposal to achieve high accuracy and fast inference speed. These methods typically involve handcrafted elements such as vanishing points (Lee et al., 2017), polynomial curves (Van Gansbeke et al., 2019), line segments (Li et al., 2019), and Bézier curves (Feng et al., 2022) to model proposals. In addition to using perspective view cameras as inputs, (Homayounfar et al., 2018) and (Liang et al., 2019) extract lane segments from overhead highway cameras and LiDAR imagery with a recurrent neural network. Instead of discovering the road's topology via boundaries detection, STSU (Can et al., 2021) and LaneGraphNet (Zürn et al., 2021) construct lane graphs from centerline segments that are encoded by Bézier curves and line segments, respectively. To model complex geometries in the urban environment, we leverage polylines to represent all the map elements in perceptual scopes.

Geometric data modeling. Another line of work closely related to VectorMapNet is geometric data generation. These methods typically treat geometric elements as a sequence, such as primitive parts of furniture (Li et al., 2017; Mo et al., 2019), states of sketch strokes (Ha & Eck, 2017), vertices of n-gon mesh (Nash et al., 2020), and parameters of SVG primitives (Carlier et al., 2020). These methods generate these sequences by leveraging autoregressive models (*e.g.* Transformer). Since the directly modeling sequence is challenging for long-range centerline maps, HDMapGen (Mi et al., 2021) views the map as a two-level hierarchy. It produces a global and local graph separately with a hierarchical graph RNN. Instead of treating geometric elements as a sequence generation problem, LETR (Xu et al., 2021) models line segment as a detection problem and tackle it with a query-based detector. Unlike the above approaches that focus on single-level geometric modelings, such as scene level (*e.g.* line segments in an image) or object-level (*e.g.* furniture), VectorMapNet is designed to address both the scene level and object level geometric modeling. Specifically, VectorMapNet constructs a map by modeling the global relationship between map elements in the scene and the local geometric details inside each element.

Learning vector representations from images. VectorMapNet bears some similarities with predicting vector graphics from raster images. Several recent works (Carlier et al., 2020; Reddy et al., 2021) use different vector representations to generate vector images. (Ganin et al., 2021) converts images to CAD, CanvasVAE (Yamaguchi, 2021) learns vectorized canvas layouts from images, and (Liu et al., 2022) generates vectorized stroke primitives from a raster line drawing. The instance segmentation community has also been concerned with a similar task of detecting object contours in a vector form from an image. These methods (Acuna et al., 2018; Liang et al., 2020; Castrejon et al., 2017; Zorzi et al., 2022; Zhang & Wang, 2019) initialize a contour for every object instance and then refine the vertex positions of the contour. However, The above methods are highly domain-dependent, and it is non-trivial to adapt them for our task that requires detecting and generating map elements with different semantics and geometry in the 3D world.

3. VectorMapNet

Problem Formulation and Challenges. Similar to HDMapNet (Li et al., 2021), our task is to vectorize map elements using data from onboard sensors of autonomous vehicle, such as RGB cameras and/or LiDARs. These map elements include but are not limited to: *Road boundaries* (boundaries of roads separating roads and sidewalks, typically irregularly-shaped curves of arbitrary lengths), *Lane dividers* (boundaries dividing lanes on the road, usually straight lines), and *Pedestrian crossings* (regions with white markings indicating legal pedestrian crossing points, typically represented as polygons). While the task is clearly defined, it is fraught with complexities and unique challenges when tackling it. (1) The diverse geometric structures of map elements make it difficult to establish a unified geometric representation. (2) The inputs and outputs of the mapping problem are not perfectly aligned. They exist in different view spaces (*e.g.* camera data is in perspective view and map elements are in BEV), and not all map elements are fully visible from input sensors. In some extreme cases, map elements may be completely occluded by vehicles. (3) The task requires more than simple vectorization; it also necessitates scene understanding because of the complex geometrical and topological relationships between map elements. For instance, map elements may overlap, or two traffic cones connected with a wire might indicate a road boundary.

3.1. Method Overview

The challenges above underline the need for a primitive that effectively represents a variety of geometric structures and a model that is capable of capturing the geometrical and topological relationships from various sensor inputs.

Polyline representation. The heterogeneous geometry of map elements calls for a unified vectorized representation. We opt to use N polylines $\boldsymbol{\mathcal{V}}^{\text{poly}} = \{\boldsymbol{V}_1^{\text{poly}}, \dots, \boldsymbol{V}_N^{\text{poly}}\}$ as primitives to represent these map elements in a map $\mathcal{M}$. Each polyline $\boldsymbol{V}_i^{\text{poly}} = \{\boldsymbol{v}_{i,n} \in \mathbb{R}^2 | n = 1, \dots, N_v\}$ is a collection of N_v ordered vertices $\boldsymbol{v}_{i,n}$. In practice, we preprocess public autonomous driving semantic maps to obtain a unified polyline representation of map elements: polygons are represented as closed polylines; curves are converted into polylines by applying the Ramer–Douglas–Peucker algorithm (Ramer, 1972).

Using polylines to represent map elements has three main advantages: (1) HD maps are typically composed of a mixture of different geometries, such as points, lines, curves, and polygons. Polylines are a flexible primitive that can represent these geometric elements effectively. (2) The order of polyline vertices is a natural way to encode the direction of map elements, which is vital to driving. (3) The polyline representation has been widely used by downstream autonomous driving modules, such as motion forecasting (Gao et al., 2020).

VectorMapNet. We introduce VectorMapNet, an end-to-end model designed to represent a map $\mathcal{M}$ with a sparse set of polylines $\boldsymbol{\mathcal{V}}^{\text{poly}}$, thus formulating the task as a sparse set detection problem. In our approach, we convert sensor data into a canonical Bird's Eye View (BEV) representation, $\boldsymbol{\mathcal{F}}_{\text{BEV}}$, and model polylines based on this BEV. Given the complexity and diversity of map elements' structural and location patterns and relationships, we divide the task into three distinct components: (1) A *BEV feature extractor* (§ 3.2) that lifts various sensor modality inputs into a canonical feature space. (2) A *map element detector* (§ 3.3) that locates and classifies all map elements by predicting element keypoints $\boldsymbol{\mathcal{A}} = \{\boldsymbol{A}_i \in \mathbb{R}^{k\times 2} | i = 1, \dots, N\}$ and their class labels $\boldsymbol{\mathcal{L}} = \{l_i \in \mathbb{Z} | i = 1, \dots, N\}$. The definition of element keypoint representation $\mathcal{A}$ is described in § 3.3. (3) A *polyline generator* (§ 3.4) that produces a sequence of ordered polyline vertices which describes the local geometry of each detected map element $(\boldsymbol{A}_i, l_i)$. An overview of three components is demonstrated in Figure 2.

3.2. BEV Feature Extractor

The objective of BEV feature extractor is to lift various modality inputs into a canonical feature space and aggregates and align features these features into a canonical representation termed BEV features $\boldsymbol{\mathcal{F}}_{\text{BEV}} \in \mathbb{R}^{W\times H\times(C_1+C_2)}$ based on their coordinates, where W and H represent the width and height of the BEV feature, respectively; C_1 and C_2 represent the output channels of the BEV feature extracted from the two common modalities: surrounding camera images $\mathcal{I}$ and LiDAR points $\mathcal{P}$.

Camera branch. We use ResNet to extract features from images, followed by a feature transformation module from image space to BEV space. VectorMapNet does not rely on certain feature transformation approaches and we opt to use a simple but popular variant of IPM, which produces BEV features of $\boldsymbol{\mathcal{F}}_{\text{BEV}}^{\mathcal{I}} \in \mathbb{R}^{W\times H\times C_1}$. The detailed structure of the image extractor can be found in Appendix C.3.

LiDAR branch. For LiDAR data $\mathcal{P}$, we use a variant of PointPillars (Lang et al., 2019) with dynamic voxelization (Zhou et al., 2020), which divides the 3D space into multiple pillars and uses pillar-wise point clouds to learn pillar-wise feature maps. We denote this feature map in BEV as $\boldsymbol{\mathcal{F}}_{\text{BEV}}^{\mathcal{P}} \in \mathbb{R}^{W\times H\times C_2}$.

For sensor fusion, we obtain the BEV features $\boldsymbol{\mathcal{F}}_{\text{BEV}} \in \mathbb{R}^{W\times H\times(C_1+C_2)}$ by concatenating $\boldsymbol{\mathcal{F}}_{\text{BEV}}^{\mathcal{I}}$ and $\boldsymbol{\mathcal{F}}_{\text{BEV}}^{\mathcal{P}}$, and then process the concatenated result with a two-layer convolutional network. An overview of the BEV feature extractor is shown at the bottom-left of Figure 2.

