Topological Understanding of 3D Transformer-Based Shape Encoders

Author:
Louis Martinez

Télécom Paris

Supervisor:
Maks Ovsjanikov

LIX, Ecole Polytechnique

October 2025

Contents

Abstract

Large transformer-based foundation models have shown remarkable ability to extract expressive representations in a purely self-supervised, data-driven manner, and recent efforts have extended them to 3D shape understanding. In parallel, Topological Data Analysis (TDA), through the lens of persistent homology, offers mathematically grounded tools to characterize the structural information carried by point clouds. However, persistent homology scales poorly with dimension and sample size, limiting its applicability to large datasets. This thesis makes two main contributions. First, we investigate to what extent pretrained 3D transformer models capture structural, non-semantic properties of shapes, using TDA as a principled framework to quantify their topological awareness. Second, building on these insights, we introduce a data-driven proxy for persistent homology: a transformer-based approach that leverages learned representations to predict topological features efficiently. Our results highlight both the limitations of current 3D transformers and the potential of bridging foundation models with TDA to advance topology-aware 3D representation learning.

1 Introduction

Foundation models for 3D data have emerged as a new paradigm for large-scale 3D shape understanding. Inspired by the success of self-supervised learning in natural language and vision, several transformer-based architectures have been adapted to 3D modalities such as point clouds, meshes, and voxels. These models learn expressive representations from unlabeled data and have achieved strong performance across tasks including classification, segmentation, and shape retrieval. The ability to extract rich, semantically meaningful features without manual supervision marks a significant step toward general-purpose 3D understanding.

Despite this progress, a key question remains open: *What do 3D encoders learn about the structural properties of shapes?* While most studies have focused on evaluating semantic or geometric performance, little is known about whether these representations also capture deeper structural information such as topology. This limitation reflects a broader issue in the adaptation of self-supervised paradigms to 3D data. Pretraining strategies originally designed for text or 2D images have been directly transferred to 3D shapes, often without considering their unique mathematical and geometric properties. As a result, current models display restricted generalization. Encoders trained on object-centric datasets struggle to extend to large-scale scenes, whereas scene-level encoders often miss fine-grained shape details. Understanding the nature and limitations of these learned representations is therefore essential to assess whether these models truly capture intrinsic shape structure or simply memorize patterns present in their training distribution.

1.1 Current State of Research

Early self-supervised 3D encoders such as Point-BERT [57] and Point-MAE [36] showed that transformers can model local and global geometric cues in point clouds. They rely on masked modeling or token reconstruction to learn shape embeddings that capture semantic and geometric structure. However, these embeddings are rarely evaluated with respect to the topology of shapes.

Topology describes how a shape is connected, regardless of geometric detail. It captures invariants such as connected components, loops, or voids that remain unchanged under continuous deformations. Understanding whether 3D encoders represent such properties is key to knowing if they learn intrinsic structure rather than only geometry or semantics.

Topological Data Analysis (TDA) provides a mathematical framework to study these properties. Through persistent homology, TDA tracks the evolution of topological features as a scale parameter changes, summarizing them in persistence diagrams. These diagrams have been used in physics, biology, and shape analysis [7] to measure structural complexity, but they remain costly to compute as their complexity grows quickly with sample size and dimension.

Recent studies have tried to bridge TDA and deep learning [8, 50]. Neural networks can learn topological signatures [42], approximate persistence diagrams [55], or embed topological features into latent spaces [34]. Conversely, topological descriptors have been used to regularize or interpret neural representations [5, 16, 46]. Yet, few works have explored whether large pretrained 3D transformers already encode topological information implicitly and how this compares to explicit TDA descriptors. This question motivates the present work.

1.2 Motivations

This work aims to connect representation learning and topological analysis for 3D shapes. Three main motivations guide our study.

understanding whether 3D shape encoders capture topology is central to interpreting what they learn. If persistent topological features can be recovered from their embeddings, this would suggest that large self-supervised models organize information according to structural properties rather than only geometric and semantic ones.

Second, topological data analysis offers a principled framework to quantify structural awareness. By comparing the topological signatures derived from a model's latent space to those computed directly from 3D shapes, we can evaluate whether the learned representations preserve key structural information. This enables a new kind of interpretability analysis grounded in topology.

