Point-JEPA: A Joint Embedding Predictive Architecture for Self-Supervised Learning on Point Cloud

Ayumu Saito
Graphics & Spatial Computing Lab
Saint Mary's University, Canada
ayumu.saito@smu.ca

Jiju Poovvancheri
Graphics & Spatial Computing Lab
Saint Mary's University, Canada
jiju.poovvancheri@smu.ca

Abstract

Recent advancements in self-supervised learning in the point cloud domain have demonstrated significant potential. However, these methods often suffer from drawbacks, including lengthy pre-training time, the necessity of reconstruction in the input space, or the necessity of additional modalities. In order to address these issues, we introduce Point-JEPA, a joint embedding predictive architecture designed specifically for point cloud data. To this end, we introduce a sequencer that orders point cloud tokens to efficiently compute and utilize tokens' proximity based on their indices during target and context selection. The sequencer also allows shared computations of the tokens' proximity between context and target selection, further improving the efficiency. Experimentally, our method achieves competitive results with state-of-the-art methods while avoiding the reconstruction in the input space or additional modality.
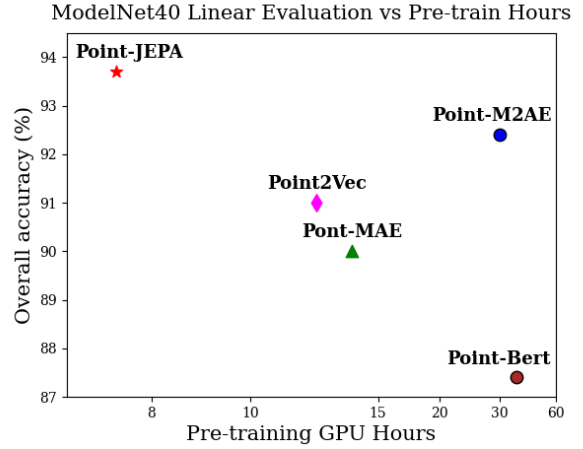
Figure 1. **ModelNet40 Linear Evaluation.** Pre-training time on NVIDIA RTX A5500 and overall accuracy with SVM linear classifier on ModelNet40 [26]. We compare PointJEPA with previous methods utilizing standard Transformer architecture.

1. Introduction

Sef-supervised learning is a paradigm that allows the model to learn a meaningful representation from unlabeled data. This allows the model to utilize a vast amount of unlabeled data and learn a strong representation, leading to advancements in natural language processing [4, 9, 20, 21] and 2D computer vision [2, 5, 8, 11, 12].

Only recently, we have seen the successful applications of self-supervised learning in the point cloud domain [7, 17, 27–29], achieving state-of-the-art results on downstream tasks. However, our initial investigation found that they require a significant amount of pre-training time as shown in Figure 1. This slow pre-training process can pose constraints in scaling to a larger dataset or deeper and more complex models, hindering the key advantage of sel-supervised learning; its capacity to learn a strong representation from a vast amount of data.

The recently proposed Joint-Embedding Predictive Ar-

chitecture [14] and successful implementations of the architecture for pre-training a model [2, 3] show JEPA's ability to learn strong off-the-shelf semantic representations. The idea behind JEPA is to learn a representation by predicting the embedding of the input signal, called *target*, from another compatible input signal, called *context*, with the help of a predictor network. This allows learning in the representation space instead of the input space, leading to efficient learning.

Inspired by I-JEPA [2], we aim to apply Joint-Embedding Predictive Architecture in the point cloud domain. While this application introduces a promising direction for self-supervised learning in the point cloud domain, it also presents a unique challenge. Unlike an image, a point cloud is a set of points with no specific order. Regardless of the permutations applied to the set, the set presents represents the same object. This implies that the model consuming N points needs to be invariant to $N!$ permutations

of the order of the input data [18]. This unordered nature of the point cloud data makes the context and target selection of the data difficult especially if we aim to select spatially contiguous patches similar to I-JEPA [2]. In order to overcome this challenge while utilizing the full potential of Joint-Embedding Predictive Architecture for efficiency, we introduce Point-JEPA. To this end, we introduce a sequencer that orders the input sequence such that elements keep the spatial proximity when they are adjacent in the data. This allows the shared computation for proximity between the target and context, allowing efficiency while achieving a state of the performance with linear evaluation on ModelNet40 [26] as shown in Figure 1. Our method also achieves promising results in other downstream tasks showing the strong and transferable learned representation.

