Project Report

Classify Unseen Objects

Jakob Kohlhas
Project Partner: Niklas Fraißl

Matriculation Number: 12208214
Study Code: 066 932
E-mail: e12208214@student.tuwien.ac.at

Advisor: Projektass. Mag.rer.soc.oec. Stefan Ohrhallinger, PhD

April 2026

1 Introduction

1.1 Problem Statement & Motivation

Understanding the composition of indoor 3D scenes is a fundamental challenge in computer vision, with applications in robotics, augmented reality and autonomous navigation among others. Typical approaches to object detection and classification in such environments rely on large amounts of labelled training data, which is costly and time-consuming to produce — particularly for 3D point clouds, where manual annotation requires per-point or per-region labelling across complex scenes. Unsupervised and weakly supervised methods offer a promising alternative by reducing or eliminating the need for ground truth labels during training. This project explores such a self-supervised approach by combining unsupervised instance detection with self-supervised feature extraction and automatic clustering to discover and group similar objects in indoor point cloud scenes. The resulting pipeline is designed to be modular, allowing individual components to be replaced or extended.

1.2 Report Scope & Structure

The project was worked on in collaboration with Niklas Fraißl, with each partner focusing on different aspects of the pipeline. This report covers the full system architecture and provides a detailed discussion of the feature extraction and clustering stages (Task II), while also giving a high-level overview of the instance detection stage (Task I). A more detailed discussion of Task I can be found in Niklas Fraißl's report. Section 2 discusses related work and motivates the selection of methods for each pipeline stage. Section 3 introduces the datasets used for development and evaluation. Section 4 describes the overall system architecture, project structure and configuration. Sections 5 and 6 cover the two main pipeline stages — instance detection and feature extraction with clustering — in technical detail. Section 7 reflects on lessons learned and concludes the report.

2 Related Work & Method Selection

2.1 Instance Detection Approaches

Unsupervised and zero-shot approaches have gained popularity in 3D object detection. UnScene3D [1] is a fully unsupervised method that generates pseudo-masks by applying Normalized Cut on self-supervised features derived from 2D DINO and 3D Contrastive Scene Contexts, followed by iterative self-training to refine the segmentation model. On ScanNet, it achieves an average precision (AP) of 15.9, representing a significant improvement over prior unsupervised baselines. SAI3D [2] is a zero-shot approach that leverages the 2D Segment Anything Model (SAM) to produce masks on posed RGB frames, which are then projected into 3D space using depth maps and camera geometry, and merged through progressive region growing with relaxing affinity thresholds. SAI3D reports an AP of 30.8 on ScanNet, outperforming UnScene3D while requiring no training on the target dataset. Both methods were published at CVPR 2024 and were selected for this pipeline to provide two complementary detection strategies. Additionally, the pipeline supports loading ground truth instances directly, which can serve as a reference during development.

2.2 Feature Extraction Approaches

Extracting meaningful feature representations from 3D point clouds is essential for grouping similar objects. Available approaches range from classical handcrafted descriptors to learned representations such as KPConv [3], Point Transformer [4], Point Cloud Transformer [5] or Point-BERT [6], which adapts the BERT [7] pre-training paradigm to point clouds through masked point modelling. Point-MAE [8] extends Point-BERT's idea with a masked autoencoder architecture: it partitions a point cloud into patches via farthest point sampling and k-nearest neighbours, masks 60% of patches at random, and trains a transformer encoder-decoder to reconstruct the masked geometry. The reconstruction loss is based on Chamfer Distance. The encoder learns general-purpose features without requiring any class labels, achieving 93.8% classification accuracy on ModelNet40 after fine-tuning. Point-MAE was selected for this pipeline due to its self-supervised nature, which aligns with the unsupervised philosophy of the overall system.

2.3 Clustering Approaches

Once feature vectors have been extracted for each detected instance, a clustering step is required to group similar objects together. Common approaches include k-means and DBSCAN among others. For this pipeline a component-based approach was preferred to keep individual stages interchangeable. Agglomerative hierarchical clustering [9] with complete linkage was chosen, as it does not require specifying the number of clusters in advance. Instead, a distance threshold controls when the merging process stops, which is particularly suited for indoor scenes where the number of object types varies from scene to scene. Cosine distance is used as the similarity metric. After clustering, UMAP [10] is applied for dimensionality reduction and visualization. UMAP constructs a fuzzy k-nearest-neighbour graph in the high-dimensional feature space and optimizes a low-dimensional layout that preserves the topological structure of the data, offering better preservation of global cluster relationships than alternatives such as t-SNE.