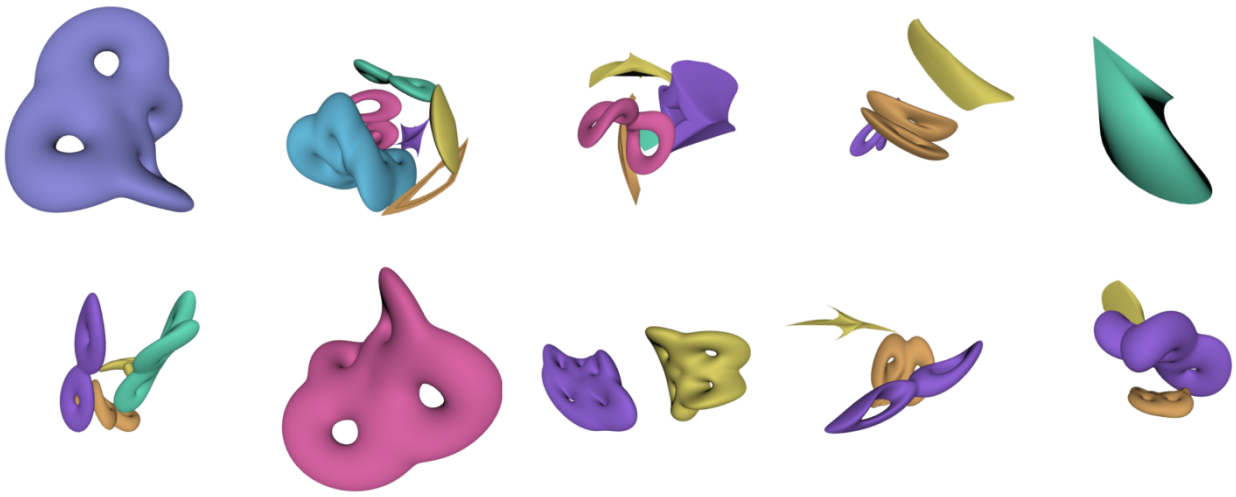


Figure 1: **Random samples from DONUT.** Despite the fact that each sample is generated from a rather small family of shapes, both the topology preserving placement and augmentations allow for a rich variety of shapes, while ensuring topological consistency.

Third, if encoders already contain topological information, their representations could serve as efficient, data-driven proxies for topological computations. Since persistent homology is computationally expensive for large or high-dimensional datasets, learning-based approximations based on pretrained representations could scale topological reasoning to complex real-world 3D data.

We therefore ask:

> *To what extent do current 3D shape encoders capture the topological structure of 3D shapes, and can their learned representations be leveraged as scalable proxies for topological computations?*

Addressing this question contributes both to the interpretability of 3D foundation models and to the development of efficient topology-aware methods for large-scale geometric data.

1.3 Contributions

The contributions of this report are three-fold:

- **A topology-controlled dataset.** We introduce DONUT (Section 2), a scalable dataset of 3D shapes with full control over their topological properties. This dataset enables systematic evaluation of models' sensitivity to structural variations.
- **A quantitative analysis of structural awareness.** We perform comprehensive experiments to measure how state-of-the-art unimodal 3D shape encoders capture structural, non-semantic information (Sections 3 and 4). Using persistent homology as an analytical tool, we assess the topological consistency of learned embeddings.
- **Architectural and training insights.** Based on the findings, we propose directions for improving 3D shape encoders, including architectural modifications and pretraining strategies that encourage topology-aware representations.

2 DONUT: Dataset Of maNifold strUcTures

Understanding how models capture topological properties of 3D shapes is a key step toward disentangling geometric representation learning from semantic categorization. Existing 3D datasets, such as Objaverse [12, 13] or ShapeNet [10], primarily organize data by semantic class or surface geometry, without systematic control of topology. Probing topology-aware representations therefore requires datasets that expose explicit topological labels under controlled geometric variability.

To address this, we introduce DONUT, a scalable dataset of synthetic 3D shapes annotated with precise and balanced topological labels. Unlike prior datasets, DONUT is built from parametric families that guarantee topological correctness while enabling high geometric diversity. This design allows principled evaluation of how models represent intrinsic topological invariants such as connected components and genus.