2. Point-JEPA Architecture

Our aim is to bring Joint-Embedding Predictive Architecture to the point cloud domain while measuring the efficient implementation. The overall framework, as shown in Figure 2, first converts the point cloud to a set of tokens, then the sequencer orders the tokens based on the spatial proximity, and Joint-Embedding Predictive Architecture is applied to the ordered tokens. We utilize a small PointNet [18] architecture for encoding the grouped points and standard Transformer [25] architecture for the context and target encoder as well as the predictor. It is important to note that our JEPA architecture operates on the token instead of patches in order to share the point encoder network between context and target encoder for efficiency similar to Point2Vec [28].

Point Cloud Tokenizer.   Similar to previous studies utilizing standard Transformer architecture on point cloud objects [7, 17, 27, 28], we adopt a point cloud tokenizer that embeds groups of points. Given a point cloud object, c center points are first sampled using the farthest point sampling [19]. Then using the k-nearest neighbors algorithm, we sample k closest points to each of c center points. These point patches are then normalized by subtracting the center point coordinate from the coordinates of the points in the patches. This allows the separation between local structural information and the positional information of the patches. In order to embed the local point patches, we utilize a small PointNet [18] architecture. This ensures that the patch embedding, or token, remains invariant to any permutations of data feeding order of points within the patch. Specifically, this PointNet contains two sets of a shared MLP and a max-pooling layer. First, a shared MLP maps each point into a feature vector. Then, we apply max-pooling to these vectors and concatenate the result back to the original feature vector. Another shared MLP then processes these concatenated vectors and a max-pooling layer is applied to produce tokens.
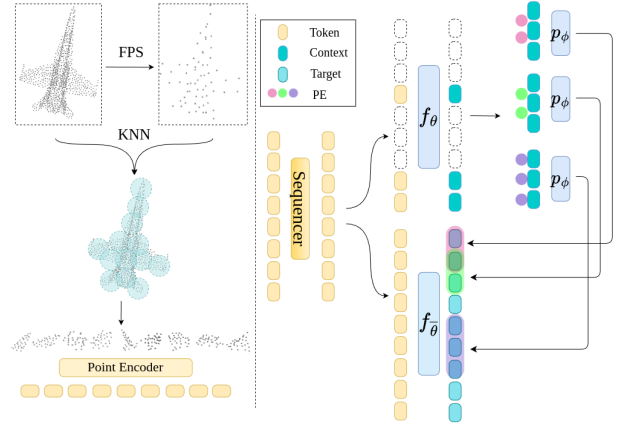

Figure 2. **Point-JEPA.** The tokenization of point cloud patches is shown on the left and the joint embedding architecture is shown on the right.

Sequencer.   Because of the previously observed advantages of spatial proximity in patches for targets and context, referred to as a block in I-JEPA [2], we aim to sample tokens that are spatially close to each other. As previously mentioned, point cloud data is permutation invariant to data feeding order, which implies that even if the indices of tokens are sequential, they might not be spatially adjacent. Moreover, our method involves selecting spatially contiguous M blocks of encoded embedding vectors as the target while ensuring that the context does not include the tokens corresponding to these embedding vectors (details in the next paragraph). In order to overcome these challenges, we introduce a sequencer that is applied after the tokenization of the points. This sequencer orders tokens based on their associated center points. The process begins with a chosen center point and its associated token. In each subsequent step, the center point closest to the one previously chosen and its associated token are selected. This is iterated until the sequencer visits all of the center points. The resulting arrangement of tokens is in a sequence where contiguous elements are also spatially contiguous in most cases. This allows the shared computation of spatial proximity between context and target selection. At the same time, this also allows simpler implementation for context and target selection. It is worth noting, however, that selecting two adjacent token indices does not always guarantee spatial proximity; there might be a gap between them. While this is true, the experiment results show that this iterative ordering is effective enough in our JEPA architecture.