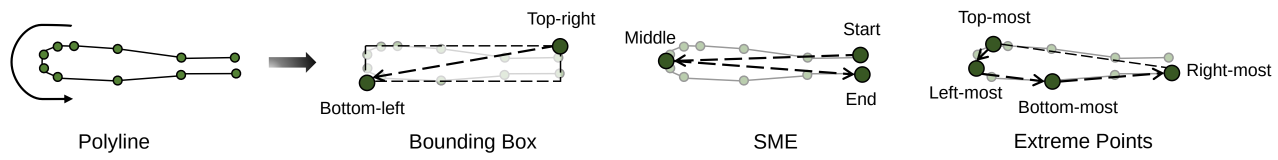

Figure 3: Three different keypoint representations are proposed here: Bounding Box (k=2), SME (k=3), and Extreme Points (k=4), where k has the same definition in § 3: the number of key points of each keypoint representation. The arrow line indicates the direction of the example polyline, and the arrow dash lines indicate the vertices order of keypoint representations.

3.3. Map Element Detector

After extracting the bird's-eye view (BEV) features, VectorMapNet have to identify and abstractly represent map elements using these features. We employ a hierarchical representation for this purpose, specifically through element queries and keypoint queries, enabling us to model the non-local shape of map elements effectively. We leverage a variant of transformer set prediction detector (Carion et al., 2020) to achieve this goal, as it is a robust detector that eliminates the need for extra post-processing. Specifically, the detector represents map elements' locations and categories by predicting their element keypoints $\boldsymbol{\mathcal{A}}$ and class labels $\boldsymbol{\mathcal{L}}$ from the BEV features $\boldsymbol{\mathcal{F}}_{\text{BEV}}$.

Element queries. The detector uses learnable element queries $\boldsymbol{q}_i^{\text{elem}} \in \mathbb{R}^{k\times d} | i = 1, \dots, N_{\max}$ as its inputs, where d represents the hidden embedding size and $N_{\max}$ is a preset constant, which is much greater than the number of map elements N in the scene. The i-th element query $\boldsymbol{q}_i^{\text{elem}}$ is composed of k element keypoint embeddings $\boldsymbol{q}_{i,j}^{\text{kp}}$: $\boldsymbol{q}_i^{\text{elem}} = \{\boldsymbol{q}_{i,j}^{\text{kp}} \in \mathbb{R}^d | j = 1, \dots, k\}$. Element queries are similar to object queries used in Detection Transformer (DETR) (Carion et al., 2020), where a query represents an object. In our case, an element query represents a map element.

Keypoint representations. In object detection problems, people use bounding box to abstract object shape. Here we use k element keypoints locations $\boldsymbol{A}_i = \{\boldsymbol{a}_{i,j} \in \mathbb{R}^2 | j = 1, ..., k\}$ (please refer to Figure 3), to represent the outline of a map element. However, defining keypoints for map elements is not straightforward due to their diversity. We conduct an ablation study to investigate the performance of different choices in § 4.3. Note that element keypoints are different from polyline vertices and the element keypoints are intermediate representations of VectorMapNet that are passed to the polyline generator (§ 3.4) for conditional prediction, and the number of keypoints for each type of polyline is fixed and determined by its definition. Polylines are our output representations.

Architecture. The overall architecture of the map element detector consists of a transformer decoder (Vaswani et al., 2017) and a prediction head, as shown at the bottom-middle of Figure 2. The decoder transforms the element queries using multi-head self-/cross-attention mechanisms. In particular, we use the deformable attention module (Zhu et al., 2020) as the decoder's cross attention module, where each element query has a 2D location grounding. It improves interpretability and accelerates training convergence (Li et al., 2022).

The prediction head has two MLPs, which decodes element queries into element keypoints $\boldsymbol{a}_{i,j} = \text{MLP}_{\text{kp}}(\boldsymbol{q}_{i,j}^{\text{kp}})$ and their class labels $l_i = \text{MLP}_{\text{cls}}([\boldsymbol{q}_{i,1}^{\text{kp}}, \dots, \boldsymbol{q}_{i,k}^{\text{kp}}])$, respectively. $[\cdot]$ is a concatenation operator. Each keypoint embedding $\boldsymbol{q}_{i,j}^{\text{kp}}$ in the map element detector consists of two learnable parts. The first parts is a keypoint position embedding $\{\boldsymbol{e}_j^{\mathbf{kp}} \in \mathbb{R}^d | j = 1, \dots, k\}$, indicating which position in an element keypoint the point belongs to. The second embedding $\{\boldsymbol{e}_i^{\mathbf{P}} \in \mathbb{R}^d | i = 1, \dots, N_{\max}\}$ encodes which map element the keypoint belongs to. The keypoint embedding $\boldsymbol{q}_{i,j}^{\text{kp}}$ is the addition of these two embeddings $\boldsymbol{e}_i^{\mathbf{P}} + \boldsymbol{e}_j^{\mathbf{kp}}$.

3.4. Polyline Generator

Upon the approximate position, shape, and category of map elements identified by map element detector, the polyline generator focuses on the detailed geometry of HD map, which entails calculating variable-length polyline vertices and their order. Accurate modeling of vertex relationships is crucial - for instance, a white line between two vertices often signifies a line connection in the vectorized map. The polyline generator operates as a discrete distribution $p(\boldsymbol{V}_i^{\text{poly}} | \boldsymbol{A}_i, l_i, \boldsymbol{\mathcal{F}}_{\text{BEV}}^f)$ over the vertices of each polyline, conditioned on the initial layout (*i.e.*, element keypoints $\boldsymbol{A}_i$ and class label l_i) and BEV features. To estimate this distribution, we decompose the joint distribution over each polyline $\boldsymbol{V}_i^{\text{poly}}$ as a product of a series of conditional vertex coordinate distributions. In particular, we transform each polyline $\boldsymbol{V}_i^{\text{poly}} = \{\boldsymbol{v}_{i,n} \in \mathbb{R}^2 | n = 1, \dots, N_v\}$ into a flattened sequence $\{v_{i,n}^f \in \mathbb{R} | n = 1, \dots, 2N_v\}$ by concatenating coordinates values of polyline vertices and add an additional *End of Sequence* token (EOS) at the end of each sequence, and the target distribution turns into:

$$p(\boldsymbol{V}_i^{\text{poly}} | \boldsymbol{A}_i, l_i, \boldsymbol{\mathcal{F}}_{\text{BEV}}; \boldsymbol{\theta}) = \prod_{n=1}^{2N_v} p(v_{i,n}^f | v_{i,<n}^f, \boldsymbol{A}_i, l_i, \boldsymbol{\mathcal{F}}_{\text{BEV}}). \quad (1)$$

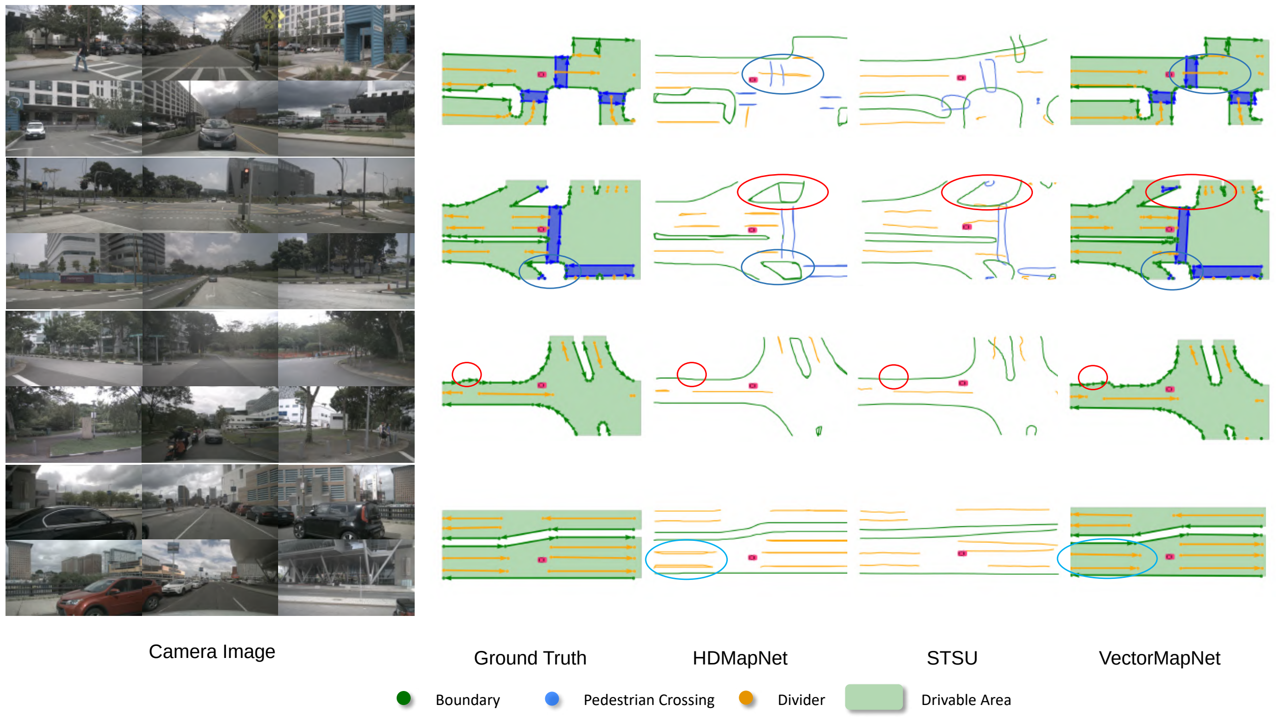

Figure 4: Qualitative results generated by VectorMapNet and baselines. We use camera images as inputs for comparisons. The areas enclosed by **red** and **blue** ellipses show that VectorMapNet can preserve sharp corners, and polyline representations prevent VectorMapNet from generating ambiguous self-looping results. The drivable area is inferred from disjoint boundaries.

