HDMapGen: A Hierarchical Graph Generative Model of High Definition Maps

Lu Mi[1,2,†,*], Hang Zhao[1,*], Charlie Nash[3], Xiaohan Jin[1], Jiyang Gao[1], Chen Sun[4],
Cordelia Schmid[4], Nir Shavit[2], Yuning Chai[1], Dragomir Anguelov[1]
[1]Waymo, [2]MIT, [3]DeepMind, [4]Google

Abstract

High Definition (HD) maps are maps with precise definitions of road lanes with rich semantics of the traffic rules. They are critical for several key stages in an autonomous driving system, including motion forecasting and planning. However, there are only a small amount of real-world road topologies and geometries, which significantly limits our ability to test out the self-driving stack to generalize onto new unseen scenarios. To address this issue, we introduce a new challenging task to generate HD maps. In this work, we explore several autoregressive models using different data representations, including sequence, plain graph, and hierarchical graph. We propose HDMapGen, a hierarchical graph generation model capable of producing high-quality and diverse HD maps through a coarse-to-fine approach. Experiments on the Argoverse dataset and an in-house dataset show that HDMapGen significantly outperforms baseline methods. Additionally, we demonstrate that HDMapGen achieves high scalability and efficiency.

1. Introduction

High Definition maps (HD maps) are electronic maps with precise depictions of the physical roads, usually with an accuracy of centimeters, together with rich semantics of traffic rules, such as one-way-street, stop, yield, *etc*. HD maps sit at the core of autonomous driving applications as they provide a strong prior for the self-driving robots to localize themselves in the 3D space [4, 34], predict other vehicles' motions [6, 40] and maneuver themselves around [8, 20]. Furthermore, HD maps are critical building blocks of city modeling and simulation. Simulating novel city environments finds a wide range of applications in game design and urban planning.

Practically, HD maps are constructed according to a strict mapping procedure: first, a fleet of vehicles with mapping sensor suites (containing LiDAR, Radar, and camera) are sent to capture the scene; then, the sensor data are processed and stitched together to obtain map imagery; finally, human specialists annotate on top of the imagery to provide vectorized representations of the world geometry with semantic attributes. On the other hand, we aim to produce synthetic HD maps in a data-driven way, mainly due to two reasons: (1) building HD maps from the real world is prohibitively expensive; (2) the small number of real-world maps prevent us from testing the generalization capability of self-driving stacks in simulation, such as motion forecasting and motion planning.

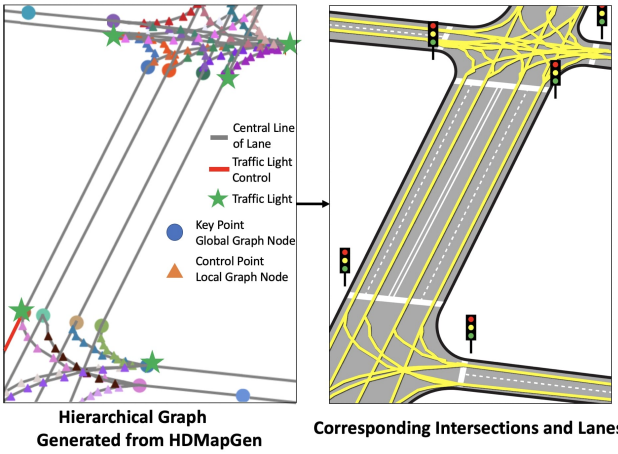

Figure 1: Left: a hierarchical map graph generated from HDMapGen. One traffic light controlled lane is shown as an example; Right: a map with corresponding intersections and lanes rendered from the hierarchical map graph.

Existing methods in modeling cities and maps are mostly relying on procedural modeling, and hand-crafted generation rules [30], and are therefore not flexible and adaptable to new scenarios. There is no previous attempt to generate HD maps using modern deep generative models to the best of our knowledge. The most related works are those which generate city layouts [10]. Compared to HD maps, city layouts are not suitable for autonomous driving applications since they only contain coarse locations of the roads (with a resolution of roughly 10 meters) and lack details such as lanes of the roads or traffic lights.

†Work done during internship at Waymo.
*Corresponding to: lumi@mit.edu, zhaohang0124@gmail.com

Unique attributes of HD maps pose new challenges to modeling them: (1) HD maps are composed of physical road elements with geometric features, *e.g*. marked straight lanes and hypothesized turning lanes; (2) Lane maps contain rich semantic attributes, *e.g*. the direction of lanes, the association between traffic lights and lanes. By addressing these challenges, our major contributions in this work are as follows:

- We pose a new important and challenging problem to generate HD lane maps in a data-driven way.
- We perform a systematic exploration of modern autoregressive generative models with different data representations and propose HDMapGen, a hierarchical graph generative model that largely outperforms other baselines.
- We evaluate our model on the maps of the public Argoverse dataset and an in-house dataset, covering cities of Miami, Pittsburg, and San Francisco. Results show that our model produces maps with high fidelity, diversity, scalability, and efficiency.

2. Related Work

2.1. Street Map Modeling and Generation

City street modeling and generation is an important component of computer-aided urban design. Most classical works rely on procedural modeling methods [1, 2, 37, 5, 9, 12, 13, 28, 30, 38]. One of the more well-known methods is the L-system [30], which generates road networks from a sequence of instructions defined by hand-crafted production rules. To impose more control over the generated results, later works proposed to sample from procedural models according to constraints or likelihood functions [33]. However, these methods are still constrained by the rules, therefore are not flexible in terms of generating maps with different city styles.