3 Datasets

The pipeline relies on two datasets for development and demonstration. ScanNet [11] provides real-world indoor 3D reconstructions captured with RGB-D sensors, consisting of richly annotated meshes with per-vertex colour information, semantic labels from 200 object categories, and instance-level segmentation data stored in aggregation and segmentation JSON files. ShapeNetCore [12] is a large-scale synthetic dataset of clean, isolated 3D object models organized by category. In this project, the ShapeNetCore test split is used to demonstrate and validate Point-MAE's feature extraction capabilities independently of the detection stage.

4 System Architecture

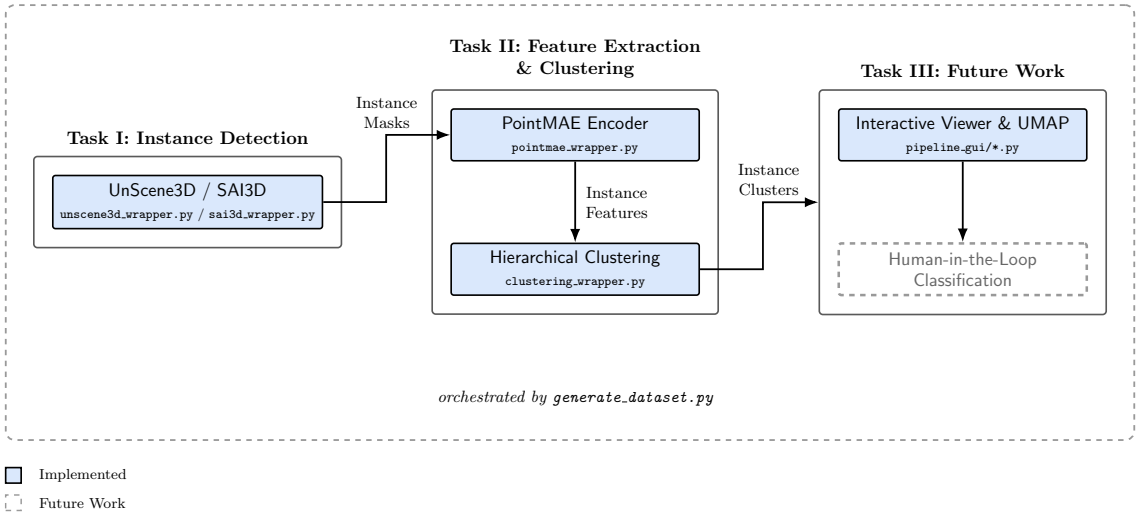

Figure 1: Architecture overview: Voxel-based instance detection, Point-MAE feature extraction, and clustering stages.

4.1 Pipeline Overview

The pipeline processes raw 3D point cloud scenes through three sequential stages. In the first stage, individual object instances are detected and segmented from a ScanNet scene using either UnScene3D (`unscene3d_wrapper.py`), SAI3D (`sai3d_wrapper.py`), or loaded ground truth labels (`data_utils.py`). Each wrapper returns a list of instance dictionaries containing the instance's 3D points, colours, and a binary mask indicating which points in the scene belong to the instance. In the second stage, each instance is passed to Point-MAE (`pointmae_wrapper.py`), which extracts a feature vector representing the geometric properties of the object. In the third stage, the feature vectors of all instances within a scene are clustered using agglomerative hierarchical clustering with cosine distance (`clustering_wrapper.py`), and the resulting cluster assignments are stored alongside the instance data as `.npz` files (e.g. `scene0000_00_instance_0.npz`). The entire pipeline is orchestrated by `generate_dataset.py`, which accepts command-line arguments for the detection, feature extraction, and clustering modes.

4.2 Project Structure & Configuration

The project follows a modular directory structure separating pipeline logic from external dependencies. The `pipeline_models/` directory contains wrapper classes that provide simplified interfaces to the external models, while `pipeline_utils/` holds data loading and point cloud manipulation helpers. The `pipeline_gui/` directory implements an interactive exploration tool using a Model-View-Controller pattern built with PyQt5 and VTK. All external codebases — UnScene3D, Point-MAE, and SAI3D — are included as Git submodules under `external/`. Central

configuration is managed through a single `Config` dataclass in `pipeline_conf/conf.py`, which defines all file paths, model checkpoint locations, CUDA settings, and mode dictionaries that map command-line arguments to internal pipeline parameters. To ensure portability across machines, a `PROJECT_ROOT_VARIABLE_MARKER` system is used: a shell script (`project_root_placeholder_replace.sh`) scans all Python and YAML files for the marker comment and updates the root path variable on the line immediately below it, allowing the entire project to be relocated by running a single command.