2.1 Existing datasets

Existing 3D datasets provide only limited support for probing whether models capture topological structure. Broadly, they fall into three categories: combinatorial-only benchmarks, geometric datasets with noisy topology, and synthetic datasets with limited diversity.

Combinatorial benchmarks. The MANTRA dataset (Manifold Triangulation Assemblage) [3] provides combinatorial triangulations of 2D and 3D manifolds, represented as abstract simplicial complexes without embedding in $\mathbb{R}^3$. MANTRA is a valuable testbed for assessing whether graph- or simplicial complex models capture higher-order structures such as Betti numbers $(\beta_0, \beta_1, \beta_2)$. However, since it lacks any geometric realization, MANTRA is not suitable for evaluating models built on geometric 3D representations such as meshes, point clouds, or implicit fields.

Dataset	#Samples	Meshes	Manifold	Balanced annot.
DONUT	30,000	✓	✓	✓
MANTRA [3]	43,100	–	✓	–
ABC [28]	1,000,000+	✓	–	–
Thingi10K [62]	10,000	✓	–	–
EuLearn [15]	3,300	✓	✓	✓

Table 1: **Overview of existing datasets and their capabilities.** We summarize here the main characteristics of existing datasets with topological annotations. Besides EuLearn, all existing datasets with topological annotations come with downsides, discussed in Section 2.1. However, since EuLearn seems to be the most promising dataset, we carried out an extensive analysis to (1) highlight limitations that make it unreliable for further experiments and (2) motivate the use of DONUT (see Appendix A.1). *Note:* The number of samples for MANTRA only takes into account 2-manifolds.

Noisy topology. Large mesh datasets such as Thingi10K [62], Objaverse [12] and ABC [28] contain CAD models and artistic objects, and include coarse topological annotations (number of components, genus). However, these annotations are unreliable for several reasons:

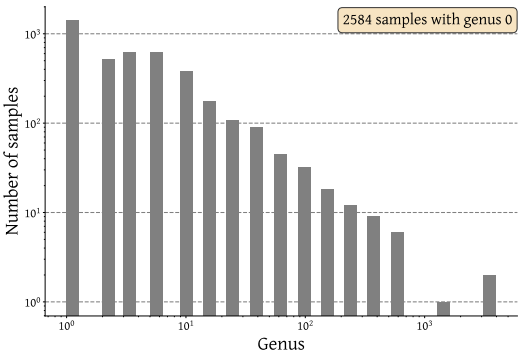
1. *Discrepancy between raw and perceptual components.* Artistic and CAD models are typically constructed from many sub-meshes, so the annotated component count often diverges from the semantically meaningful object count.

2. *Severe class imbalance.* While a wide range of genus values is theoretically possible, the overwhelming majority of meshes in both datasets have genus 0–2, making them unsuitable for balanced probing tasks.

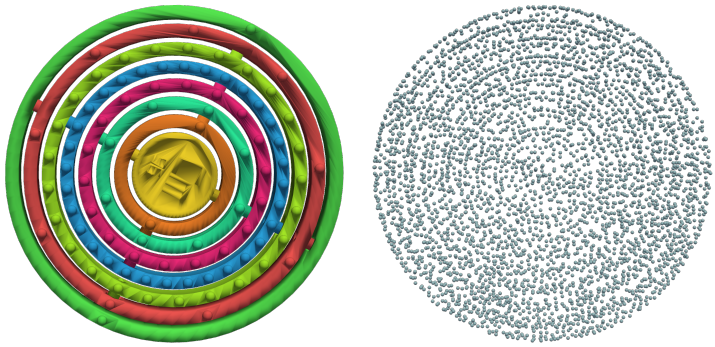
3. *Structural artifacts.* Many meshes are non-manifold or self-intersecting, rendering quantities such as the Euler characteristic ill-defined:

$$\chi = V - E + F = 2 - 2g - b + c \tag{1}$$

where V, E, F are the number of vertices, edges, and faces, g is genus, b the number of boundary components, and c the number of connected components. Attempts to repair such meshes (e.g., Manifold [23], ManifoldPlus [24], DOGN [48], CLAY [58]) face tradeoffs between oversmoothing geometry, introducing artifacts, or incurring prohibitive computational cost. As a result, these datasets cannot provide reliable large-scale topological ground truth.