Context and Target.   Targets in Point-JEPA can be considered patch-level representations of the point cloud object, which the predictor aims to predict. As illustrated in Figure 2, the target encoder initially encodes the tokens con-
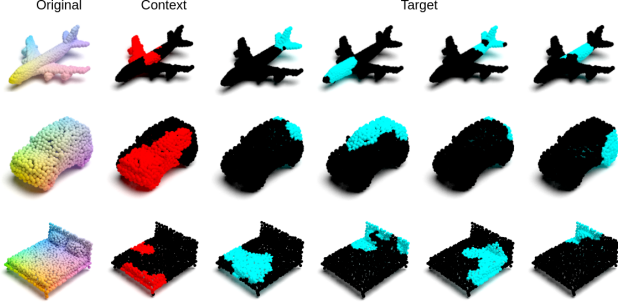
Figure 3. **Context and Targets.** We visualize the corresponding grouped points of context and target blocks. Here, we use (0.15, 0.2) for the target selection ratio and (0.4, 0.75) for the context selection ratio.

ventionally, and we randomly select M possibly overlapping target blocks, which are sets of adjacent embedding vectors. We denote these sets as target blocks. Context, on the other hand, is the representation of the point cloud object which is passed to the predictor to facilitate the reconstruction of target blocks. In order to avoid trivial learning, we remove the corresponding tokens of the target blocks during the context selection step. Out of this subset of tokens, we select a set of tokens that are adjacent in their indices and denote it as a context block. Because of this, context may be formed from multiple sets of spatially contiguous patches while targets often consist of one "block" of contiguous patches as shown in Figure 3.

Predictor and Loss The task of predictor given targets and context is analogous to the task of supervised prediction. Given a context as input along with a certain condition, it aims to predict the target representations. Here, the condition involves the mask tokens, which are created from shared learned parameters, as well as positional encoding created from center points associated with the targets. Because the predictor's task is to predict the representation produced by the target encoder, the loss can be defined to minimize the disagreement between the predictions and targets. To this end, we utilize Smooth L1 loss to measure the dissimilarity between target embedding and predicted embedding. The parameters of the target encoder, as well as the point tokenizer, are updated via gradient-based optimization, while the parameters of the target encoder are updated via the exponential moving average of the context encoder.

3. Experiments

In this section, we first describe the details of self-supervised pre-training. Following that, we compare the performance of the learned representation to some of the top-performing self-supervised learning methods in the point cloud domain, especially those that utilize the ShapeNet [6] dataset in pre-training. We specifically evaluate the learned representation using linear probing, end-to-end fine-tuning, and a few-shot learning setting.

3.1. Self-Supervised Pre-training

We pre-train our model on training set of ShapeNet [6] following the previous studies utilizing the standard Transformer [25] architecture such as Point-MAE [17], Point-M2AE [29], PointGPT [7], and Point2Vec [28]. The dataset consists of 41952 3D point cloud instances created from synthetic 3D meshes from 55 categories. As previously mentioned, standard Transformer [25] architecture is used for the context and target encoder as well as the predictor. During pre-training, we set the number of center points to 64 and the group size to 32. The point tokenization is applied to the input point cloud containing 1024 points per object. We set the depth of the Transformer in the context and target encoder to 12 with the embedding width of 384 and 6 heads. For the predictor, we use the narrower dimension of 192 following I-JEPA [2]. The depth of the predictor is set to 6, and the number of heads is set to 6.

3.2. Downstream Tasks

In this section, we report the performance of the learned representation on several downstream tasks. Following the previous studies [7, 27–29], we report the overall accuracy as a percentage. Unless specified otherwise, we report the mean accuracy and standard deviation from 10 independent runs, each with a different seed, in order to account for variability across independent runs.

Linear Probing. After pre-training on ShapeNet [6], we evaluate the learned representation via linear probing on ModelNet40 [26]. Specifically, we freeze the learned context encoder and place the SVM classifier on top. To enforce invariance to geometric transformation, we utilize max and mean pooling on the output of the Transformer encoder [17, 28]. Here, we utilize 1024 points for both training and test sets. As shown in Table 1, our method achieves state-of-the-art accuracy, providing +0.8% performance gain, showing the robustness of the learned representation.