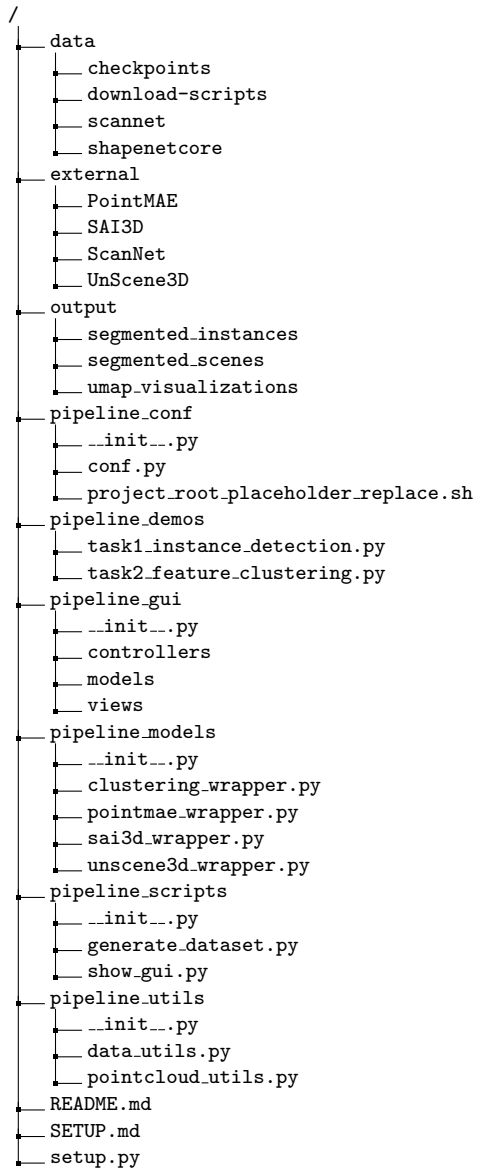

Figure 2: Project directory structure overview.

5 Instance Detection (Task I)

Task I addresses the segmentation of individual object instances from raw ScanNet scenes without any labelled training data. Two complementary methods are integrated as interchangeable branches: UnScene3D, which operates entirely on 3D point cloud data using self-supervised features and iterative pseudo-mask refinement, and SAI3D, which lifts 2D segmentation masks from a foundation model into 3D via depth reprojection. Figure 3 illustrates the shared input and output stages common to both branches.

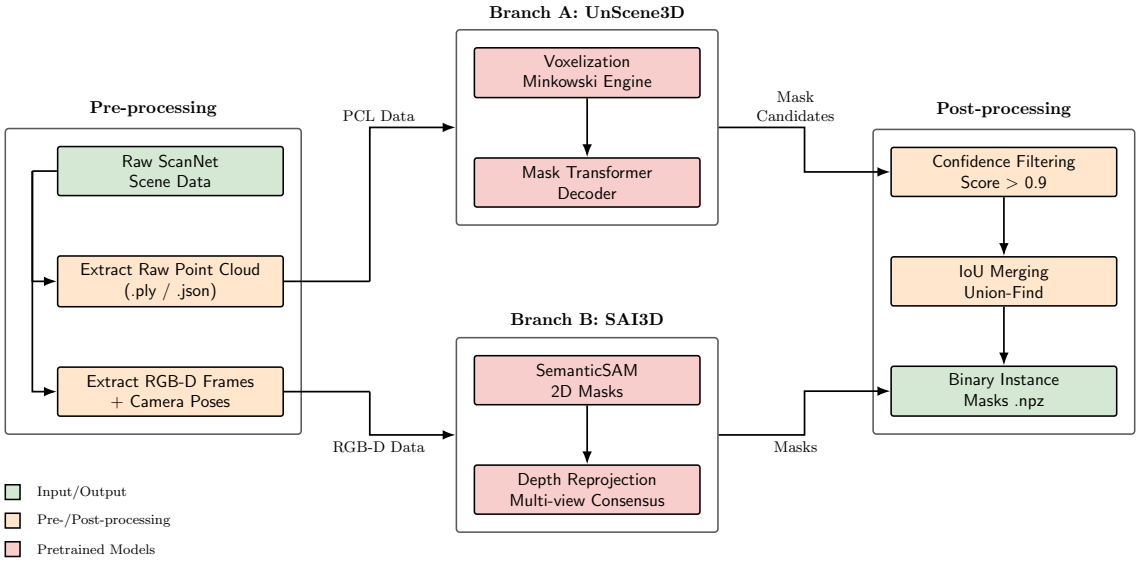

Figure 3: Task I - Instance Detection Pipeline: Modular integration of UnScene3D and SAI3D branches with shared I/O stages.