Following PolyGen (Nash et al., 2020), we use a categorical distribution to model the probability of each vertex position given the preceding vertex position. This allows us to model the complex and irregular shapes of map elements while maintaining the efficiency of discrete distributions. And we model this distribution using an autoregressive network that outputs the parameters of a predictive distribution at each step for the next vertex coordinate. This predictive distribution is defined over all possible discrete vertex coordinate values and EOS.

Vertices as discrete variables. Using discrete distributions to model polyline vertices has the advantage of representing arbitrary shapes, *i.e.*, categorical distributions can easily represent various polylines, such as multi-modal, skewed, peaked, or long-tailed, that are commonly seen in our task. Thus, we quantize the coordinate values into discrete tokens and model each token with a categorical distribution. We also conduct an ablation study in Appendix § D.2 to investigate other choices.

Architecture. To model these local geometric structures of polylines, the autoregressive network we choose is Transformer (Vaswani et al., 2017) (see the bottom-right of Figure 2). Transformer architecture has consistently demonstrated superior performance in conditional sequence generation tasks and are highly effective at capturing the vertex dependencies present in map data. Each polyline's keypoint coordinates and class label are tokenized and fed in as the query inputs of the transformer decoder. Then a sequence of vertex tokens are fed into the transformer iteratively, integrating BEV features with cross-attention, and decoded as polyline vertices. Note that the generator can generate all polylines in parallel.

Vertex embeddings. Following PolyGen (Nash et al., 2020), we use an addition of three learned embeddings as the embedding of each vertex token: *Coordinate Embedding*, indicating whether the token represents x or y coordinate; *Position Embedding*, representing which vertex the token belongs to; *Value Embedding*, expressing the token's quantized coordinate value.

3.5. Learning

We train our model by minimizing the sum of map element detector loss and polyline generator loss:

$$\mathcal{L} = \mathcal{L}_{det} + \mathcal{L}_{gen}. \tag{2}$$

Map element detector loss. Following (Wang et al., 2022; Zhu et al., 2020), the detector is trained with bipartite matching loss, thus avoiding post-processing steps like non-maximum suppression (NMS). We describe the detail of the map element detector loss $\mathcal{L}_{det}$ function in Appendix § C.4.

Polyline generator loss. Polyline generator is trained to maximize the log-probability of the polyline vertices. We use negative log-likelihood as its loss function:

$$\mathcal{L}_{gen} = -\frac{1}{2N_v} \sum_{n=1}^{2N_v} \log \hat{p}(v^f_{i,n} | v^f_{i,<n}, \boldsymbol{A}_i, l_i, \boldsymbol{\mathcal{F}}^f_{\mathrm{BEV}}), \quad (3)$$

where $\hat{p}(v^f_{i,n}|\dots)$ is the conditional probability of discrete coordinate value $v^f_{i,n}$, and $v^f_{i,<n}$ are ground truth discrete coordinate values with index less than n. The default training strategy is teacher forcing, meaning that we use ground truth keypoints as generator input. To avoid the exposure bias (Bengio et al., 2015), we further experiment with first training with teacher forcing, and then fine-tuning with predicted keypoints.

4. Experiments

Experiments protocol. We conduct experiments on the nuScenes (Caesar et al., 2020) and Argoverse2 (Wilson et al., 2021) dataset. Following HDMapNet (Li et al., 2021), we assess the quality of a predicted HD map by comparing its components (*i.e.*, polylines) with ground truth. Both HDMapNet and our paper use Chamfer distance for polyline matching (Chamfer AP). Additionally, we also introduced another distance metric termed Fréchet distance (Fréchet AP), which better measures the distance between polylines by considering the order of vertices. The definitions and calculation processes of Chamfer AP and Fréchet AP are in § A.2. Additionally, the details of dataset settings (§ A.1), implementations (§ C), and additional qualitative results (§ B) are presented in the Appendix as well.

4.1. Comparison with Baselines

The HD semantic map construction is a new problem, and there are no established methods to compare with. Therefore, we carefully chose two baselines HDMapNet and STSU (Can et al., 2021) that are representative and can effectively compare with VectorMapNet. Specifically: HDMapNet can provide valuable insights into the effectiveness of commonly used map segmentation methods for HD semantic map construction. STSU, a direct map structure learning method, can provide valuable insights into its effectiveness for HD semantic map construction. Moreover, our baseline comparison also includes the results of HDMapNet and VectorMapNet using different modalities as inputs, which demonstrate the impact of different feature extraction methods on HD semantic map construction. The details of baselines model are described in Appendix § C.5. We report the average precision that uses Chamfer distance as the threshold to determine the positive matches with ground truth. $\{0.5, 1.0, 1.5\}$ are the predefined thresholds of Chamfer distance AP.

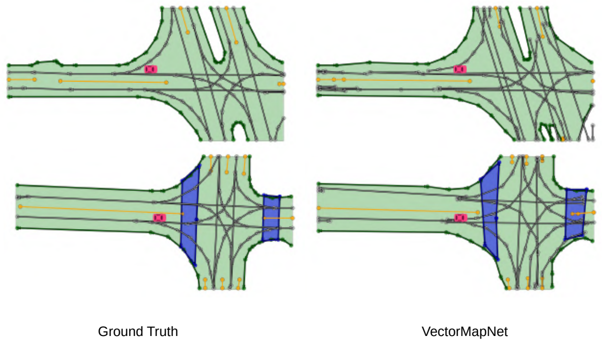

Figure 5: The centerline predictions by VectorMapNet, where the gray lines are the predicted centerlines.

Results on nuScenes. As shown in Table 1, VectorMapNet outperforms HDMapNet by a large margin under all settings (+17.9 mAP in Camera, +9.9 mAP in LiDAR, and +14.2 mAP in Fusion). Compared to camera-only and LiDAR-only, sensor fusion introduces +4.3 mAP improvement and +11.2 mAP improvement, respectively. As described in § 3.5, our two stage training strategy further boosts the performance of both camera-only and sensor fusion methods by +6.9 mAP and +8.5 mAP, respectively. STSU is -29.2 mAP lower than VectorMapNet. Since STSU treats all map elements as a set of fixed-size segments, we hypothesize that ignoring the fine geometry of map elements hurts the performance.

Results on Argoverse2. We further compare HDMapNet and VectorMapNet on Argoverse2 dataset, shown in Table 2. Since Argoverse2 provides z-axis annotations, we give VectorMapNet results both in 2D and 3D. In many cases of Argoverse2, the annotated boundaries and divider lines overlap with each other, making it difficult for models to separate them. It results in a drop in performance of both methods, especially in AP$_{divider}$ of HDMapNet (21.7 AP$_{divider}$ to 5.7 AP$_{divider}$) because its rasterized representation fails to handle these cases. In contrast, VectorMapNet remains competent, showing the advantage of using vectorized representation to represent overlapping elements.

4.2. Qualitative Analysis

Benefits of using polylines as primitives. From visualizations, we find that using polylines as primitives has brought us two benefits compared with baselines: First, polylines effectively encode the detailed geometries of map elements, *e.g.* the corners of boundaries (see the red ellipses in Figure 4). Second, polyline representations prevent VectorMapNet from generating ambiguous results, as it consistently encodes direction information. In contrast, Rasterized methods are prone to falsely generating loopy curves (see the blue ellipses in Figure 4). These ambiguities hinder safe autonomous driving. Therefore, the polyline is a desired primitive for map learning, as it can reflect real-world road

Table 1: Results on nuScenes dataset. Fusion denotes the model using both images and LiDAR points as inputs. Methods with fine-tune means the model is applied two stage training strategy introduced in § 3.5

Methods	AP_{ped}	$AP_{divider}$	$AP_{boundary}$	mAP
STSU (Can et al., 2021)	7.0	11.6	16.5	11.7
HDMapNet (Camera) (Li et al., 2021)	14.4	21.7	33.0	23.0
HDMapNet (LiDAR) (Li et al., 2021)	10.4	24.1	37.9	24.1
HDMapNet (Fusion) (Li et al., 2021)	16.3	29.6	46.7	31.0
VectorMapNet (Camera)	36.1	47.3	39.3	40.9
VectorMapNet (Camera) + fine-tune	42.5	51.4	44.1	46.0
VectorMapNet (LiDAR)	25.7	37.6	38.6	34.0
VectorMapNet (Fusion)	37.6	50.5	47.5	45.2
VectorMapNet (Fusion) + fine-tune	**48.2**	**60.1**	**53.0**	**53.7**

Table 2: Results on Argoverse2 dataset.