End-to-end Fine-Tuning We also investigate the performance of the learned representation via end-to-end fine-tuning. After pre-training, we utilize the context encoder to extract the max and average pooled outputs. These outputs are then processed by a three-layer MLP for classification tasks. This class-specific head as well as the context encoder is fine-tuned end-to-end on ModelNet40 [26] and ScanObjectNN [23]. ModelNet40 consists of 12311 synthetic 3D objects from 40 distinct categories, while ScanObjectNN contains objects from 15 classes, each containing

Table 1. **Linear Evaluation on ModelNet40 [26].** We compare Point-JEPA to self-supervised learning methods pre-trained with point cloud data created from ShapeNet [6].
* signifies that the result for linear evaluation is not available in the original paper. We cite the results from [29, 30].

methods	Overall Accuracy
Latent-GAN [1]	85.7
3D-PointCapsNet [31]	88.9
STRL [13]	90.3
Sauder *et al.* [22]	90.6
Fu *et al.* [10]	91.4
Transformer-OcCo* [27]	89.6
Point-Bert* [27]	87.4
Point-MAE* [17]	90.0
Point-M2AE [29]	92.9
Point-JEPA (Ours)	$\textbf{93.7}_{\pm 0.2}$

2902 unique instances collected by scanning real-world objects. For ModelNet40, we sub-sample 1024 points per object and sample 64 center points with 32 points in each point patch. On the other hand, we utilize all 2048 points for the ScanObjNN dataset and sample 128 center points with 32 nearest neighbors for the grouped points. As shown in Table 2, our method achieves competitive results when compared to other state-of-the-art methods. Especially, in the OBJ-BG variant of the ScanObjNN [23] dataset, which presents a realistic representation of a point cloud that includes both the object and its background, our method achieves an improvement of $+1\%$ over the best-performing method. This shows the learned representation obtained from pre-training with Point-JEPA can easily be transferred to a classification task.

Table 2. **End-to-End Classification.** Overall accuracy on Model-Net40 [26] and ScanObjNN [23] with end-to-end fine-tuning. We specifically compare our methods to the method utilizing standard Transformer architecture pre-trained on ShapeNet)[6] with only point cloud (no additional modality.

	Overall Accuracy		
	ModelNet40		**ScanObjNN**
Method	+Voting	-Voting	OBJ-BG
Point-BERT [27]	93.2	92.7	87.4
Point-MAE [17]	93.8	93.2	90.0
Point-M2AE [29]	94.0	93.4	91.2
Point2Vec [28]	**94.8**	**94.7**	91.2
PointGPT-S [7]	94.0	–	91.6
Point-JEPA (Ours)	$94.1_{\pm 0.1}$	$93.8_{\pm 0.2}$	$\textbf{92.9}_{\pm 0.4}$

Table 3. **Result of Few-Shot classification on ModelNet40 [26].** 10 independent trials are completed under one setting. We report mean and standard deviation over 10 trials.

	Overall Accuracy			
	5-way		10-way	
Method	10-shot	20-shot	10-shot	20-shot
Point-BERT [27]	$94.6_{\pm 3.1}$	$96.3_{\pm 2.7}$	$91.0_{\pm 5.4}$	$92.7_{\pm 5.1}$
Point-MAE [17]	$96.3_{\pm 2.5}$	$97.8_{\pm 1.8}$	$92.6_{\pm 4.1}$	$95.0_{\pm 3.0}$
Point-M2AE [29]	$96.8_{\pm 1.8}$	$98.3_{\pm 1.4}$	$92.3_{\pm 4.5}$	$95.0_{\pm 3.0}$
Point2Vec [28]	$97.0_{\pm 2.8}$	$98.7_{\pm 1.2}$	$93.9_{\pm 4.1}$	$95.8_{\pm 3.1}$
PointGPT-S [7]	$96.8_{\pm 2.0}$	$98.6_{\pm 1.1}$	$92.6_{\pm 4.6}$	$95.2_{\pm 3.4}$
Point-JEPA (Ours)	$\textbf{97.4}_{\pm 2.2}$	$\textbf{99.2}_{\pm 0.8}$	$\textbf{95.0}_{\pm 3.6}$	$\textbf{96.4}_{\pm 2.7}$

Few-Shot Learning We conduct few-shot learning experiments on Modelnet40 [26]. Experiments are done in m-way, n-shot setting as shown in Table 3. Specifically, we randomly sample n instances of m classes for training. We select 20 instances of m support classes for evaluation. Under one setting, we run 10 independent runs under a fixed random seed on 10 different folds of the dataset, and we report the mean and standard deviation of overall accuracy. As shown in Table 3, our method exceeds the performance of the current state-of-the-art in all settings. Our method yields a $+1.1\%$ improvement in the most difficult 10-way 10-shot setting, showing the robustness of the learned representation of Point-JEPA, especially in the low-data regime.