5.1 UnScene3D

UnScene3D [1] is a fully unsupervised 3D instance segmentation method that operates without any labelled training data. The approach begins by oversegmenting the input point cloud into superpoints based on geometric properties. Self-supervised features are then obtained from two sources: 2D DINO features extracted from posed RGB images and projected onto the 3D points, and 3D Contrastive Scene Context (CSC) features learned directly on the point cloud. These features are used to iteratively partition the scene via Normalized Cut, producing a set of noisy pseudo-masks that approximate individual object instances. A segmentation network is then trained on these pseudo-masks using noise-robust losses, and refined over multiple self-training rounds where the model's own predictions are combined with the initial masks to improve coverage. In the pipeline, the UnScene3D wrapper (`unscene3d_wrapper.py`) normalizes input colours, voxelizes the point cloud at 2 cm resolution, constructs a sparse tensor via MinkowskiEngine, and runs the pretrained model to predict instance masks. Predicted masks are filtered by a confidence threshold of 0.9, and overlapping masks are merged using an IoU-based union-find algorithm.

5.2 SAI3D

SAI3D [2] is a zero-shot 3D instance segmentation method that combines 2D foundation models with 3D geometric reasoning. In a first step, the Segment Anything Model (SAM) is applied to posed RGB images of the scene to generate 2D instance masks. In the pipeline, a Semantic-SAM variant is used, and masks are generated for every fifth frame to balance coverage and computational cost (`sai3d_wrapper.py`). The resulting 2D masks are then projected into 3D space by leveraging the corresponding depth maps, camera intrinsics and extrinsic pose matrices. To determine which 3D points belong to the same object, the point cloud is first divided into small, locally coherent groups (superpoints) of neighbouring points based on geometric similarity. An affinity graph is then built over these superpoints, where the similarity is measured by how consistently they share the same 2D mask labels across multiple camera views. Instances are formed through progressive region growing with a sequence of relaxing thresholds from 0.9 down to 0.5, merging groups that share consistent mask assignments. The projection and merging logic is handled by a custom `SAI3D_Projection_Wrapper` class that extends the original SAI3D codebase. Unlike UnScene3D, SAI3D produces final instance masks directly without requiring additional confidence filtering or IoU-based merging. For a more detailed discussion of the instance detection stage, the reader is referred to Niklas Fraißl's report.

6 Feature Extraction & Clustering (Task II)

Task II takes the binary instance masks produced by Task I and derives a compact geometric representation for each detected object. Each instance point cloud is passed through Point-MAE's transformer encoder to produce a 768-dimensional feature vector, which is then used as input to agglomerative hierarchical clustering to group instances by geometric similarity across the scene. Figure 4 shows the data flow from mask input through encoding to cluster output.