(a) **Genus distribution of Thingi10K.** Both axes are in log scale. The genus, estimated from the Euler characteristic provided as metadata, could not be computed for 2,651 samples, which are excluded. The histogram shows that most meshes (over 3,000 of 7,344) have genus 0 or 1, highlighting the scarcity of higher-genus shapes.



(b) **Raw connected components.** The left figure shows a mesh made of 8 connected components, highlighted with different colors. However, once converted into a point cloud (right figure), we lose track of this information. Therefore, labeling this sample as having 8 connected components would be misleading for a model.

Figure 2

Synthetic datasets with limited diversity. The EuLearn dataset [15] addresses class imbalance by generating surfaces of varying genus from Fourier curves. While balanced across genera, EuLearn suffers from two critical issues: (i) samples within each genus class lack diversity, making them nearly indistinguishable, and (ii) strong correlations emerge between genus and canonical orientation, enabling trivial classification via nearest-neighbor with Chamfer distance. Consequently, EuLearn fails to disentangle topological structure from geometric shortcuts, limiting its usefulness for robust probing.

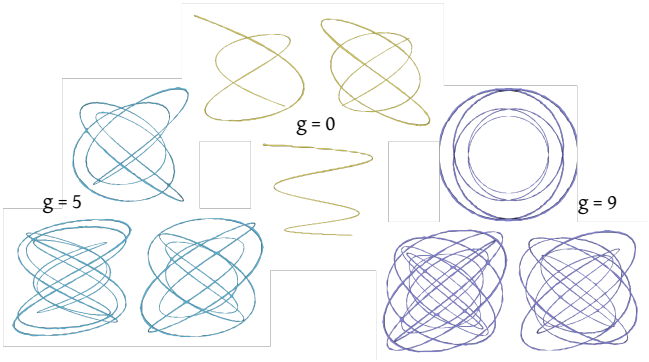


Figure 3: **Samples from the EuLearn dataset across different genera.** As the genus increases, shapes become geometrically more complex. This trend highlights a confounding factor in the dataset: geometric complexity grows together with genus.

2.2 Method

DONUT is a synthetic dataset designed to study how models capture topology under controlled conditions. Building such a dataset raises two challenges. First, synthetic data may differ from real 3D shapes used to pre-

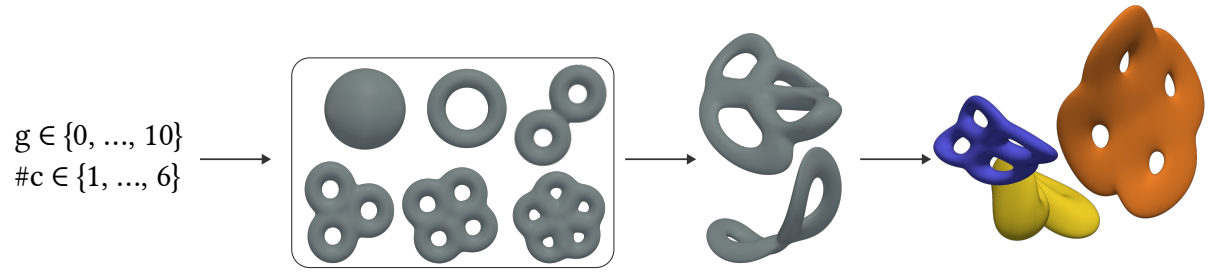


Figure 4: **Generation Process Overview.** (1) For each sample, we pick a target genus and number of connected components. (Section 2.2.1) (2) We then distribute these values across a set of template shapes (Section 2.2.2). (3) Each mesh is instantiated by sampling parameters from its parametric expression, and individually warped to bring more geometric diversity. (4) Finally, we apply a series of topology-preserving augmentations to increase geometric diversity while maintaining the original topological labels, and compose the final sample by merging all meshes. (Section 2.2.3)

train large models, which could make it out-of-distribution. Second, synthetic datasets often lack geometric diversity, making probing tasks trivial.