4. Conclusion

This work introduced Point-JEPA, a joint embedding predictive architecture applied to point cloud objects. In order to efficiently select targets and context blocks even under the invariance property of point cloud data, we introduced a sequencer, which orders the center points and their corresponding tokens by iteratively selecting the next closest center point. This eliminates the necessity of computing spatial proximity between every pair of tokens or embedding vectors when sampling the targets and context. Our method experimentally achieves state-of-the-art performance in several downstream tasks, showing a strong learned representation. Specifically, Point-JEPA excels in few-shot learning as well as linear evaluation, making the method highly useful when there is a large amount of unlabeled data and a limited amount of labeled data. It is also worth noting that Point-JEPA converges much faster during pre-training, offering a more efficient pre-training alternative in the point cloud domain.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas J. Guibas. Representation learning and adversarial

generation of 3d point clouds. *CoRR*, abs/1707.02392, 2017. 4

[2] Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture, 2023. 1, 2, 3, 7

[3] Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mido Assran, and Nicolas Ballas. V-JEPA: Latent video prediction for visual representation learning, 2024. 1

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. 1

[5] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. *CoRR*, abs/2104.14294, 2021. 1

[6] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 3, 4, 7

[7] Guangyan Chen, Meiling Wang, Yi Yang, Kai Yu, Li Yuan, and Yufeng Yue. Pointgpt: Auto-regressively generative pre-training from point clouds, 2023. 1, 2, 3, 4, 7

[8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1597–1607. PMLR, 2020. 1

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. 1

[10] Kexue Fu, Peng Gao, Renrui Zhang, Hongsheng Li, Yu Qiao, and Manning Wang. Distillation with contrast is all you need for self-supervised point cloud representation learning, 2022. 4

[11] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. *CoRR*, abs/2006.07733, 2020. 1

[12] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross B. Girshick. Masked autoencoders are scalable vision learners. *CoRR*, abs/2111.06377, 2021. 1

[13] Siyuan Huang, Yichen Xie, Song-Chun Zhu, and Yixin Zhu. Spatio-temporal self-supervised representation learning for 3d point clouds, 2021. 4

[14] Yann LeCun. A path towards autonomous machine intelligence, 2022. 1

[15] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. 7

[16] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. 7

[17] Yatian Pang, Wenxiao Wang, Francis E. H. Tay, Wei Liu, Yonghong Tian, and Li Yuan. Masked autoencoders for point cloud self-supervised learning, 2022. 1, 2, 3, 4, 7

[18] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017. 2

[19] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017. 2, 7

[20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 1

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019. 1

[22] Jonathan Sauder and Bjarne Sievers. Context prediction for unsupervised deep learning on point clouds. *CoRR*, abs/1901.08396, 2019. 4

[23] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. *CoRR*, abs/1908.04616, 2019. 3, 4

[24] Laurens van der Maaten and Geoffrey E. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9: 2579–2605, 2008. 9, 10

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. 2, 3

[26] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015. 1, 2, 3, 4, 7, 9, 10

[27] Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. Point-bert: Pre-training 3d point cloud transformers with masked point modeling. *CoRR*, abs/2111.14819, 2021. 1, 2, 3, 4, 7

[28] Karim Abou Zeid, Jonas Schult, Alexander Hermans, and Bastian Leibe. Point2vec for self-supervised representation learning on point clouds, 2023. 2, 3, 4, 7

[29] Renrui Zhang, Ziyu Guo, Rongyao Fang, Bin Zhao, Dong Wang, Yu Qiao, Hongsheng Li, and Peng Gao. Point-

m2ae: Multi-scale masked autoencoders for hierarchical point cloud pre-training, 2022. 1, 3, 4, 7

[30] Renrui Zhang, Liuhui Wang, Yu Qiao, Peng Gao, and Hongsheng Li. Learning 3d representations from 2d pre-trained models via image-to-point masked autoencoders, 2022. 4

[31] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3d point-capsule networks. *CoRR*, abs/1812.10775, 2018. 4

Point-JEPA: A Joint Embedding Predictive Architecture for Self-Supervised Learning on Point Cloud

Supplementary Material

A. Further Pre-training Details

Optimization. We utilize AdamW [16] optimizer with cosine learning decay [15]. Starting from learning rate of 10^{-5}, we increase it to 10^{-3} in the first 30 epochs and decay it to 10^{-6}. The batch size for pre-training is set to 512, and the beta value for Smooth L1 loss is set to 2, similar to Point2Vec [28]. The target encoder and context encoder initially have identical parameters. The context encoder's parameters are updated via backpropagation, while the target encoder's parameters are updated using the exponential moving average of the context encoder parameters, that is $\overline{\theta}_{t+1} \leftarrow \tau\overline{\theta}_t + (1-\tau)\theta_t$ where $\tau \in [0,1]$ denotes the decay rate, $\overline{\theta}$ denotes the parameter of the context encoder, and $\overline{\theta}$ denotes the parameter of the target encoder. We gradually increase the decay rate of the exponential moving average from 0.995 to 1.0 during pre-training.