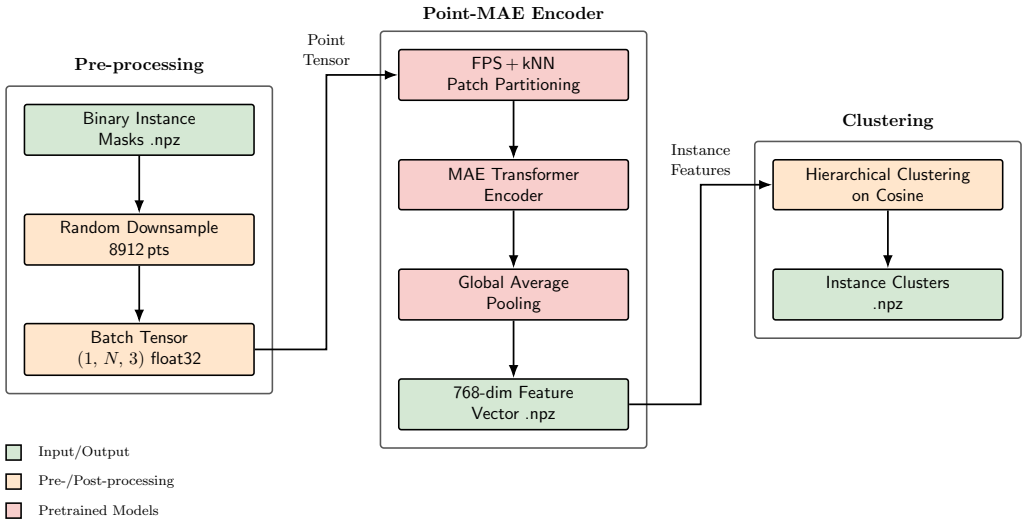

Figure 4: Point-MAE: FPS patch partitioning with masked autoencoder transformer encoder for instance-level feature extraction and post-processing clustering.

6.1 Point-MAE

Point-MAE [8] is a self-supervised masked autoencoder designed for learning representations from 3D point clouds. The input point cloud is first divided into a fixed number of local patches using farthest point sampling to select patch centres, followed by k-nearest-neighbour grouping to assign surrounding points to each centre. Each patch is embedded into a high-dimensional token via a small PointNet-style network. During pre-training, 60% of these tokens are randomly masked, and only the visible tokens are fed into a 12-layer transformer encoder. A lightweight decoder then receives both the encoded visible tokens and learnable mask tokens, and is trained to reconstruct the original coordinates of the masked patches. This forces the encoder to learn meaningful geometric representations of the input, as it must capture enough structural information from the visible patches alone to enable reconstruction. After pre-training on ShapeNet, the encoder is fine-tuned on ModelNet40 for classification, achieving 93.8% accuracy. In the pipeline, `pointmae_wrapper.py` loads the fine-tuned checkpoint and calls the model's forward pass on each detected instance to obtain a feature vector that encodes the object's geometric characteristics.

6.2 UMAP Dimensionality Reduction

UMAP [10] (Uniform Manifold Approximation and Projection) is used in the pipeline to reduce the high-dimensional feature vectors produced by Point-MAE into a two-dimensional representation suitable for visualization. The algorithm first constructs a weighted k-nearest-neighbour graph in the original feature space, where edge weights reflect the local distance distribution around each point. It then finds 2D coordinates for each point such that the neighbourhood structure of the original high-dimensional graph is preserved as closely as possible, using stochastic gradient descent to minimize the cross-entropy between the high-dimensional and low-dimensional representations. Compared to t-SNE, UMAP tends to better preserve global relationships between clusters, meaning that the relative positions of distinct object groups in the 2D plot reflect their actual similarity in feature space. In the pipeline, UMAP serves two roles: in the demo script (`task2_feature_clustering.py`), it visualizes how Point-MAE features of ShapeNetCore objects naturally separate by category, and in the GUI (`data_model.py`), it provides an interactive 2D scatter plot where each point represents a detected instance, coloured by its cluster assignment.

6.3 Hierarchical Clustering

The final stage of Task II groups the extracted feature vectors into clusters representing similar object types. Scikit-learn's `AgglomerativeClustering` is used, which starts by treating each instance as its own cluster and then iteratively merges the two closest clusters until a stopping criterion is met (`clustering_wrapper.py`). Complete linkage (distance of two clusters is defined as the maximal distance between any pair of their members) is used as merging strategy. Cosine distance serves as the distance metric, computed over all pairwise feature vectors via scikit-learn's `pairwise_distances` function. Rather than specifying a fixed number of clusters, a distance threshold of 0.1 is used as the stopping criterion. This is important for the given use case, as the number of distinct object types in an indoor scene is not known in advance and can vary significantly between scenes.