In recent years, deep learning methods have been applied to map reconstruction and encoding. Several works [3, 18, 22, 25] attempted to extract and reconstruct road topologies from overhead images. VectorNet [14] and LaneGCN [23] proposed to efficiently encode roads using graph attention networks and graph convolutions, replacing traditional rendering-based models. Chu *et al*. propose Neural Turtle Graphics (NTG) [10], a generative model to produce roads iteratively. They use an encoder-decoder RNN model that encodes incoming paths into a node and decodes outgoing nodes and edges. The goal of NTG is to generate city-level road layouts. In comparison, we focus on high definition road and lane generation, which is much more challenging.

2.2. Graph Generative Models

Our model architecture is inspired by the rapid progress in the area of graph generation spearheaded by seminal works such as graph recurrent neural networks (GRNN) [17], graph generative adversarial networks (GraphGAN) [36], variational graph auto-encoder (VGAE) [19], and graph recurrent attention networks (GRAN) [24]. More recently, Cao *et al*. proposed MolGAN [11], and Samanta *et al*. proposed NEVAE [31] to generate small molecule graphs. These methods all focus on generating graph topologies, such as adjacency matrices. In comparison, HD lane maps are graphs that come with spatial coordinates and geometric features, which poses another layer of complexity to the generation problem.

2.3. Geometric Data Generation

Given the geometric nature of maps, another line of related work is geometric data generation. To generate 2D sketch drawings, Ha *et al*. proposed SketchRNN [16], an RNN model with a VAE structure to sequentially produce sketch strokes following human drawing sequences. To assemble furniture from primitive parts, GRASS [21] and StructureNet [26] adopted several variations of auto-encoding models. More recently, Nash *et al*. proposed PolyGen [27], an autoregressive transformer model that generates 3D furniture meshes. Compared to human sketches, furniture, and molecules, graphs of lane maps usually consist of more nodes and edges. Instead of treating the whole map as a sequence, our method generates a map with a two-level hierarchy, greatly improving the generation quality.

3. Method

In Section 3.1, we explore different data representations to generate HD maps and demonstrate the efficiency of using hierarchical graph. In Section 3.2, we introduce our autoregressive graph generative model HDMapGen that uses a hierarchical graph as data representation.

3.1. Data Representation

To provide detailed road information to the autonomous vehicle, HD maps used in applications such as autonomous driving typically contain many essential components, including lanes, boundaries, crosswalks, traffic lights, stop signs, *etc*. In this work, we focus on generating lanes, which are a core component of HD maps. A lane is usually represented geometrically by its central line as a curve, and the curve is stored as a polyline with a sufficient amount of control points on it to reconstruct the curvature. Our goal is to generate these control points of central lane lines. Meanwhile, each lane has a unique ID. Its predecessor and successor lane IDs are also provided. Moreover, the lanes also

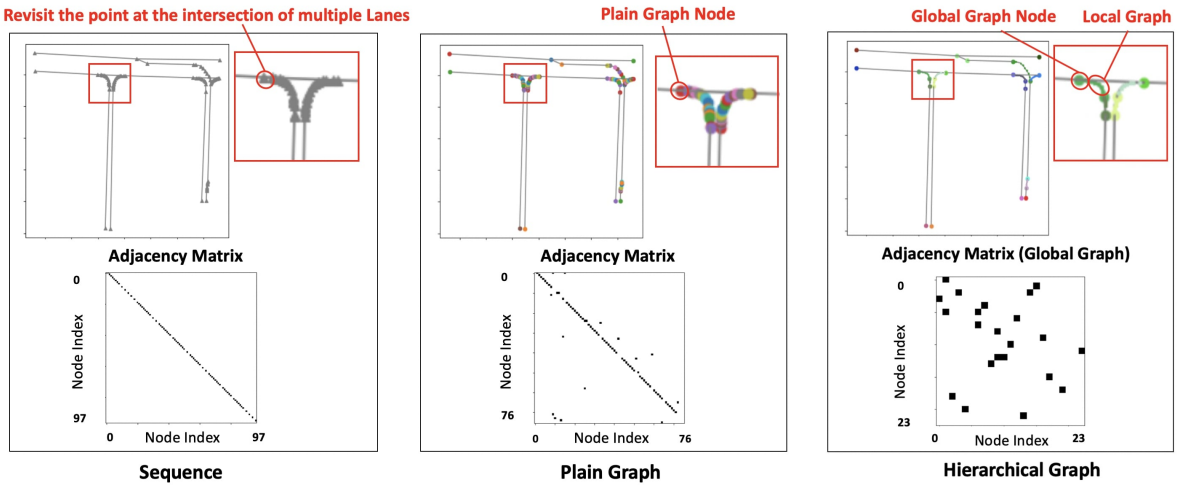

Figure 2: Different data representations of HD maps. Sequence representation leads to a big diagonal adjacency matrix, and it also suffers from revisiting the intersection points multiple times; plain graph representation results in a big sparse adjacency matrix; our proposed hierarchical graph representation gives a smaller and denser adjacency matrix (of the global graph), which reduces generation difficulty and improves efficiency.

have semantic attributes such as whether any traffic light controls it or not.

For lane map generation, there are various ways to represent the lane objects. We explore three different data representations: a sequence, a plain graph, and a hierarchical graph, as shown in Figure 2. We will later show that the hierarchical graph representation that HDMapGen utilizes is the most efficient and scalable. Moreover, it vastly outperforms other representation methods.