Masking and Ordering. To determine the sequence of tokens, we utilize the iterative ordering of associated center points, as previously mentioned. We chose the starting point in this sequence with the lowest sum of its coordinates. This method allows the sequence to start from a point on the outer edge of the object rather than from a point within the object's interior. This consistency in selecting the initial point is experimentally shown to deliver a slightly better-learned representation than taking the first available index (See Sec. C for the details).

For masking, we define a range of ratios with both upper and lower limits similar to I-JEPA [2]. To start with, we clarify that the term "block" refers to a sequence of tokens and their corresponding encoded embedding vectors that are contiguous in their indices. Because of the sequencing process applied before the target and context selection, most contiguous tokens and encoded embedding vectors are also spatially contiguous. For the target, we randomly select 4 blocks of embedding vectors from within the 0.15 to 0.2 range. We then remove the corresponding tokens of encoded embedding vectors that have already been chosen as targets for further selection. Following this, we sample tokens within the ratio range of 0.4 to 0.75 out of the available tokens. Because some of the tokens are not available for context selection, we note that context block usually consists of multiple sets of tokens that are spatially contiguous.

B. Further Results

Following previous studies [7, 17, 27–29], we report the performance of Point-JEPA in part segmentation task. Here, we utilize the ShapeNetPart [6] dataset, consisting of 16881 objects from 16 categories. We utilize the identical architecture employed in Point2Vec [28] for this task. Specifically, we take the embeddings from 4th, 8th, and 12th Transformer block and take the average of them. Then, we apply mean and average pooling to this averaged output. The max and mean pooled embedding along with a one-hot encoded class label of an object is used as a global feature vector for the object. The original averaged output is also up-sampled using the PontNet++ [19] feature propagation layer to create a feature vector for each point. Then, each feature vector is concatenated with the global feature vector. A shared MLP is utilized on this concatenated vector to predict the segmentation label for the given point. Although the Point-JEPA shows competitive results as shown in Table 4, its performance is slightly worse than the state-of-the-art methods. This, coupled with the fact that its learned presentation has shown to be superior in classification tasks suggests that Point-JEPA's learned representation emphasizes global features over local features.

Table 4. **Part Segmentation on ShapeNetPart [6]**. mIoU$_C$ is the mean IoU for all part categories, and mIoU$_I$ is the mean IoU for all instances.

Method	mIoU$_C$	mIoU$_I$
Transformer-OcCo [27]	83.4	85.1
Point-Bert [27]	84.1	85.6
Point-MAE [17]	84.1	86.1
Point-M2AE [29]	**84.9**	**86.5**
Point2Vec [28]	84.6	86.3
PointGPT-S [7]	84.1	86.2
Point-JEPA (Ours)	83.9 ± 0.1	85.8 ± 0.1

C. Ablation

In this section, we conducted thorough ablation studies to understand the effect of moving parts of Point-JEPA. We pre-train Point-JEPA on the ShapeNet [6] dataset under various settings and evaluate the learned representation with linear probing on the ModelNet40 [26] dataset.

Masking Strategy. We investigate the impact of the masking type on the performance. We consider single-block masking and multi-block masking. For single-block masking strategies, we consider random masking and contiguous masking. For random masking, we randomly select the 60% of indices out of all encoded embedding vectors. Similarly, for contiguous masking, embedding vectors that are spatially contiguous are selected. In this setting, all tokens not corresponding to the selected target blocks are used as context (denoted as rest). On the other hand, in the muli-block masking setting, we sample multiple possibly overlapping spatially contiguous embedding vectors as targets, and we remove the corresponding tokens already utilized for targets during context selection. In this setting, we set the ratio range of 0.15 to 0.2 for targets, while we set the ratio range of 0.4 to 0.75 for context. As shown in Table 10, the single-block masking achieves sub-optimal performance regardless of the spatial contiguity of the target embedding. It shows that our method learns stronger representation by utilizing a smaller amount of targets with a larger frequency.