		Fréchet Distance				Chamfer Distance			
Keypoint Representaion	$\#dim$	AP_{ped}	$AP_{divider}$	$AP_{boundary}$	mAP	AP_{ped}	$AP_{divider}$	$AP_{boundary}$	mAP
HDMapNet (Camera) (Li et al., 2021)	2	-	-	-	-	13.1	5.7	37.6	18.8
VectorMapNet (Camera)	2	43.2	45.5	52.0	46.9	38.3	36.1	39.2	37.9
VectorMapNet (Camera)	3	41.7	42.3	49.9	44.6	36.5	35.0	36.2	35.8

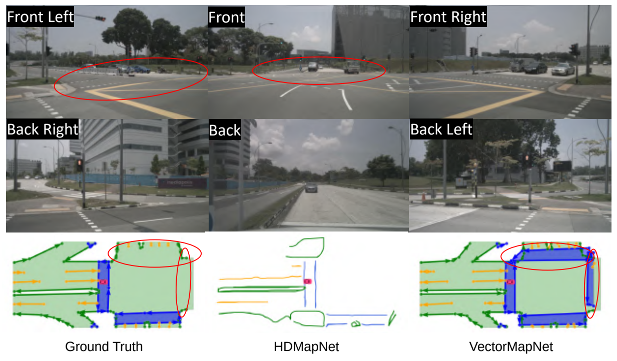

Figure 6: An example of VectorMapNet detecting unlabeled map elements. The **red ellipses** indicate two pedestrian crossings that are missing in ground truth annotations, while VectorMapNet detects it correctly. All the predictions are generated from camera images.

layouts and explicitly encode directions.

Benefits of posing map learning as a detection problem. VectorMapNet operates in a top-down detection manner: it first models the map's topology and the locations of map elements, then generates the details of these elements. Visualizations demonstrate that VectorMapNet captures all map elements comprehensively, even the smaller ones near edges. The high mAP of VectorMapNet, when compared to other baselines, validates this observation. We attribute these impressive results to the model's ability to model topological relationships between map elements, thus implicitly capturing complex scene interrelationships. This is evidenced by Figure 6, where the model identifies pedestrian crossings at intersections that are missed in the annotations of the HD map provided by the dataset. Although these relationships are not explicitly taught, the model learns them via controlled information propagation between query embeddings, using self-attention modules — a technique from the original Transformer paper. This showcases the model's proficient scene understanding.

Centerline prediction by VectorMapNet. As discussed in § 3.1 and above, the polyline is a versatile primitive, capable of representing map element classes that extend beyond the elements in the HD semantic map setting. To further demonstrate this flexibility, we expand VectorMapNet to predict the centerline, an imaginary line commonly used as a reference for driving direction, vehicle positioning, and navigation. The adaptation is quite straightforward: VectorMapNet treats centerlines as a set of polylines and implicitly encodes their topological relations. This process involves no modifications to the model structure. Figure 5 displays the results of VectorMapNet's centerline prediction.

4.3. Ablation Studies

We provide ablation studies for keypoint representation in this section. For other ablation studies (*i.e.*, curve sampling strategies, vertex modeling methods, and extrinsic robustness), please refer to Appendix § D.

Keypoint representations. Since there is no straightforward keypoint design to represent map elements with few fixed number of points, we propose three simple representations as shown in Figure 3: *Bounding Box (Bbox)*, which is the smallest box enclosing a polyline, and its keypoints are defined as the top-right and bottom-left points of the box; *Start-Middle-End (SME)*, which samples the start, middle, and end point from a polyline; *Extreme Points*, which are the left-most, right-most, top-most, and bottom-most points of a polyline. We experiment with these representations and list the results in Table 3. Our results show that the bounding box representation leads to the best mean average performance in both metrics, outperforming others by 2.0 Fréchet mAP and 7.3 Chamfer mAP.

Table 3: Ablation study of keypoint representaions. k is the keypoint number of each keypoint representation.

		Fréchet Distance				Chamfer Distance			
Keypoint Representaion	k	AP_{ped}	$\text{AP}_{divider}$	$\text{AP}_{boundary}$	mAP	AP_{ped}	$\text{AP}_{divider}$	$\text{AP}_{boundary}$	mAP
Bbox	2	**47.4**	46.9	**62.8**	**52.4**	**36.1**	**47.3**	**39.3**	**40.9**
SME	3	47.0	**47.4**	56.9	50.4	27.6	34.4	35.4	32.5
Extreme	4	41.7	47.3	59.0	49.4	30.4	33.1	37.3	33.6

Table 4: The benefits of predicted maps in improving the motion forecasting baseline. There are three input settings: past trajectories (denoted as Traj.), past trajectories with the human-annotated HD map from the nuScenes (denoted as Traj. + G.T. Map), and past trajectories with the predicted map from VectorMapNet (denoted as Traj. + Pred. Map). The predicted map greatly improves the prediction performance compared with the model that only use past trajectories.

Model Inputs	minADE ↓	minFDE↓	MR@2m↓
Traj.	0.909	1.577	19.6
Traj. + G.T. Map	0.779	1.390	18.0
Traj. + Pred. Map	0.826	1.477	18.2

4.4. Motion Forecasting with Vectorized HD Maps from VectorMapNet

To evaluate the capacity of our method to understand scene relationships and to investigate its usefulness in subsequent tasks, we put our predicted HD map to the test within a motion forecasting task. This task heavily relies on precise map information for accurate prediction of future motion.

Task Settings. The motion forecasting requires that the model have to predict 6 possible future trajectories (3 seconds) from past agents' trajectories (1 second) and an HD semantic map spanning $60m \times 30m$. Data is generated from the nuScenes tracking dataset, selecting agents with complete 3-second future observations. This results in 25,645 training and 5,460 test samples. We examine three input scenarios: past trajectories alone, past trajectories with the true HD map, and past trajectories with the VectorMapNet predicted map. We utilize mmTransformer (Liu et al., 2021) for motion forecasting due to its versatility in using map data or relying solely on past trajectories. This assists in assessing the quality of our learned maps.

Results. To evaluate the performance of motion forecasting under different input settings, we report results on three commonly used metrics (Chang et al., 2019): minimum average displacement error (minADE), minimum final displacement error (minFDE) and miss rate (MR). To get the results, these metrics only account for the best trajectory out of 6 predicted trajectories. Results in Table 4 show that the map predicted by VectorMapNet has encoded environment information that greatly helps the motion forecaster, compared with the model that only takes past trajectories as inputs. The gap between the ground-truth map and the predicted map is not big either, especially in terms of MR (-0.2%). We think future research could further close the performance gap.

5. Discussions

Limitations. It is worth noting that the model has some limitations, and we leave it for future works. *Lacking Temporal Information*: The model generates coherent geometries in a single frame but doesn't guarantee temporally consistent predictions. *Mismatch Problem of a Two-stage Model*: A feature space mismatch exists between the map element detector and the polyline generator due to the teacher-forcing training strategy. Although fine-tuning is necessary for optimal performance, it results in tricky training schedules. *Hallucination Ability*: The model can make predictions at locations that are occluded and not visible to cameras, showcasing its scene understanding capabilities. However, this reduces the model's interpretability.

For further discussions, such as the potential societal impact of our method, please refer to Appendix § E.

6. Conclusions

We present VectorMapNet, an end-to-end model to tackle the HD semantic map learning problem. Unlike existing works, VectorMapNet uses polylines as the primitives to represent vectorized HD map elements. To predict polylines from sensor data, we decompose the problem into a detection step and a generation step. Our experiments show that VectorMapNet can generate coherent and complex geometries for urban map elements, benefiting from the polyline primitives. We believe that this novel way to learn HD maps provides a new perspective on the HD semantic map learning problem.

Acknowledgements

This work is supported by the National Key R&D Program of China (2022ZD0161700). We would like to thank Qi Li and Tianyuan Zhang for their help on various baselines, and thank Ziyuan Huang and Bowen Li for paper proofreading.

References

Acuna, D., Ling, H., Kar, A., and Fidler, S. Efficient interactive annotation of segmentation datasets with polygon-rnn++. 2018.

Agarwal, P. K., Avraham, R. B., Kaplan, H., and Sharir, M. Computing the discrete fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43(2):429–449, 2014.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28, 2015.

Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11621–11631, 2020.