Sequence. While assuming all lines (lanes in our case) are connected, a straightforward representation is to sort all the lines in a map and treat the entire map as one sequence of points. This approach was adopted by SketchRNN [16]. Each point in the sequence has an offset distance in the x and y direction $(\Delta x, \Delta y)$ from the previous point, and a state variable $q \in \{1, 2, 3\}$. The state $q = 1$ indicates that the next point is starting a new lane object; the state $q = 2$ identifies the next point to continue in the same lane object, and the state $q = 3$ indicates that the entire map generation has completed. One of the major disadvantages of this method is that there is no perfect way to pre-define a sequential ordering of these points. Secondly, to cover an intersection point of multiple lanes requires revisiting that point more than once. Under this representation, an intersection point will be represented as multiple independent points in the sequence.

Plain Graph. The second baseline is to use a plain graph to represent a map. This strategy was adopted in recent work on road layout generation NTG [10]. Under this representation, a plain graph contains all control points as nodes and all line connections between control points as edges. Moreover, each node has a node attribute of 2D coordinates (x, y). The points inside a lane object have a degree of two, while the intersection points have a degree greater than two. This representation solves the revisiting issue of sequence generation. However, the method is still inefficient due to a large number of control points to guarantee a high-resolution map. To generate an edge between every pair of control points, it resorts to a very large adjacency matrix.

Hierarchical Graph (HDMapGen). Here, we propose an efficient and scalable alternative to use a hierarchical graph to represent the map. Under this approach, we first construct a *global graph* with *key points* as its nodes. Key points are defined as the endpoints of lanes or the intersection points of multiple lanes. Then the edge of the global graph represents whether a lane exists between two key points. Since key points are a small subset (30%) of original points, which require a much smaller adjacency matrix. In the second step, we represent each lane's curvatures details with a *local graph*. Each lane is constructed with a uniquely defined sequence of *control points* between two key points. Given this fixed topology, we could directly predict the coordinates of the local nodes. Moreover, we also predict a corresponding node mask to allow variations of the number of control points during generation. As demonstrated in Figure 2, using a hierarchical graph has a better performance to preserve the natural hierarchical structure information of the HD map. Moreover, it also enables a smaller size of the adjacency matrix and higher efficiency than using plain graph and sequence.

3.2. Autoregressive Modeling

Autoregressive modeling has achieved remarkable performance in many generation tasks[15, 27], especially for sequential data such as speech [29] and text [32]. More recently, several studies such as GraphRNN [39] and GRAN [24] also took an autoregressive approach for the graph generation. In this work, we also use autoregressive modeling for HD map generation.

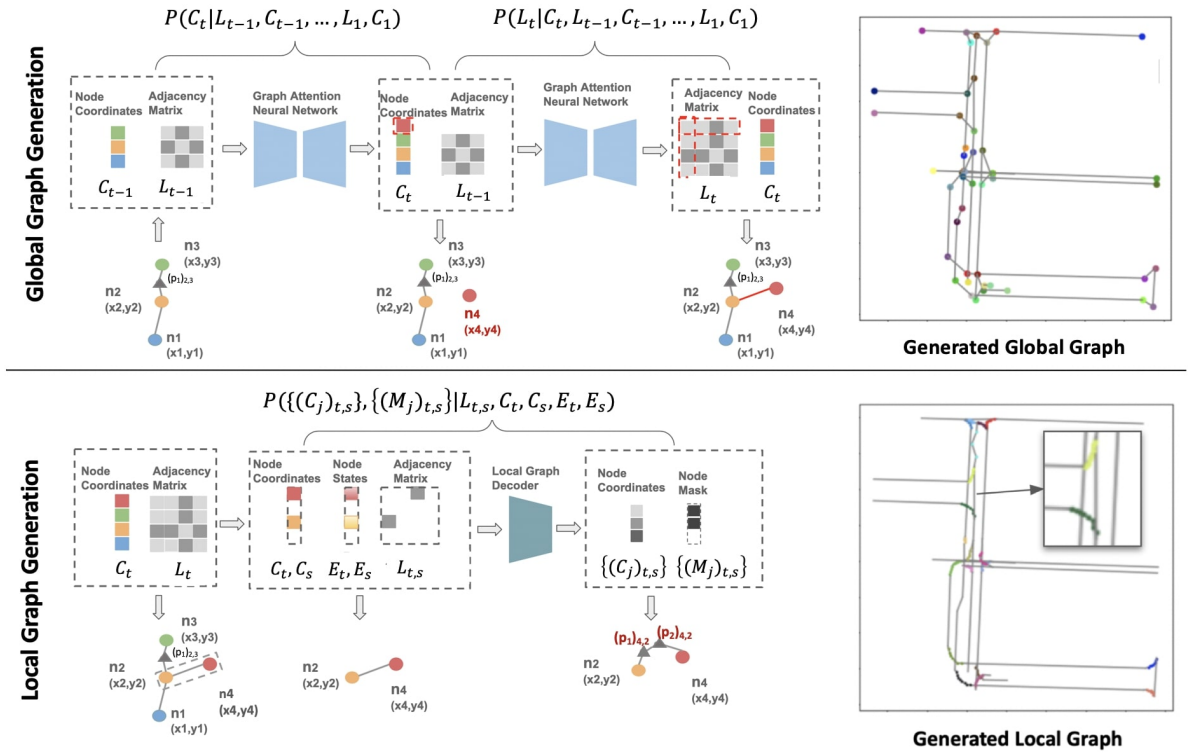

Figure 3: HDMapGen pipeline. At each step, global graph generator (top) produces a new node with its coordinates and its connections to existing nodes, which is a row and a column in the adjacency matrix; local graph generator (bottom) further decodes coordinates of local nodes between the two connected global nodes. We also demonstrate the global graph and the local graph generated from HDMapGen.