Ratio of Targets. We change the ratio of the selected embedding vectors for the target selection while keeping the number of target blocks and the ratio of context tokens fixed. As shown in Table 5, the performance increases when you increase the ratio to a certain point. However, beyond this point, further increasing the ratio results in decreased performance. This implies that Point-JEPA does not require a large size for the target blocks and benefits from a sufficient amount of available tokens for context selection.

Targets		Context	
Ratio	Freq.	Ratio	**Modelnet40 Linear**
(0.1, 0.2)	4	(0.85, 1.0)	93.0
(0.15, 0.2)	4	(0.85, 1.0)	93.3
(0.2, 0.25)	4	(0.85, 1.0)	93.2
(0.25, 0.3)	4	(0.85, 1.0)	92.4
(0.3, 0.35)	4	(0.85, 1.0)	90.5
(0.35, 0.4)	4	(0.85, 1.0)	84.6

Table 5. **Ratio Range for Target.** The ratio of encoded embedding vectors selected for each target.

Ratio of Context. In this study, we change the ratio of tokens selected for context encoding while keeping the number of targets and the ratio range for targets fixed. As shown in Table 6, having a relatively large difference between the lower and upper bound of the ratio can improve performance. In other words, Point-JEPA learns a better representation when the number of selected context tokens varies more between training iterations. Additionally, when the

upper bound of the ratio is somewhat constrained, we see increased performance.

Targets		Context	
Ratio	Freq.	Ratio	**Modelnet40 Linear**
(0.15, 0.2)	4	(0.85, 1.0)	93.1
(0.15, 0.2)	4	(0.75, 1.0)	92.8
(0.15, 0.2)	4	(0.65, 1.0)	93.4
(0.15, 0.2)	4	(0.45, 1.0)	93.6
(0.15, 0.2)	4	(0.6, 0.75)	93.4
(0.15, 0.2)	4	(0.5, 0.75)	93.1
(0.15, 0.2)	4	(0.4, 0.75)	93.7

Table 6. **Ratio Range for Context.** The ratio of tokens selected for context encoding.

Number of Target Blocks. We will also consider the effect of the number of blocks chosen for targets on the performance of the learned representation while we keep the ratio for targets and context fixed. As shown in Table 7, the performance increases as we increase the number of targets. However, the performance decreases as you increase the number of target blocks after a specific frequency. Similarly to what we observed in Section C, we note that our method benefits from having a sufficient amount of tokens available for context encoding.

Targets		Context	
Ratio	Freq.	Ratio	**Modelnet40 Linear**
(0.15, 0.2)	1	(0.85, 1.0)	93.0
(0.15, 0.2)	2	(0.85, 1.0)	93.5
(0.15, 0.2)	3	(0.4, 0.75)	93.4
(0.15, 0.2)	4	(0.4, 0.75)	93.7
(0.15, 0.2)	5	(0.4, 0.75)	93.4
(0.15, 0.2)	6	(0.4, 0.75)	93.2

Table 7. **Number of Target blocks.** We change the number (frequency) of target blocks while keeping the other components fixed.

Predictor Depth	**Modelnet40 Linear**
2	92.5
3	92.8
4	93.2
5	93.4
6	93.7

Table 8. **Predictor Depth.** Predictor depth and its effect on learned representation.

8

Predictor Depth We also study the effect of the predictor's depth on the learned representation. To this end, we vary the predictor depth and observe its effect on the linear evaluation accuracy. As shown in Table 8, Point-JEPA benefits from a deeper predictor.

Initial Point for Sequencer In this section, we study the importance of the initial center point in the sequencer. We compare a sequencer that takes the initial index of the center point against another sequencer that takes the minimum sum of the coordinates as the initial point.

Sequencer Initial Point	Modelnet40 Linear
Minimum index	92.7
Minimum Coordinate Sum	93.7

Table 9. **Initial Point For Sequencer.** The initial selected point in the sequencer and its effect on the learned representation.

As previously mentioned, selecting the first center point presented in the sequencer does not guarantee the selection from the edge of the object. On the other hand, selecting the point with minimum coordinate sum at least guarantees that the starting center point in the sequencer lies near the edge of the object. As shown in Table 9, taking the point near the edge of the object helps with learning stronger representation.

