General Point Model Pretraining with Autoencoding and Autoregressive

Supplementary Material

6. Implementation Details

6.1. Stage 1: Discrete VAE Pre-training

Architecture: Following [63], our discrete VAE (dVAE) consists of a tokenizer and decoder. The tokenizer module encompasses a robust 4-layer DGCNN [53], while the decoder module integrates a 4-layer DGCNN, followed by the FoldingNet [60]. The comprehensive network architecture of our dVAE is meticulously depicted in Table 7, encompassing essential dimensions such as D_{in} (input feature dimension), D_{out} (output feature dimension), and D_{mid} (hidden layer dimension). Moreover, N_{out} denotes the number of point groups in each layer, and K signifies the number of neighbors involved in the k-Nearest Neighbors (kNN) operation. To further enhance the representation, the FoldingLayer incorporates 2D grids into the input, ultimately generating immersive 3D point clouds.

Structure	Module	D_{in}	D_{out}	K	N_{out}	D_{mid}
dVAE tokenizer	Linear	384	128	-	-	-
	DGCNN	128	256	4	64	-
	DGCNN	256	512	4	64	-
	DGCNN	512	512	4	64	-
	DGCNN	512	1024	4	64	-
	Linear	2304	8192	-	-	-
dVAE decoder	Linear	384	128	-	-	-
	DGCNN	128	256	4	64	-
	DGCNN	256	512	4	64	-
	DGCNN	512	512	4	64	-
	DGCNN	512	1024	4	64	-
	Linear	2304	256	-	-	-
	MLP	256	48	-	-	1024
	FoldingLayer	256	3	-	-	1024

Table 7. Details of our model discrete VAE.

Hyper-parameters: We define the learnable vocabulary size as 8192 in our approach, with each individual 'word' represented as a 384-d vector. The dVAE's optimal performance heavily relies on two critical hyperparameters: α for the Kullback-Leibler (KL) loss term and the temperature τ for the Gumbel-Softmax distribution. We follow [63], initially set α to 0 for the first 18 epochs, approximately 10,000 steps, gradually increasing it to 0.1 over the subsequent 180 epochs, roughly 100,000 steps, employing a cosine schedule. Regarding τ, we adopt a similar decay strategy as described in [41], progressively reducing it from 1 to 0.0625 throughout the initial 180 epochs with a cosine schedule, comprising around 100,000 steps.

Experiment Settings: Table 8 shows the experimental settings for dVAE pre-training in stage 1.

Config	Value
optimizer	AdamW [31]
learning rate	5e-4
weight decay	0.05
learning rate schedule	cosine [30]
warmingup epochs	10
augmentation	RandomSampling
batch size	600
number of points	1024
number of groups	64
group size	32
epochs	300
dataset	ShapeNet

Table 8. Experiment settings for dVAE pre-training.

6.2. Stage 2: GPM Pre-training

Config	Value
optimizer	AdamW
learning rate	5e-4
weight decay	0.05
learning rate schedule	cosine
warmingup epochs	3
augmentation	ScaleAndTranslate
batch size	448
number of points	1024
number of groups	64
input length	129
group size	32
mask ratio	[0.25, 0.45]
mask method	rand mask
epochs	300
dataset	ShapeNet

Table 9. Experiment setting for our GPM pre-training.

Architecture: Adhering to [63], we adopt the standard Transformer architecture as the backbone of our GPM. It comprises a cascade of Transformer blocks [51], each composed of a multi-head self-attention layer and a FeedForward Network (FFN). Within these two layers, we incorporate LayerNorm (LN) to ensure optimal performance and stability.

Hyper-parameters: In our experiments, we configure the transformer architecture with 12 blocks. Each multi-head self-attention layer consists of 6 heads. Additionally, the feature dimension of the transformer layer is set to 384. Following [63], we employ the stochastic depth strategy as proposed in [49], incorporating a dropout rate of 0.1.

Experiment Settings: In our study, we divide the input into two parts: PartA and PartB. The sequence length of PartA is 64, and we prepend a [**CLS**] token at the beginning for downstream task fine-tuning. At the beginning of PartB, we insert an [**SOS**] token to indicate the start of the autoregressive process. We use $\{[\text{SOS}], p_1, ..., p_{n-1}\}$ as PartB and $\{p_1, p2, ..., p_n\}$ as the supervision signal for the autoregressive process. Therefore, the total input sequence length is 129. The detailed experiment setup is shown in Table 9.

6.3. Downstream Tasks

Structure	Module	D_{in}	D_{out}	K	N_{out}	D_{mid}
classification head	MLP	768	N_{cls}	-	-	256
segmentation head	MLP	387	384	-	-	1536
	DGCNN	384	512	4	128	-
	DGCNN	512	384	4	128	-
	DGCNN	384	512	4	256	-
	DGCNN	512	384	4	256	-
	DGCNN	384	512	4	512	-
	DGCNN	512	384	4	512	-
	DGCNN	384	512	4	2048	-
	DGCNN	512	384	4	2048	-

Table 10. Details of our model discrete VAE.

Classification Setups: For our classification task, we employ a two-layer Multi-Layer Perceptron (MLP) with dropout as our classification head. In this process, we extract the output feature of the [**CLS**] token and perform max-pooling on the remaining node's features. These two features are subsequently fused and fed into our classification head. The comprehensive architecture of our classification head is presented in Table 10, where N_{cls} represents the number of classes specific to the dataset under consideration. The details of hyper-parameters of classification task are shown in Table 11.