Our proposed HDMapGen is a two-level hierarchical graph generative model, as shown in Figure 3. Firstly, the global graph generation process outputs a graph with both topological structures represented by adjacency matrix L, and the geometric features represented by nodes' spatial coordinates C. Each node of the global graph is generated autoregressively. In our task, we first model the global graph as an undirected graph and later assign the lane direction information as the sequential order in the local graph. At the current step t, the global graph decoder predicts the C_t of this new node at t and all corresponding edges (lanes) $L_{t,s}$ between previously generated nodes s in a single shot, where $s \in [1, t-1]$. Then, it constructs a new row L_t and column (due to the symmetry for the undirected graph) of the adjacency matrix L. Then, we apply with a local graph decoder, which outputs the curvature details in each lane object $L_{t,s}$. The local graph outputs for each global edge $L_{t,s}$ are two sequences with a fixed length W of coordinates $\{(C_j)_{t,s}\}$ and valid mask $\{(M_j)_{t,s}\}$ for each local node j, where $j \in [1, W]$. The mask defines which node is valid to allow a variation of the number of nodes in each local graph. The local graph decoder is also generating the local graphs for all lane objects $L_{t,s}$ in a single shot, which enables very fast and efficient running. We further add another single-shot semantic attribute decoder to predict semantic features of all generated edges $L_{t,s}$. We introduce the details of the key components of HDMapGen in the following text.

Global Graph Generation. To apply autoregressive modeling, we firstly factorize the probability of generating adjacency matrix L and coordinates C of the global graph as

$$P(L, C) = \prod_{t=1}^{T} P(L_t, C_t \mid L_1, C_1, \cdots, L_{t-1}, C_{t-1}),$$

where $P(L_t, C_t \mid L_1, C_1, \cdots, L_{t-1}, C_{t-1})$ defines that at the current step t, the conditional probability of generating every new node's attributes C_t and its edges L_t with nodes generated from all previous steps 1 to $t-1$.

Furthermore, since the process of generating L_t and C_t at each step t are not independent, we explore three variants of global graph decoder to study the performance with different priority of generating L_t or C_t. We use *coordinate-first* to refer to generating node coordinate first and then topology, defined as $P(L_t, C_t) = P(L_t \mid C_t)P(C_t)$, as shown in Figure 3. And we use *topology-first* to refer to first generating graph topology and then generate node coordinates, defined as $P(L_t, C_t) = P(C_t \mid L_t)P(L_t)$. Another simplified strategy ignores the dependency between nodes and edges is named as *independent*, defined as $P(L_t, C_t) = P(C_t)P(L_t)$. More details are in Supplementary.

Inspired by GRAN [24], which has state-of-the-art performance in sample quality and time efficiency, we also incorporate a recurrent graph attention network for the global graph generation. The graph attention module [35] per-

forms the node state update with the attentive message passing, enabling a better representation of global information. In the model, we firstly define an initial node state E_s^0 before the message passing at each step t as $E_s^0 = W_L L_s + W_C C_s + b, s \in [1, t-1]$, where W_L, W_C and b are parameters of the MLP encoders taking topology and geometry inputs. Then for all nodes including new node t and nodes s already generated, we perform multiple runs of message passing. For each run r, the message $m_{i,k}^r$ between node i and its neighbor node $k \in \mathcal{N}(i)$ is $m_{i,k}^r = f(E_i^r - E_k^r)$, where $\mathcal{N}(i)$ are the neighbor nodes of i. And a binary mask is defined as $B_i = 0$ to indicate if node i is already generated, or defined as $B_i = 1$ if under construction at step t. The node state E_i^r is further concatenated with this mask B_i into $\tilde{E_i^r} = [E_i^r, B_i]$. And the attention weights a_{ik}^r for edge $L_{i,k}$ is defined as $a_{ik}^r = Sigmoid(g(\tilde{E_i^r} - \tilde{E_i^k}))$. And the node state update is then calculated as $E_i^{r+1} = GRU(E_i^r, \sum_{k \in \mathcal{N}(i)} a_{ik}^r m_{ik}^r)$. In this experiment, we use MLPs for both f and g. The GNN model has 7 layers and a propagation number of 1. After the message passing is completed, we take the final node states E_t and E_s after message passing, and use model $MLP(E_t - E_s)$ to decode them into a mixture of Bernoulli distribution for topology outputs L_t. And another model $MLP(E_t)$ is applied to generate a 2D Gaussian mixture model (GMM) for coordinate outputs C_t. The entire model is optimized with the minimization of negative log-likelihood (NLL) for coordinate outputs and binary cross-entropy (BCE) for topology outputs. And for the GMM, we further add a temperature term τ to control the diversity (variance) during the sampling. The original standard deviation σ_x and σ_y in GMM are modified into $\sigma_x \tau$ and $\sigma_y \tau$. We follow the common teacher-forcing strategy for autoregressive model training by providing ground truth of $L_1, C_1, \cdots, L_{t-1}, C_{t-1}$ as inputs when predicting $P(L_t, C_t)$, which avoids performing the reparameterization trick for the sampling process during the backpropagation.

Local Graph Generation. As the local graph to represent the curvature details of lanes always has a unique topology as a sequence of control points, we use a padding vector with a fixed maximum length W to represent the sequence. We model it as a sequence of coordinate output $\{(C_j)_{t,s}\}$ and a corresponding valid mask $\{(M_j)_{t,s}\}$, where $j \in [1, W]$. Then the conditional probability of the local graph during generation is defined as $P(\{(C_j)_{t,s}\}, \{(M_j)_{t,s}\} \mid L_{t,s}, C_t, C_s, E_t, E_s)$. The valid mask enables a variation in the number of nodes in each graph. For the straight lines which have been filtered with redundant control points after map preprocessing, the valid mask $\{(M_j)_{t,s}\}$ is all $0s$, while for the lanes at the corner which usually have multiple control points remained to guarantee the smoothness, $\{(M_j)_{t,s}\}$ is likely to have more values of 1. And we use an MLP model to generate the local graph. The models are optimized with the minimization of mean square error (MSE) for coordinate output $\{(C_j)_{t,s}\}$, and with the minimization of BCE for the valid mask $\{(M_j)_{t,s}\}$.