Can, Y. B., Liniger, A., Unal, O., Paudel, D., and Van Gool, L. Understanding bird's-eye view semantic hd-maps using an onboard monocular camera. *arXiv preprint arXiv:2012.03040*, 2020.

Can, Y. B., Liniger, A., Paudel, D. P., and Van Gool, L. Structured bird's-eye-view traffic scene understanding from onboard images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15661–15670, 2021.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. In *European conference on computer vision*, pp. 213–229. Springer, 2020.

Carlier, A., Danelljan, M., Alahi, A., and Timofte, R. Deepsvg: A hierarchical generative network for vector graphics animation. *Advances in Neural Information Processing Systems*, 33:16351–16361, 2020.

Casas, S., Sadat, A., and Urtasun, R. Mp3: A unified model to map, perceive, predict and plan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14403–14412, 2021.

Castrejon, L., Kundu, K., Urtasun, R., and Fidler, S. Annotating object instances with a polygon-rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5230–5238, 2017.

Chang, M.-F., Lambert, J., Sangkloy, P., Singh, J., Bak, S., Hartnett, A., Wang, D., Carr, P., Lucey, S., Ramanan, D., et al. Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8748–8757, 2019.

Chen, X., Zhang, T., Wang, Y., Wang, Y., and Zhao, H. Futr3d: A unified sensor fusion framework for 3d detection. *arXiv preprint arXiv:2203.10642*, 2022.

Eiter, T. and Mannila, H. Computing discrete fréchet distance. 1994.

Feng, Z., Guo, S., Tan, X., Xu, K., Wang, M., and Ma, L. Rethinking efficient lane detection via curve modeling. *arXiv preprint arXiv:2203.02431*, 2022.

Ganin, Y., Bartunov, S., Li, Y., Keller, E., and Saliceti, S. Computer-aided design as language. *Advances in Neural Information Processing Systems*, 34:5885–5897, 2021.

Gao, J., Sun, C., Zhao, H., Shen, Y., Anguelov, D., Li, C., and Schmid, C. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11525–11533, 2020.

Ha, D. and Eck, D. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Homayounfar, N., Ma, W.-C., Lakshmikanth, S. K., and Urtasun, R. Hierarchical recurrent attention networks for structured online maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3417–3426, 2018.

Kuhn, H. W. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., and Beijbom, O. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12697–12705, 2019.

Lee, S., Kim, J., Shin Yoon, J., Shin, S., Bailo, O., Kim, N., Lee, T.-H., Seok Hong, H., Han, S.-H., and So Kweon, I. Vpgnet: Vanishing point guided network for lane and road marking detection and recognition. In *Proceedings of the IEEE international conference on computer vision*, pp. 1947–1955, 2017.

Li, F., Zhang, H., Liu, S., Guo, J., Ni, L. M., and Zhang, L. Dn-detr: Accelerate detr training by introducing query denoising. *arXiv preprint arXiv:2203.01305*, 2022.

Li, J., Xu, K., Chaudhuri, S., Yumer, E., Zhang, H., and Guibas, L. Grass: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017.

Li, Q., Wang, Y., Wang, Y., and Zhao, H. Hdmapnet: A local semantic map learning and evaluation framework. *arXiv preprint arXiv:2107.06307*, 2021.

Li, X., Li, J., Hu, X., and Yang, J. Line-cnn: End-to-end traffic line detection with line proposal unit. *IEEE Transactions on Intelligent Transportation Systems*, 21 (1):248–258, 2019.

Liang, J., Homayounfar, N., Ma, W.-C., Wang, S., and Urtasun, R. Convolutional recurrent network for road boundary extraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9512–9521, 2019.

Liang, J., Homayounfar, N., Ma, W.-C., Xiong, Y., Hu, R., and Urtasun, R. Polytransform: Deep polygon transformer for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9131–9140, 2020.

Liu, H., Li, C., Liu, X., and Wong, T.-T. End-to-end line drawing vectorization. 2022.

Liu, Y., Zhang, J., Fang, L., Jiang, Q., and Zhou, B. Multimodal motion prediction with stacked transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7577–7586, 2021.

Loshchilov, I. and Hutter, F. Fixing weight decay regularization in adam. 2018.

Lu, C., van de Molengraft, M. J. G., and Dubbelman, G. Monocular semantic occupancy grid mapping with convolutional variational encoder–decoder networks. *IEEE Robotics and Automation Letters*, 4(2):445–452, 2019.

Mallot, H. A., Bülthoff, H. H., Little, J., and Bohrer, S. Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biological cybernetics*, 64(3):177–185, 1991.

Mattyus, G., Wang, S., Fidler, S., and Urtasun, R. Enhancing road maps by parsing aerial images around the world. In *Proceedings of the IEEE international conference on computer vision*, pp. 1689–1697, 2015.

Máttyus, G., Wang, S., Fidler, S., and Urtasun, R. Hd maps: Fine-grained road segmentation by parsing ground and aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3611–3619, 2016.

Mi, L., Zhao, H., Nash, C., Jin, X., Gao, J., Sun, C., Schmid, C., Shavit, N., Chai, Y., and Anguelov, D. Hdmapgen: A hierarchical graph generative model of high definition maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4227–4236, 2021.

Mo, K., Guerrero, P., Yi, L., Su, H., Wonka, P., Mitra, N., and Guibas, L. J. Structurenet: Hierarchical graph networks for 3d shape generation. *arXiv preprint arXiv:1908.00575*, 2019.

Nash, C., Ganin, Y., Eslami, S. A., and Battaglia, P. Polygen: An autoregressive generative model of 3d meshes. In *International Conference on Machine Learning*, pp. 7220–7229. PMLR, 2020.

Neven, D., De Brabandere, B., Georgoulis, S., Proesmans, M., and Van Gool, L. Towards end-to-end lane detection: an instance segmentation approach. In *2018 IEEE intelligent vehicles symposium (IV)*, pp. 286–291. IEEE, 2018.

Pan, B., Sun, J., Leung, H. Y. T., Andonian, A., and Zhou, B. Cross-view semantic segmentation for sensing surroundings. *IEEE Robotics and Automation Letters*, 5(3): 4867–4873, 2020.

Pan, X., Shi, J., Luo, P., Wang, X., and Tang, X. Spatial as deep: Spatial cnn for traffic scene understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Philion, J. and Fidler, S. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d. In *European Conference on Computer Vision*, pp. 194–210. Springer, 2020.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

Ramer, U. An iterative procedure for the polygonal approximation of plane curves. *Comput. Graph. Image Process.*, 1:244–256, 1972.

Reddy, P., Gharbi, M., Lukac, M., and Mitra, N. J. Im2vec: Synthesizing vector graphics without vector supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7342–7351, 2021.

Roddick, T. and Cipolla, R. Predicting semantic map representations from images using pyramid occupancy networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11138–11147, 2020.

Rong, G., Shin, B. H., Tabatabaee, H., Lu, Q., Lemke, S., Možeiko, M., Boise, E., Uhm, G., Gerow, M., Mehta, S., et al. Lgsvl simulator: A high fidelity simulator for autonomous driving. *arXiv preprint arXiv:2005.03778*, 2020.

Sadat, A., Casas, S., Ren, M., Wu, X., Dhawan, P., and Urtasun, R. Perceive, predict, and plan: Safe motion planning through interpretable semantic representations. In *European Conference on Computer Vision*, pp. 414–430. Springer, 2020.

Van Gansbeke, W., De Brabandere, B., Neven, D., Proesmans, M., and Van Gool, L. End-to-end lane detection through differentiable least-squares fitting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, S., Fidler, S., and Urtasun, R. Holistic 3d scene understanding from a single geo-tagged image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3964–3972, 2015.

Wang, S., Bai, M., Mattyus, G., Chu, H., Luo, W., Yang, B., Liang, J., Cheverie, J., Fidler, S., and Urtasun, R. Torontocity: Seeing the world with a million eyes. *arXiv preprint arXiv:1612.00423*, 2016.

Wang, Y., Guizilini, V. C., Zhang, T., Wang, Y., Zhao, H., and Solomon, J. Detr3d: 3d object detection from multi-view images via 3d-to-2d queries. In *Conference on Robot Learning*, pp. 180–191. PMLR, 2022.

Wilson, B., Qi, W., Agarwal, T., Lambert, J., Singh, J., Khandelwal, S., Pan, B., Kumar, R., Hartnett, A., Pontes, J. K., Ramanan, D., Carr, P., and Hays, J. Argoverse 2: Next generation datasets for self-driving perception and forecasting. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks 2021)*, 2021.

Xu, Y., Xu, W., Cheung, D., and Tu, Z. Line segment detection using transformers without edges. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4257–4266, 2021.

Yamaguchi, K. Canvasvae: Learning to generate vector graphic documents. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5481–5489, 2021.

Yang, B., Liang, M., and Urtasun, R. Hdnet: Exploiting hd maps for 3d object detection. In *Conference on Robot Learning*, pp. 146–155. PMLR, 2018.