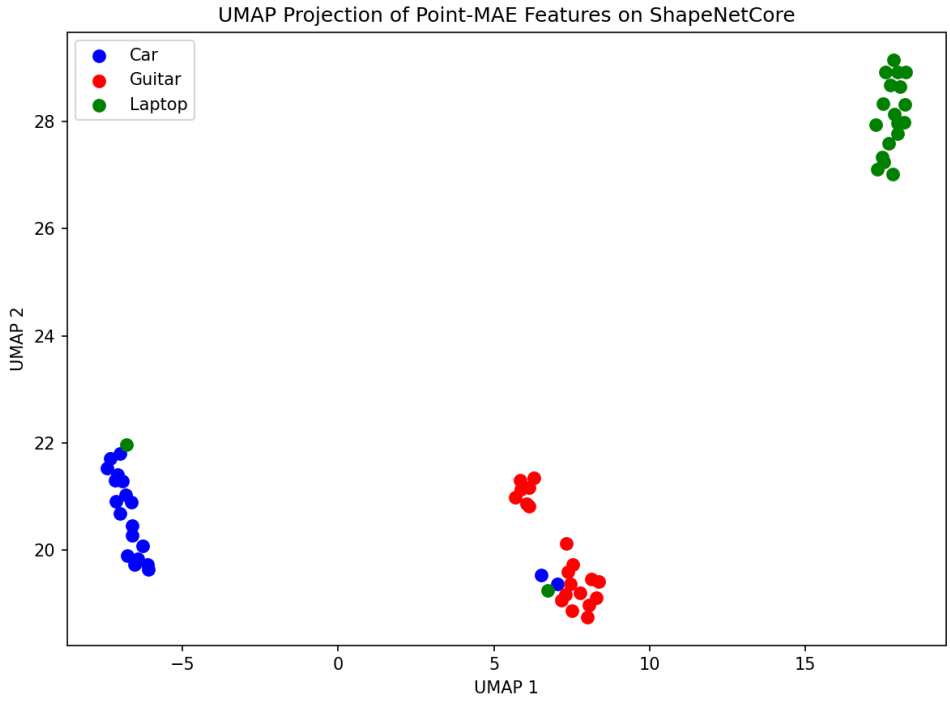

Figure 5: UMAP projection of Point-MAE features extracted from ShapeNetCore objects. The plot demonstrates that geometric features learned by the encoder naturally form distinct clusters corresponding to semantic categories such as Cars, Guitars, and Laptops.

7 Lessons Learned & Conclusion

The development of this pipeline posed several challenges, most of which were related to the integration of external codebases, compatibility issues and the CUDA environment. Installing the required CUDA extensions, in particular MinkowskiEngine as a prerequisite for UnScene3D, was difficult because of strict requirements on specific compiler and CUDA versions. Conflicting dependencies between UnScene3D, SAI3D, and Point-MAE, which often required different versions of PyTorch and associated libraries, required careful management of environment variables and occasional modifications to the original code. The project was initially developed on a private machine, where shared working was enabled via VNC, which in turn caused problems with rendering 3D point clouds through VTK, as VNC does not natively support default OpenGL configuration. The project was later transferred to the university's DISCORD server, where during migration additional compatibility issues had to be solved. Dividing the project work between partners was at times complicated because of the sequential nature of the pipeline stages (Task II depending on Task I outputs etc.).

References

[1] D. Rozenberszki, O. Litany, and A. Dai, "Unscene3d: Unsupervised 3d instance segmentation for indoor scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2024, pp. 19 957–19 967.

[2] Y. Yin, Y. Liu, Y. Xiao, D. Cohen-Or, J. Huang, and B. Chen, "Sai3d: Segment any instance in 3d scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2024, pp. 3292–3302.

[3] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019.

[4] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 16 259–16 268.

[5] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu, "PCT: Point cloud transformer," *Computational Visual Media*, vol. 7, no. 2, pp. 187–199, Jun. 2021.

[6] X. Yu, L. Tang, Y. Rao, T. Huang, J. Zhou, and J. Lu, *Point-bert: Pre-training 3d point cloud transformers with masked point modeling*, 2022. arXiv: 2111.14819 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2111.14819.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. [Online]. Available: https://aclanthology.org/N19-1423/.

[8] Y. Pang, W. Wang, F. E. H. Tay, W. Liu, Y. Tian, and L. Yuan, *Masked autoencoders for point cloud self-supervised learning*, 2022. arXiv: 2203.06604 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2203.06604.

[9] F. Nielsen, "Hierarchical clustering," in *Introduction to HPC with MPI for Data Science*. Cham: Springer International Publishing, 2016, pp. 195–211, ISBN: 978-3-319-21903-5. DOI: 10.1007/978-3-319-21903-5_8. [Online]. Available: https://doi.org/10.1007/978-3-319-21903-5_8.

[10] L. McInnes, J. Healy, and J. Melville, *Umap: Uniform manifold approximation and projection for dimension reduction*, 2020. arXiv: 1802.03426 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1802.03426.

[11] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Niessner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.

[12] A. X. Chang et al., *Shapenet: An information-rich 3d model repository*, 2015. arXiv: 1512.03012 [cs.GR]. [Online]. Available: https://arxiv.org/abs/1512.03012.