Semantic Attribute Generation. Like the local graph generation, we predict an additional attribute, traffic lights feature for the generated edge. Given a lane $L_{t,s}$ between node t and node s, we predict whether the lane is controlled by traffic lights with an additional edge feature decoder using MLP. We train this edge feature decoder for binary classification with the minimization of BCE.

4. Experiments

In this section, we demonstrate the efficacy of our proposed HDMapGen model to generate high-quality maps. We explore three autoregressive generative models for sequence, plain graph, and hierarchical graph input representations and evaluate generation quality, diversity, scalability, and efficiency.

4.1. Dataset and Implementation Details

Datasets. We evaluate the performance of our models on two datasets across three cities in the United States. One is the public Argoverse dataset [7], which covers Miami (total size of 204 kilometers of lanes) and Pittsburg (total size of 86 kilometers). We randomly sample 12000 maps with a field-of-view (FoV) of $200m \times 200m$ as the training dataset for Miami and 5000 maps with the same FoV for Pittsburg. We also evaluate an in-house dataset, which covers maps from the city of San Francisco. We train on 6000 maps with a FoV of $120m \times 120m$.

Map Preprocessing. The key components of HD maps, the central line of drivable lanes, are usually over-sampled to guarantee a sufficient resolution. However, graph neural network is limited in scalability. So in a pre-processing step, we remove redundant control points with a small variation of curvature in each lane. This step enables us to remove 70% of points while not compromise the map quality. We then define a hierarchical spatial graph based on the preprocessed vector map. We use Depth-First-Search (DFS) to construct the global graph's adjacency matrix and define the generation order for autoregressive modeling. The start node is randomly selected. Then we define the sequence of control points between global nodes as a local graph. The sequential order is the same as the lane direction.

Implementation Details. HDMapGen uses graph attention neural network as the core part to perform the attentive message passing for graph inputs. We use multi-layer-perceptrons (MLPs) for all other encoding and decoding steps. The graph node coordinates are normalized to $[-1, 1]$. The model is trained on a single Tesla V100 GPU with the Adam optimizer, where we use an initial learning rate of 0.0001 with a momentum of 0.9.

4.2. Baselines

We compare the quality of our hierarchical graph generative model HDMapGen with a sequence generative model in SketchRNN [16] and a plain graph generative model PlainGen derived from the global generation step of HDMapGen.

SketchRNN. We map all control points of the lane objects inside each HD map into a sequence for a sequence generative model. Each sequential data point has a pair of 2D coordinates and a state variable to define each line's continuity. We choose an ascending order of spatial coordinates to define the sequential order. We explore conditional and unconditional variants for this model. For the conditional generation, a target map is provided as both input and target as a variational auto-encoder during training. Meanwhile, a target map is still provided as input during sampling. For an unconditional generation, a decoder-only model is trained to generate the target maps.

PlainGen. The plain graph generative model has the same implementation as the global graph generative model in HDMapGen, and it takes the entire plain graph as inputs. Every control point in the map is taken as a global node in the plain graph. A corresponding edge is constructed to define whether these two nodes are connected. We use DFS to construct the adjacency matrix and define the order of generation. And we explore two variants of *coordinate-first* and *topology-first* for this model.

4.3. Qualitative Analysis

Global Graph Generation. We first show the global graph from HDMapGen in Figure 4. We could see with only a small set of global nodes could represent a typical pattern in HD maps to include two parallel crossroads. We also demonstrate that the diversity of model outputs could be improved by increasing the temperature τ. While the diversity and realism trade-off still exist during generation, we find an empirical bound of 0.2 for τ to guarantee high-quality samples in our experiments.

Local Graph Generation. The local graph represents a sequence of control points that reveal the curvature details of each lane object. For lanes, the density of control points usually increases at the corner of each lane. In Figure 5, we show our model is capable of generating such features to have smooth curves at the corner of each lane.

Semantic Attribute Generation. As shown in Figure 5, the traffic lights and the traffic light controlled lanes generated from HDMapGen are mostly located on the crossroads and turn roads. The generated semantic attributes are consistent with the real-world urban scenes.

Comparison with Baselines. In Figure 6, we show the qualitative comparison between our models and other baselines. The results show that the proposed hierarchical graph generative model HDMapGen (both *coordinate-first* and *topology-first*) generate the highest quality maps and vastly outperform other methods. They capture the typical features of HD maps, including patterns like the overall layouts, the crossroads, parallel lanes, *etc*. Moreover, they are capable of generating maps with different city styles. For the maps generated from HDMapGen with the *independent* model, the problematic crossing of lanes happens frequently, as the model fails to take the dependence of coordinates and topology into account. For PlainGen, we find the model to completely fail for this task, as it is too challenging to generate a graph with a much large number of nodes and edges in this setting, compared to the hierarchical graph. For SketchRNN, we find the model can learn the overall geometric patterns. However, there are a large number of problematic cut-offs or dead-ends occurring in the generated lanes. This is because, during the sequential data generation, it is possible to stop generating a consecutive point at any step and then re-start at any arbitrary location. This property is not an issue for a sketch drawing task. However, for HD map generation, the model needs to guarantee the continuity constraint between generated points.