Yang, W., Li, Q., Liu, W., Yu, Y., Ma, Y., He, S., and Pan, J. Projecting your view attentively: Monocular road scene layout estimation via cross-view transformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15536–15545, 2021.

Zhang, Z. and Wang, Y. Jointnet: A common neural network for road and building extraction. *Remote Sensing*, 11(6): 696, 2019.

Zhao, H., Gao, J., Lan, T., Sun, C., Sapp, B., Varadarajan, B., Shen, Y., Shen, Y., Chai, Y., Schmid, C., et al. Tnt: Target-driven trajectory prediction. *arXiv preprint arXiv:2008.08294*, 2020.

Zhou, B. and Krähenbühl, P. Cross-view transformers for real-time map-view semantic segmentation. *arXiv preprint arXiv:2205.02833*, 2022.

Zhou, Y., Sun, P., Zhang, Y., Anguelov, D., Gao, J., Ouyang, T., Guo, J., Ngiam, J., and Vasudevan, V. End-to-end multi-view fusion for 3d object detection in lidar point clouds. In *Conference on Robot Learning*, pp. 923–932. PMLR, 2020.

Zhu, X., Su, W., Lu, L., Li, B., Wang, X., and Dai, J. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.

Zorzi, S., Bazrafkan, S., Habenschuss, S., and Fraundorfer, F. Polyworld: Polygonal building extraction with graph neural networks in satellite images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1848–1857, 2022.

Zürn, J., Vertens, J., and Burgard, W. Lane graph estimation for scene understanding in urban driving. *IEEE Robotics and Automation Letters*, 6(4):8615–8622, 2021.

A. Experiment Setup

A.1. Dataset

nuScenes We experiment on nuScenes (Caesar et al., 2020) dataset, which contains 1000 sequences of recordings collected by autonomous driving cars. Each episode is annotated at 2Hz and contains 6 camera images and LiDAR sweeps. Our dataset setup and pre-processing steps are identical to that of HDMapNet (Li et al., 2021), which includes three categories of map elements – pedestrian crossing, divider, and road boundary – from the nuScenes dataset.

Argoverse2 We further conduct experiments on Argoverse2 (Wilson et al., 2021) dataset. Like nuScenes, it contains 1000 logs (700, 150, 150 for training, validation and test set). Each episode provides 15s of 20Hz camera images, 10Hz LiDAR sweeps and a vectorized map. We use the same pre-processing settings as on nuScenes dataset.

A.2. Metrics

In contrast to existing methods which generate rasterized results, our method does not require rasterizing curves on grids. Therefore, we opt not to use Intersection-Over-Union (IoU) as a metric. We use a distance-based metric to evaluate the similarity between predicted curves and ground-truth curves. We follow the instance-level evaluation metric proposed by HDMapNet (Li et al., 2021) to compare the instance-level detection performance of our model to baseline methods. The metric is average precision (AP), where positive/negative samples are based on geometric similarity, more concretely, Chamfer distance and Fréchet distance. For clarity, we call the AP based on Chamfer distance and Fréchet distance as Chamfer AP and Fréchet AP, respectively.

Chamfer distance. Chamfer distance is a distance measure that quantifies the similarity between two *unordered* sets. The Chamfer distance is an evaluation metric that quantifies the similarity between two unordered sets by taking into account the distance of each permutation of the elements of set as follows:

$$D_{chamfer}(\mathcal{S}_1, \mathcal{S}_2) = \frac{1}{2}\left(\frac{1}{|\mathcal{S}_1|}\sum_{p\in\mathcal{S}_1}\min_{q\in\mathcal{S}_2}\|p,q\|_2 + \frac{1}{|\mathcal{S}_2|}\sum_{q\in\mathcal{S}_2}\min_{p\in\mathcal{S}_1}\|q,p\|_2\right). \tag{4}$$

In our experiments, we use chamfer distance to calculate the distance between a prediction and a ground truth polyline set, and each polyline set is represented by uniformly sampling a polyline to N_{pts} vertices, where N_{pts} is set to 100 in our experiments.

Fréchet distance. The order of polyline vertices is not measured by Chamfer distance. Therefore, we introduce Fréchet distance as an additional measure. Fréchet distance is a measure of similarity of curves that takes both the positions and the *order* of the points along the curves into consideration. Our implementation is based on discrete Fréchet distance (Eiter & Mannila, 1994; Agarwal et al., 2014).

We use the discrete version of Fréchet distance (Eiter & Mannila, 1994; Agarwal et al., 2014) to evaluate the geometric similarity between two polyline P and Q. We denote $\sigma(P)$ as a sequence of endpoints of the line segments of P. In particular, $\sigma(P) = (p_1, \ldots, p_m)$ is a sequence with m vertices that uniformly sampled from the original input polyline P, where each position of P between p_i and p_{i+1} can be approximated by using an affine transformation that is $p_{i+\lambda} = (1-\lambda)p_i + \lambda p_{i+1}$ and the m in our experiment is set as 100.

Let P and Q be polyline and $\sigma(P) = (u_1, \ldots, u_p)$ and $\sigma(Q) = (v_1, \ldots, v_q)$ the corresponding sequences. A coupling L is a sequence of distinct pairs between $\sigma(P)$ and $\sigma(Q)$:

$$(u_{a_1}, v_{b_1}), \ldots, (u_{a_m}, v_{b_m}). \tag{5}$$

These indexes $\{a_1, \ldots, a_m\}$ and $\{b_1, \ldots, b_m\}$ are nondecreasing surjection such that $a_1 = 1$, $a_m = p$, $b_1 = 1$, $b_m = q$ and for all $i < j \in \{1, \ldots, q\}$, $a_i \leq a_j$ and $b_i \leq b_j$.

We define the norm $\|L\|$ of the L is the length of the longest pair in L, that is,

$$\|L\| = \max_{i=1,\ldots,m} d(u_{a_i}, v_{b_i}). \tag{6}$$

The discrete Fréchet distance between polyline P and Q is defined to be

$$\delta_{dF}(P,Q) = \min\{\|L\|, L \text{ is a coupling between } P \text{ and } Q\}. \tag{7}$$

This equation indicates that the distance of discrete Fréchet distance is the minimum norm of all possible couplings. To Find the coupling plausible L that has the minimum norm, we use a Dynamic programming-based algorithm that is described in Algorithm 1.

Algorithm 1 The Algorithm of Discrete Fréchet Distance

```
Input: polyline $P = (u_1, \ldots, u_p)$ and $Q = (v_1, \ldots, v_q)$.
Output: $\delta_{dF}(P, Q)$
$ca$ : an 2d array of real with size of $(p \times q)$  Function $c\,(i,\ j)$
    if $ca(i,j) > -1$ then
        return $ca(i,j)$
    else if $i = 1$ and $j = 1$ then
        $ca(i,j) := d(u1, v1)$
    else if $i > 1$ and $j = 1$ then
        $ca(i,j) := \max\{c(i-1,1), d(u_i, v_1)\}$
    else if $i = 1$ and $j > 1$ then
        $ca(i,j) := \max\{c(1, j-1),\ d(u_1, v_j)\}$
    else if $i > 1$ and $j > 1$ then
        $ca(i,j) := \max\{\min(c(i-1,j), c(i-1,j-1), c(i,j-1)), d(u_i, v_j)\}$
    else
        $ca(i,j) := \infty$
    end
    return $ca(i,j)$
end
begin
    for $i = 1$ to $p$ do
        for $j = 1$ to $q$ do
            ca(i, j) := -1.0;
        end
    end
    return $c(p,q)$
end
```

B. More Qualitative results of VectorMapNet

B.1. Visualization results of VectorMapNet(Fusion)

We visualized three cases of VectorMapNet (Fusion) and VectorMapNet (Camera) to demonstrate that LiDAR information can complement visual information to generate more robust map predictions. In the first case, the camera view is constrained by the nearby vehicles, so it can not provide helpful surrounding information. LiDAR sensor bypasses the nearby vehicle and provides some cue for VectorMapNet to generate a better result than its camera-only counterpart (see Figure 7). For the second case (see Figure 8), the model cannot detect the nearby parking gate because it locates in the blind zone of cameras. In contrast, the LiDAR provides depth information and helps the VectorMapNet(Fusion) detect the missing lane boundary. LiDAR points can prevent the model from falsely detecting map elements in bad weather conditions as well. As shown in Figure 9, some puddles are near the intersection. With the light reflection, these puddles visually look like a lane boundary. However, the LiDAR data shows that there does not have any bump in there. Unlike the camera-only model, this depth information from LiDAR helps our fusion model not generate a non existed lane boundary.

C. Implementation details

C.1. Overall Architectures.

BEV feature extractor outputs a feature map with a size of $(200, 100, 128)$. It uses ResNet50 (He et al., 2016) for shared CNN backbone. We use a single layer PointNet (Qi et al., 2017) whose outputs have 64 dimensions as the LiDAR backbone to aggregate LiDAR points into a pillar. We set the number of element queries $N_{\max}$ in map element detector as 100.