Learned Representation In order to qualitatively analyze the learned representation, we reduce the dimension of the learned representation by utilizing t-SNE [24]. Specifically, we introduce max and mean pooling on the output of the context encoder, similar to the classification setup, and apply t-SNE on the pooled embedding. We visualize the learned representation on ModelNet40 [26] with no fine-tuning on the dataset. In other words, the context encoder has not been trained with this data. Despite this, our context encoder produces discriminative features as shown in Figure 4, showing the robustness of the learned representation.

Targets			Context		
Strategy	Ratio	Freq.	Strategy	Ratio	**Modelnet40 Linear**
random	(0.6, 0.6)	1	rest	–	92.5
contiguous	(0.6, 0.6)	1	rest	–	92.3
contiguous	(0.15, 0.2)	4	contiguous	(0.4, 0.75)	93.7

Table 10. **Masking Strategies.** Here, we apply multi-block and single-block masking strategies and see their effect on the learned representation.
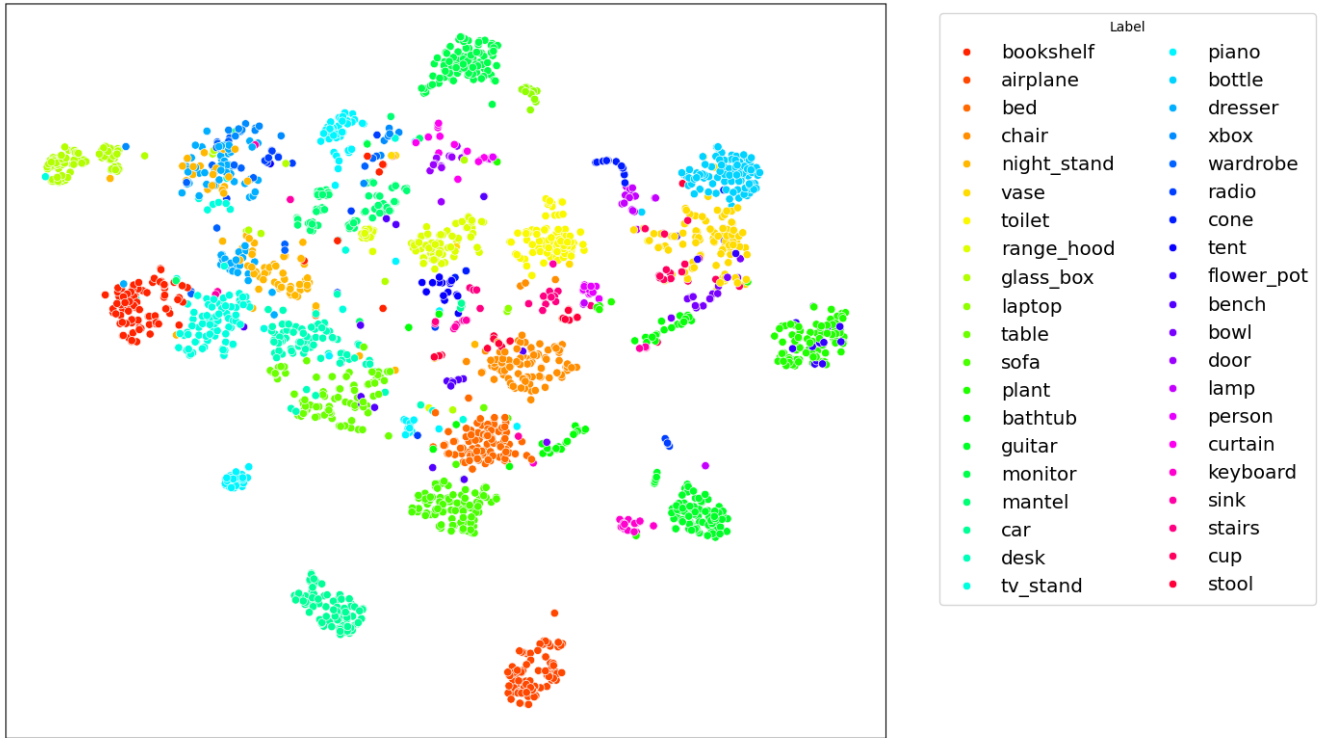

Figure 4. **Embedding Visualization on ModelNet40[26].** We visualize the context encoder's learned representation with t-SNE [24].