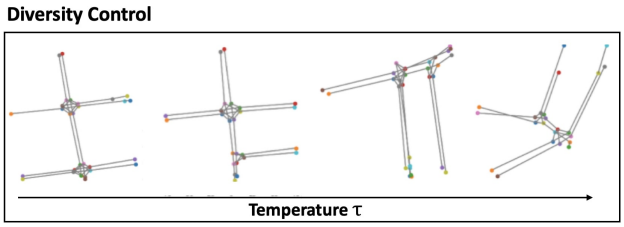

Figure 4: The diversity of the global graph from HDMapGen is improved as temperature τ increases, but the realism suffers.

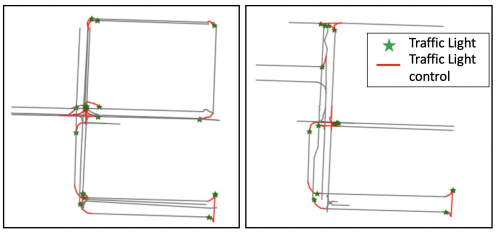

Figure 5: Semantic attributes of traffic lights and local graph of lanes controlled by traffic light generated from HDMapGen.

4.4. Quantitative Analysis

4.4.1 Metrics

We design four metrics to quantify the generation results:

Topology fidelity. We use maximum mean discrepancy (MMD) [17] to quantify the similarity with real maps on graph statistics using Gaussian kernels with the first Wasserstein distance. The topology statistics we apply here is the *degree* distributions and *spectrum* of graph Laplacian.

Geometry fidelity. We use geometry features, the *length*, and *orientation* of lanes to quantify the similarity with real

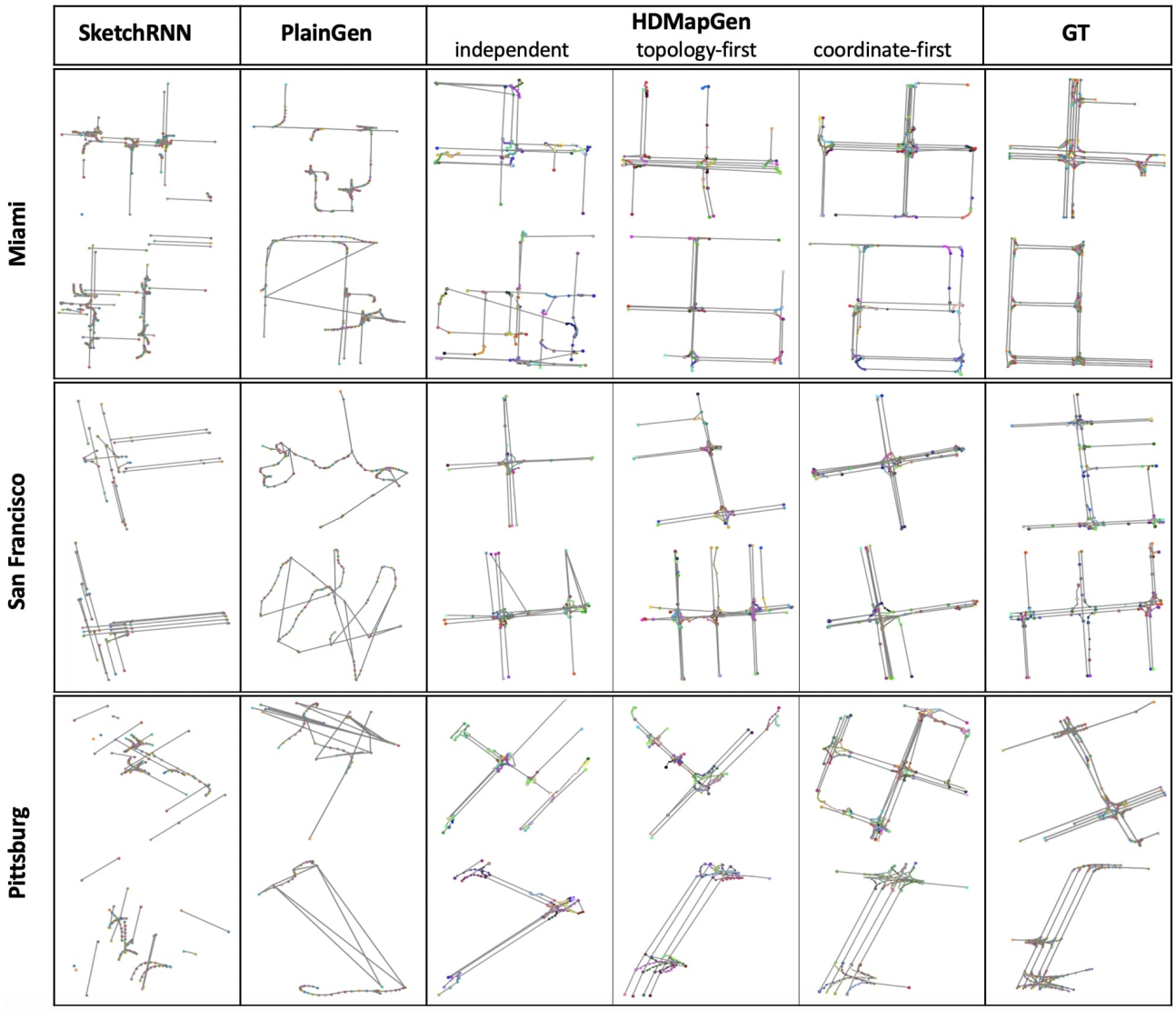

Figure 6: HD Maps with different city styles from hierarchical graph generative models HDMapGen (*coordinate-first*, *topology-first* and *independent*) outperform plain graph generative model PlainGen, and sequence generative model sketchRNN.

maps. Then we compute the Fréchet Distance to measure two normal distributions using these features.