Config	Value
optimizer	AdamW
learning rate	5e-4
weight decay	0.05
learning rate schedule	cosine
warmingup epochs	10
augmentation	ScaleAndTranslate
batch size	32
number of points	1024
number of groups	64
input length	64
group size	32
epochs	300
dataset	ModelNet [55] & ScanObjectNN [50]

Table 11. Details of object classification fune-tuning.

Config	Value
optimizer	AdamW
learning rate	5e-4
weight decay	0.05
learning rate schedule	cosine
warmingup epochs	10
augmentation	ScaleAndTranslate
batch size	16
number of points	2048
number of groups	128
input length	64
group size	32
epochs	300
dataset	ModelNet [55] & ScanObjectNN [50]

Table 12. Details of segmentation fune-tuning.

Segmentation Setups: We employ an upsampling-propagation strategy that comprises two essential steps: 1) geometry-based feature upsampling and 2) hierarchical feature propagation. These steps enable us to effectively tackle the challenge of dense prediction by incorporating both local and global information in the feature maps.

To capture a comprehensive range of information, we extract features from multiple layers of the transformer architecture. Notably, shallow layers tend to capture low-level details, whereas deeper layers encapsulate higher-level information. To enable upsampling of feature maps at different resolutions, we employ a two-step approach. Firstly, we apply Farthest Point Sampling (FPS) to the original point cloud, generating point clouds at various resolutions. Subsequently, we upsample the feature maps from different layers to match the corresponding resolutions accordingly. This approach allows us to effectively leverage features at different levels of granularity, enhancing the overall representation capacity of the model.

After obtaining feature maps at varying resolutions, we proceed with feature propagation from coarse-grained to

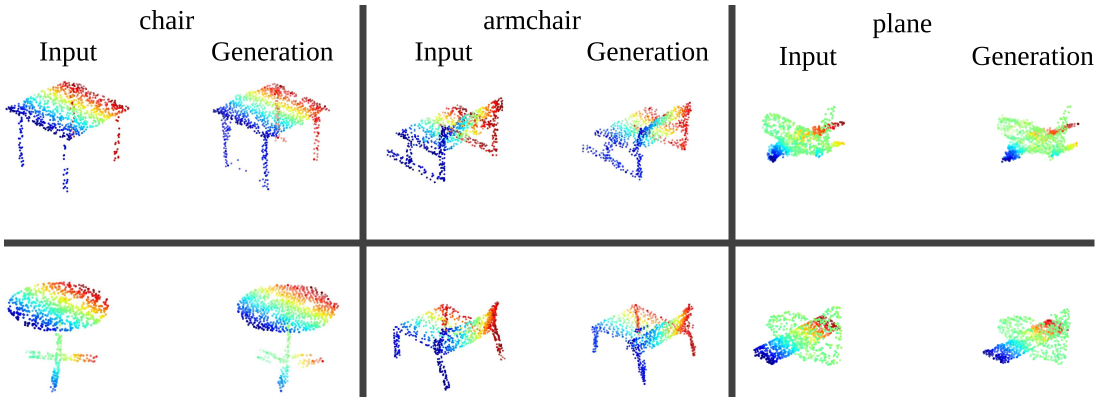

Figure 5. Results on point cloud generation autoregressively. It is evident that we have largely reconstructed the original point cloud.

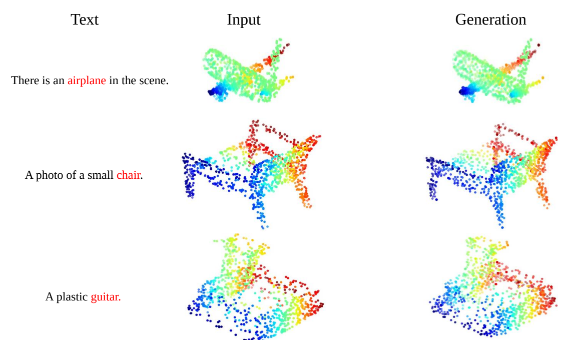

Figure 6. Text-conditional point cloud generation results.

fine-grained feature maps. Subsequently, we employ a lightweight DGCNN to update the feature representation of the points in dense point cloud. This hierarchical feature update process is iteratively performed as the resolution increases, ultimately yielding a dense feature map that can be effectively leveraged for segmentation tasks. For a comprehensive understanding of the segmentation head's architecture and the hyper-parameters in segmentation task, please refer to Table 10 and Table 12.

Unconditional and Conditional Point Generation: We visualize the results of unconditional point cloud generation using GPM and present them in Figure 5. And owing to dividing input into PartA and PartB during pre-training, our model fits to conditional point cloud generation tasks naturally. In this work, we focus on text-conditional point cloud generation tasks. Each text input is associated with three different templates, and we randomly select one template as a condition for point cloud generation. The results of this process are illustrated in Figure 6.

6.4. Additional Ablation Results

In order to ensure fairness and account for the discrepancy in D_{in} compared to the setting in Point-BERT [63], we conduct experiments to examine whether the change of D_{in} will lead to performance enhancements. The experimental results are presented in the Table 13 below.

D_{in}	Acc (ModelNet 1k)	Cls.mIOU	Inst.mIOU
256	93.7	84.22	85.78
384	93.8	84.20	85.80

Table 13. Results on ModelNet dataset classification and segmentation tasks with different D_{in}. It can be observed that the impact of different D_{in} on the results is negligible, to the extent that it can be disregarded.