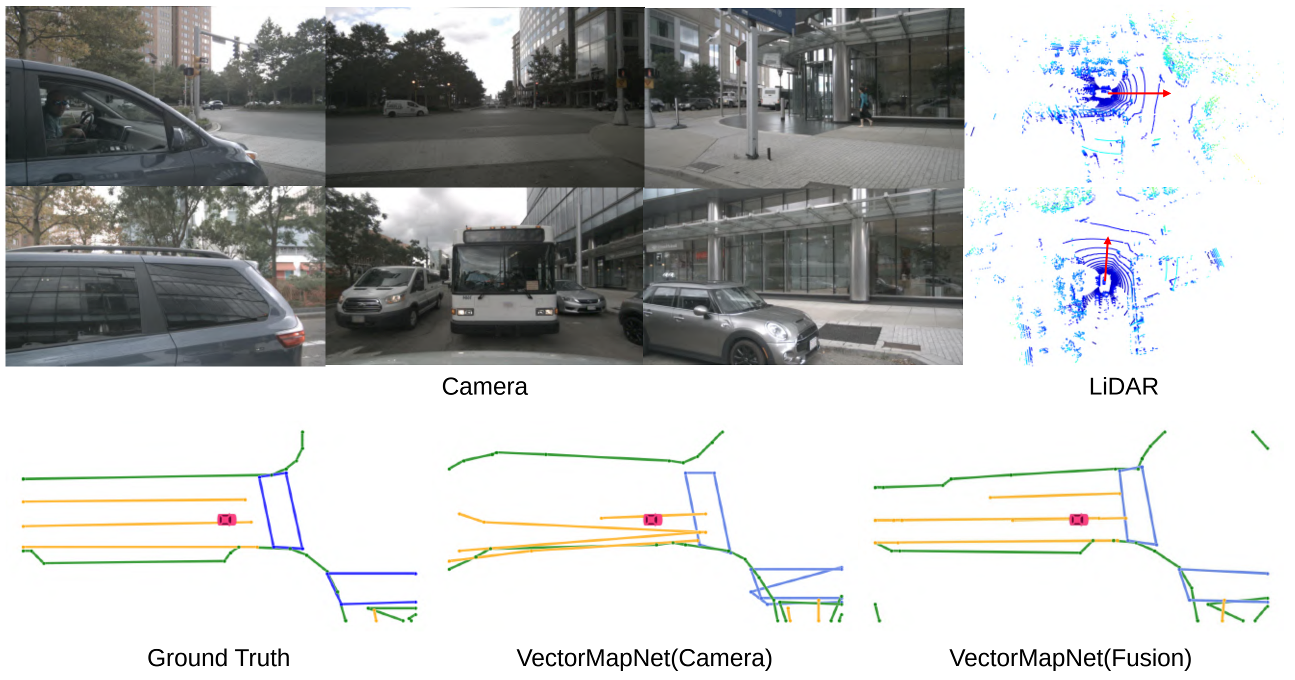

Figure 7: When the ego car cameras are occluded by the nearby vehicles, VectorMapNet(Camera) can not precept the surrounding map. With the depth cue from LiDAR, VectorMapNet(Fusion) can generate a more plausible result than its camera counterpart.

The transformer decoders we used in map element detector and polyline generator both have 6 decoder layers, and their hidden embeddings' size is 256. For the output space of polyline generator, we divide the map space (see § 3.4) evenly into 200×100 rectangular grids, and each grid has a size of $0.3m \times 0.3m$.

C.2. Training settings.

We train all our models on 8 GTX3090 GPUs for 110 epochs with a total batch size of 32. We use AdamW (Loshchilov & Hutter, 2018) optimizer with a gradient clipping norm of 5.0. For the learning rate schedule, we use a step schedule that multiplies a learning rate by 0.1 at epoch 100 and has a linear warm-up period at the first 5000 steps. The dropout rate for all modules is 0.2, following the transformer's settings (Vaswani et al., 2017). Data augmentation is only deployed during polyline generator's training; specifically, two I.I.D. Gaussian noises are added to each input vertex's x and y coordinates with a probability of 0.3.

C.3. Model Details

Camera Branch of Map Feature Extractor. For image data $\mathcal{I}$, we use a shared CNN backbone to obtain each camera's image features in the camera space, then use the Inverse Perspective Mapping (IPM) (Mallot et al., 1991) technique to transform these features into BEV space. Since the depth information is missing in camera images, we follow one common approach that assumes the ground is mostly planar and transforms the images to BEV via homography. Without knowing the exact height of the ground plane, this homography is not an accurate transformation. To alleviate this issue, we transform the image features into four BEV planes with different heights (we use $(-1m, 0m, 1m, 2m)$ in practice). The camera BEV features $\mathcal{F}_{\text{BEV}}^{\mathcal{I}} \in \mathbb{R}^{W \times H \times C_1}$ are the concatenation of these feature maps.

C.4. Loss

Loss settings. The loss function of map element detector is a linear combination of three parts: a negative log-likelihood for element keypoint classification, a smooth L1 loss, and an IoU loss for keypoints regression. The coefficients of these loss components are $2, 0.1, 1$. The matching cost of map element detector is the same as the loss combination. The loss function

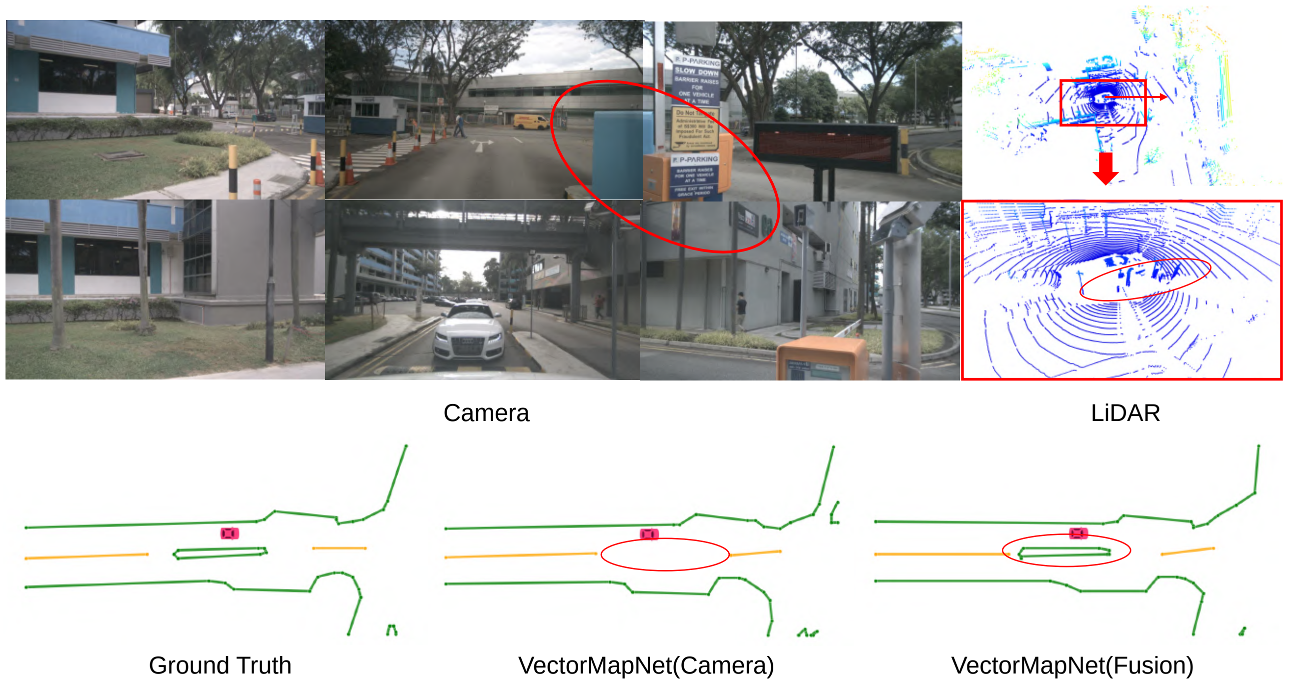

Figure 8: The blind area of onboard cameras may cause our model to miss the map elements closed ego vehicle. In contrast, we can easily find that LiDAR data has sensed some obstacles near the ego vehicle in the right-most column. With these cues, our fusion model detects the missed lane boundary by our camera-only model.

of polyline generator is a negative log-likelihood. We train VectorMapNet by simply summing up these losses.