Urban planning. We use the common urban planning features, *connectivity* to represent node degree, node *density* within a region, *reach* as the number of accessible lane within a region to evaluate the transportation plausibility, and *convenience* as the Dijkstra shortest path length for node pairs of our generated maps compared with real maps [10]. We also use Fréchet Distance to measure two normal distributions of these features.

As shown in Table 1, HDMapGen (both *coordinate-first* and *topology-first*) outperform the baselines on most of the metrics. The results are consistent with the qualitative analysis: SketchRNN has a poor performance in topology-related features, such as the degree. Since problematic cut-offs or dead-ends of lanes frequently happen in SketchRNN, so the generated maps have a very different distribution of node degrees from real maps. PlainGen has a better performance in the spectrum of graph Laplacian. As we evaluate topology features by transforming outputs of all models into the plain graph level, which might preserve the intrinsic topology patterns for outputs from PlainGen.

Diversity. We use a metric quantified by Chamfer distance to quantify the map-wise diversity of the global graph generated from HDMapGen. As shown in Table 2, we evaluate the diversity on two levels, one is the novelty compared to real map ground truth, the other is the internal diversity among output samples. It shows that results generated with varied temperatures are novel compared with real maps. For internal diversity, we could see the Chamfer distance drastically increases as the temperature becomes larger.

4.5. Ablation Studies

We further conduct ablation studies on the impact of using different dependence and generation priority on three variants of HDMapGen models *coordinate-first*, *topology-*

Measurement		Urban Planning				Geometry Fidelity		Topology Fidelity	
Metrics		Fréchet Distance						MMD	
Features		Conne.	Densi.	Reach.	Conve.	Len.	Orien.	Deg.	Spec.
Methods		10^0	10^1	10^1	10^1	10^{-1}	10^1	10^{-1}	10^{-1}
SketchRNN	conditional	0.50	21.20	13.43	49.4	2.04	1.77	1.03	4.94
	unconditional	0.44	41.12	29.83	49.5	1.64	1.22	0.83	4.74
PlainGen	topology-first	0.54	5.18	14.57	22.6	1.85	1.54	0.17	0.31
	coordinate-first	0.39	4.30	4.81	12.0	1.64	0.46	0.12	**0.29**
HDMapGen	independent	0.31	4.38	**4.22**	11.2	**1.49**	0.52	0.06	0.54
	topology-first	0.26	**4.06**	4.47	**9.9**	**1.49**	**0.37**	**0.05**	0.63
	coordinate-first	**0.17**	4.79	4.74	11.8	**1.49**	0.53	0.07	0.90

Table 1: Measurements of urban planning, geometry fidelity and topology fidelity on HDMapGen and baselines PlainGen and SketchRNN. Urban planing (features of *connectivity*, *density*, *reach* and *convenience*) and Geometry (features of the *length* and *orientation* of lanes) are quantified with a Fréchet Distance metric. Topology fidelity (features of node *degree* and *spectrum* of graph Laplacian) is quantified with a MMD metric. For all metrics, lower is better. Results are evaluated on Argoverse dataset.

Temperature		0.1	0.2	0.3	0.4	0.5
Diversity	GT	11.6	11.9	11.1	11.6	**13.0**
Diversity	Output	2.75	4.83	5.83	6.62	**8.34**

Table 2: Diversity quantified by Chamfer distance (scaling by 10^4) for global graph generated from HDMapGen using different temperatures τ. Novelty is compared to ground truth or among outputs internally. Results are evaluated on our in-house dataset.

first and *independence*.

We show the ablation results in Table 3. For *coordinate-first* model, BCE, which quantifies the topology prediction performance, is the best optimized. As after coordinates are generated as a prior, then the topology prediction could be better optimized. Meanwhile, for *topology-first* model, NLL, which quantifies the geometry prediction performance, is the best optimized with the generated topology as prior. In contrast, *independence* model is the worst as it fails to model the dependence of coordinate and topology during generation.

Metrics	coordinate-first	topology-first	independent
NLL	-5.610	**-6.142**	-4.490
BCE	**0.001**	0.050	0.053

Table 3: Negative Log likelihood (NLL) and binary cross-entropy (BCE) on the generated global graph from three variants of HDMapGen including *coordinate-first*, *topology-first* and *independent*. Results are evaluated on Argoverse dataset.

4.6. Scalability Analysis

We further experiment HDMapGen with a FoV of $400m \times 400m$. As shown in Figure 7, HDMapGen consistently achieves promising results for large graph generation, demonstrating its scalability.

4.7. Latency Analysis

We show a latency comparison in Table 4. HDMapGen achieves a speed-up of ten-fold compared to SketchRNN. HDMapGen is much more efficient due to (1) using a smaller number of nodes to represent a global graph, and (2) utilizing GRAN with state-of-the-art time-efficiency for global graph generation and a single-shot decoder for the local graph generation.

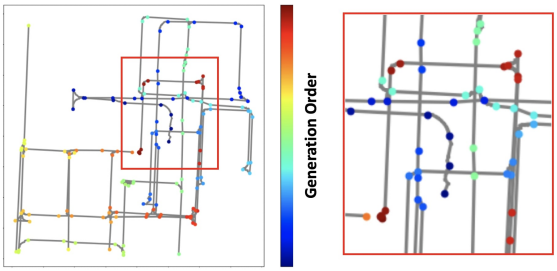

Figure 7: The global node generation order (from blue to red) and the scalability of our hierarchical graph generative model HDMapGen to generate HD maps with a FoV of $400m \times 400m$.

Model	HDMapGen	PlainGen	SketchRNN
Time [s]	**0.20**	0.89	2.28

Table 4: The generation time of an HD map with a FoV of $200m \times 200m$ on Argoverse dataset using HDMapGen, PlainGen, and SketchRNN. HDMapGen is clearly the fastest.

5. Conclusion

In this work, we introduce a novel and challenging task to generate HD maps in a data-driven way. We performed a systematic exploration for autoregressive generative models with different data representations. Our proposed hierarchical graph generative model HDMapGen largely outperforms other baselines. We further demonstrated the advantages of HDMapGen in generation quality, diversity, scalability, and efficiency on real-world datasets.

6. Acknowledgement

We would thank Benjamin Sapp and Tianxing He for insightful comments and suggestions.

References

[1] Esri: Cityengine. https://www.esri.com/en-us/arcgis/products/esri-cityengine.

[2] Daniel G Aliaga, Carlos A Vanegas, and Bedrich Benes. Interactive example-based urban layout synthesis. In *SIGGRAPH Asia*, 2008.

[3] 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 *CVPR*, 2018.

[4] Sven Bauer, Yasamin Alkhorshid, and Gerd Wanielik. Using high-definition maps for precise urban vehicle localization. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 492–497. IEEE, 2016.

[5] Jan Benes, Alexander Wilkie, and Jaroslav Kriványek. Procedural modelling of urban road networks. *Computer Graphics Forum*, 2014.

[6] Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. In *CoRL*, 2019.

[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 Conference on Computer Vision and Pattern Recognition*, pages 8748–8757, 2019.

[8] Anning Chen, Arvind Ramanandan, and Jay A Farrell. High-precision lane-level road map building for vehicle navigation. In *IEEE/ION position, location and navigation symposium*, pages 1035–1042. IEEE, 2010.

[9] Guoning Chen, Gregory Esch, Peter Wonka, Pascal Müller, and Eugene Zhang. Interactive procedural street modeling. *TOG*, 2008.

[10] Hang Chu, Daiqing Li, David Acuna, Amlan Kar, Maria Shugrina, Xinkai Wei, Ming-Yu Liu, Antonio Torralba, and Sanja Fidler. Neural turtle graphics for modeling city road layouts. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4522–4530, 2019.

[11] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[12] Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. Worldbrush - interactive example-based synthesis of procedural virtual worlds. *TOG*, 2015.

[13] Eric Galin, Adrien Peytavie, Éric Guérin, and Bedrich Benes. Authoring hierarchical road networks. *Computer Graphics Forum*, 2011.

[14] Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, and Cordelia Schmid. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11525–11533, 2020.

[15] Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep autoregressive networks. In *International Conference on Machine Learning*, pages 1242–1250. PMLR, 2014.

[16] David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

[17] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks. In *Advances in neural information processing systems*, pages 10701–10711, 2019.

[18] 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 (ICCV)*, October 2019.

[19] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[20] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[21] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. Grass: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017.

[22] Zuoyue Li, Jan Dirk Wegner, and Aurélien Lucchi. Polymapper: Extracting city maps using polygons. *arXiv:1812.01497*, 2018.

[23] Ming Liang, Bin Yang, Rui Hu, Yun Chen, Renjie Liao, Song Feng, and Raquel Urtasun. Learning lane graph representations for motion forecasting. *arXiv preprint arXiv:2007.13732*, 2020.

[24] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, pages 4255–4265, 2019.

[25] Gellért Máttyus, Wenjie Luo, and Raquel Urtasun. Deeproadmapper: Extracting road topology from aerial images. In *ICCV*, 2017.

[26] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas J Guibas. Structurenet: Hierarchical graph networks for 3d shape generation. *arXiv preprint arXiv:1908.00575*, 2019.

[27] Charlie Nash, Yaroslav Ganin, SM Eslami, and Peter W Battaglia. Polygen: An autoregressive generative model of 3d meshes. *arXiv preprint arXiv:2002.10880*, 2020.

[28] G Nishida, I Garcia-Dorado, and D G Aliaga. Example-driven procedural urban roads. *Computer Graphics Forum*, 2015.

[29] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[30] Yoav IH Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308, 2001.

[31] Bidisha Samanta, Abir De, Gourhari Jana, Vicenç Gómez, Pratim Chattaraj, Niloy Ganguly, and Manuel Gomez-Rodriguez. Nevae: A deep generative model for molecular graphs. *Journal of Machine Learning Research*, 21(114):1–33, 2020.

[32] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[33] Jerry O Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Mech, and Vladlen Koltun. Metropolis procedural modeling. *TOG*, 2011.

[34] Sebastian Thrun. Simultaneous localization and mapping. In *Robotics and cognitive approaches to spatial mapping*, pages 13–41. Springer, 2007.

[35] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[36] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Graphgan: Graph representation learning with generative adversarial nets. *arXiv preprint arXiv:1711.08267*, 2017.

[37] Benjamin Watson, Pascal Müller, Oleg Veryovka, Andy Fuller, Peter Wonka, and Chris Sexton. Procedural urban modeling in practice. *IEEE Computer Graphics and Applications*, 28(3):18–26, 2008.

[38] Yong-Liang Yang, Jun Wang, Etienne Vouga, and Peter Wonka. Urban pattern: Layout design by hierarchical domain splitting. *ACM Transactions on Graphics (TOG)*, 32(6):1–12, 2013.

[39] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, 2018.

[40] Hang Zhao, Jiyang Gao, Tian Lan, Chen Sun, Benjamin Sapp, Balakrishnan Varadarajan, Yue Shen, Yi Shen, Yuning Chai, Cordelia Schmid, et al. Tnt: Target-driven trajectory prediction. *arXiv preprint arXiv:2008.08294*, 2020.