map element detector loss. To get the loss, we first establish a correspondence between the ground-truth ($\mathcal{A}$, $\boldsymbol{\mathcal{L}}$) and the prediction ($\hat{\mathcal{A}}$, $\hat{\boldsymbol{\mathcal{L}}}$). Assuming the number of ground-truth map element keypoints N is smaller than the number of predictions N_{max}, and we pad the set of ground-truth ($\mathcal{A}$, $\boldsymbol{\mathcal{L}}$) with $\emptyset$s (no object) up to N_{max}. The correspondence σ is a permutation of N_{max} elements $\sigma \in \mathcal{P}$ with the lowest cost: $\sigma^* = \underset{\sigma \in \mathcal{P}}{argmin} \sum_{j=1}^{N_{max}} -\mathbb{1}_{(l_j \neq \emptyset)} \hat{p}_{\sigma(j)}(l_j) + -\mathbb{1}_{(l_j \neq \emptyset)} \boldsymbol{\mathcal{L}}_{keypoint}(a_j, \hat{a}_{\sigma(j)})$, where $\hat{p}_{\sigma(j)}(l_j)$ is the probability of class label l_j for the prediction with index $\sigma(j)$, and the loss of keypoints parameters $\boldsymbol{\mathcal{L}}_{keypoint}$ is an addition of a smooth L1 loss and an IoU loss. With these notations we define the loss of detector as:

$$\boldsymbol{\mathcal{L}}_{det} = \sum_{j=1}^{N_{max}} -\log \hat{p}_{\sigma^*(j)}(l_j) + \mathbb{1}_{(l_j \neq \emptyset)} \boldsymbol{\mathcal{L}}_{keypoint}(a_j, \hat{a}_{\sigma^*(j)}),$$

where σ^* is the optimal assignment computed by Hungarian algorithm (Kuhn, 1955).

C.5. Baseline model

HDMapNet For all experiments in our paper, we employed the official HDMapNet model from the provided codebase and directly take its vectorized results. As the Argoverse dataset was not included in the original HDMapNet paper, we adapted the NuScenes data processing steps from their codebase to create an Argoverse2 dataloader for our experiments.

STSU For STSU, It uses a transformer module to detect the moving objects and centerline segments. It uses an association head to piece the segments together as the road graph. In order to adapt STSU to our task, we use a two-layer MLP to predict lane segments and only keep its object branch and polyline branch.

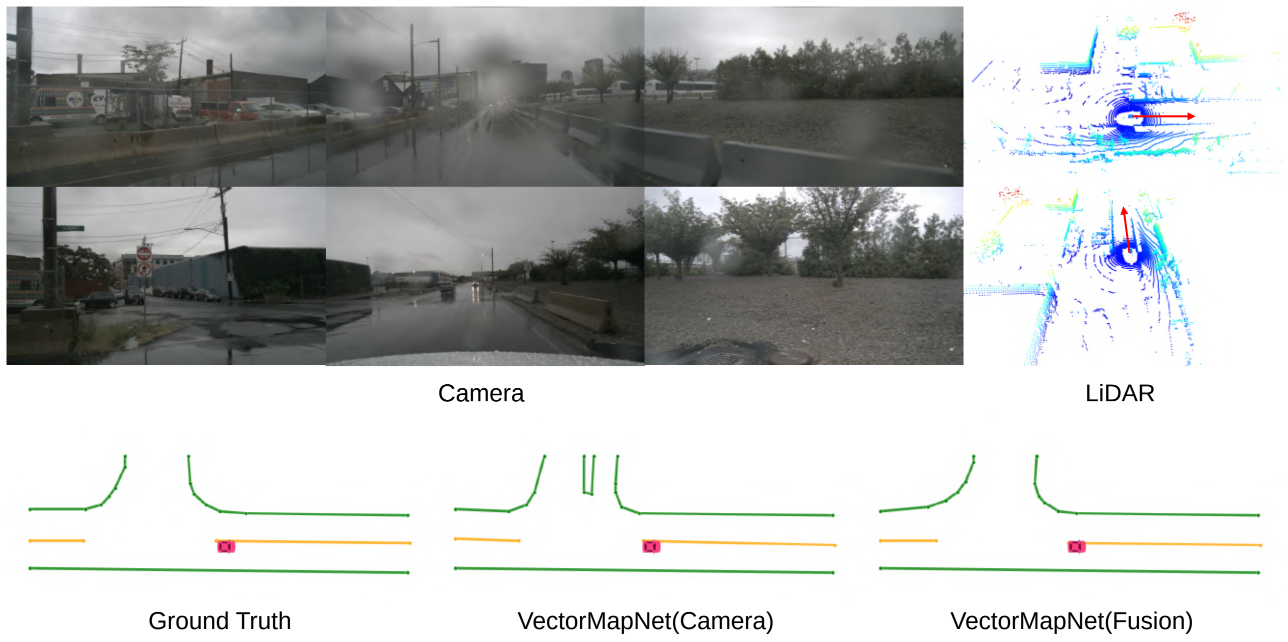

Figure 9: The qualitative results of VectorMapNet in bad weather conditions. VectorMapNet(Camera) falsely detects these puddles near the intersection as a lane boundary. The fusion result shows that the miss detection issue can be resolved by combining the depth information.

Table 5: Ablation study of curves sampling strategies.

	Fréchet Distance				Chamfer Distance			
Vertex Sampling Method	AP$_{ped}$	AP$_{divider}$	AP$_{boundary}$	mAP	AP$_{ped}$	AP$_{divider}$	AP$_{boundary}$	mAP
curvature-based	47.0	47.4	56.9	50.4	27.6	34.4	35.4	32.5
fixed interval	26.0	23.6	37.1	28.9	14.6	17.6	18.7	17.0

D. More Ablation Studies

D.1. Curve sampling strategies

We use two approaches to sample polylines. The first is based on the original nuScenes setting (Caesar et al., 2020), which samples vertices at the position where the curvature changes are beyond a certain threshold. The second is to sample the vertices at fixed intervals ($1m$). We compare our methods under these two sampling strategies and the results are shown in Table 5. The curvature-based sampling outperforms its fixed-sampling counterpart by a large margin and achieves a leading 21.5 Fréchet mAP and 15.5 Chamfer mAP. We hypothesize that the fixed-sampling method involves a large set of redundant vertices that have negligible contributions to the geometry, thus under-weighs the essential vertices ($e.g.$ the vertices at the corner of a polyline) in the learning process.

D.2. Vertex modeling methods.

We investigate both discrete and continuous ways to model polyline vertices. The discrete version of polyline generator is described in § 3.4. With the same model structure, we follow SketchRNN (Ha & Eck, 2017) and use mixture of Gaussian distributions to model the vertices of polylines as continuous variables. The comparison is shown in Table 6. We find that using discrete embeddings vertex coordinates results in a considerable gain in performance, with Chamfer mAP increasing

Table 6: Ablation study of vertex modeling methods.

	Fréchet Distance				Chamfer Distance			
Modeling Method	AP_{ped}	$\text{AP}_{divider}$	$\text{AP}_{boundary}$	mAP	AP_{ped}	$\text{AP}_{divider}$	$\text{AP}_{boundary}$	mAP
discrete	47.0	47.4	56.9	50.4	27.6	34.4	35.4	32.5
continuous	38.0	41.6	46.1	41.9	26.5	28.1	30.1	26.5

from 18.2 to 32.5 and the Fréchet mAP increasing from 26.8 to 50.4. These improvements suggest that the non-local characteristic of categorical distribution helps our model to capture complex vertex coordinate distributions.

D.3. Extrinsic Robustness

Thanks for this suggestion. To probe the robustness of VectorMapNet, we follow lift-splat-shoot(Philion & Fidler, 2020), which tests the model under the noise that occurs in self-driving, such as camera extrinsic being biased. The table 7 shows that training the model with noisy extrinsic can lead to better test-time performance. And our model maintains its good performance for high amounts of extrinsic noise. The results show our model's robustness against extrinsic noise.

Table 7: VectorMapNet performance under different extrinsic noise.

mAP	Test time extrinsic noise			
Train time extrinsic noise	0	0.1	0.3	0.6
0	42.2	42.6	42.4	42.5
0.1	**43.8**	43.6	43.5	43.6
0.3	43.0	43.0	43.1	43.1
0.6	42.6	42.72	42.8	42.9

E. Additional Discussions

The potential negative societal impact. While there are legitimate concerns regarding privacy issues in autonomous driving systems that use generated maps, we'd like to reassure that our method is designed with such concerns in mind. Our proposed VectorMapNet model relies solely on onboard sensor observations and doesn't track any global locations or individual movements. Consequently, it poses no risk of leaking personal information, such as patterns in individuals' movements, thus upholding the highest standards of privacy.

Confidence indicator. Learning-based models can certainly provide confidence indicators to describe prediction uncertainty. Our model can generate two types of confidence scores: (1) The DETR-like Map Element Detector generates a confidence score for each detected map element. It is an instance-level score. (2) The auto-regressive Polyline Generator generates a score for each point on a polyline. It is a point-level score. Both of them can indicate the confidence or likelihood of the model's prediction. However, how to better use this uncertainty for downstream tasks still remains an open question. We leave it for future research.