The first issue is minor because most pretrained encoders (e.g., trained on ShapeNet or Objaverse) already include synthetic geometry. Figure 5 confirms that DONUT samples project into the same embedding space as these datasets, showing no strong distribution shift. The second issue is addressed through a generation pipeline that jointly ensures (i) exact topological control, (ii) high geometric diversity, and (iii) scalability. Figure 4 summarizes the process.

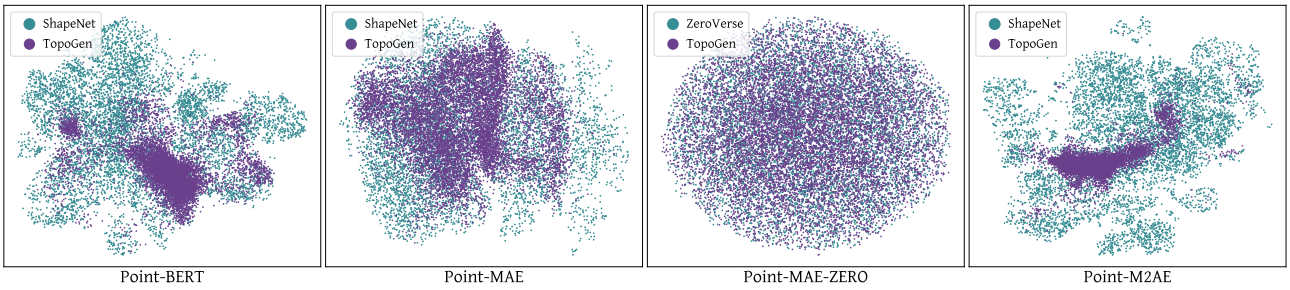


Figure 5: **UMAP embeddings of features learned by 3D encoders.**

2.2.1 Topological Label Sampling

To ensure balanced supervision across all classes, both the genus and the number of connected components are sampled such that their marginal distributions are approximately uniform. Appendix A formally describes how the labels are sampled and further details the properties of the dataset we used for all our experiments.

2.2.2 Shape Instantiation

Within a sample, each component is instantiated from a parametric family chosen according to its target genus:

- *Genus 0:* Superellipsoids and cones
- *Genus 1:* Supertoroids
- *Genus* ≥ 2*:* K-tori

Superquadrics. Superellipsoids (resp. toroids) are part of a wider family of parametric shapes called superquadrics. They were introduced by Barr et al. [4] in 1981 and are widely used in computer graphics for their ability to represent a large range of shapes with a small number of compact parameters. Superquadrics have recently regained attention in 3D scene understanding, notably in SuperDec [14], because of their expressiveness and their ability to approximate complex structures by composition. Their implicit equation is given by:

$$\left(\left| \frac{x}{s_x} \right|^{\frac{2}{\epsilon_2}} + \left| \frac{y}{s_y} \right|^{\frac{2}{\epsilon_2}} \right)^{\frac{\epsilon_2}{\epsilon_1}} + \left| \frac{z}{s_z} \right|^{\frac{2}{\epsilon_1}} = 1 \tag{2}$$

where $(s_x, s_y, s_z) > 0$ are scale factors along the x, y, z axes respectively, and $(\epsilon_1, \epsilon_2) > 0$ are shape exponents that modulate the surface's roundness (resp. sharpness). Sampling these parameters within predefined ranges allows to efficiently sample a variety of shapes while maintaining control over their topological properties (see Figure 6).

2.2.3 Meshes Generation

To generate many samples (up to 10^5) in a reasonable amount of time, we leverage the parametric forms of the shapes whenever possible. This approach minimizes computationally intensive operations and allows for efficient mesh generation.

Cones. To be completed

Superquadrics. Meshes corresponding to super ellipsoids and super toroids are generated directly from their parametric forms. The procedure consists of two steps: 1) generate a base template mesh (a sphere for ellipsoids, a torus for toroids) where vertices are expressed in spherical (resp. toroidal) coordinates, and 2) deform it according to the superquadric equations. This construction is lightweight, parallelizable, and supports fast large-scale dataset generation. The parametric equations we use in practice are as follows:

$$\text{Ellipsoid} \quad \begin{cases} x(u,v) &= s_x \, C_{\epsilon_1}(v) \, C_{\epsilon_2}(u) \\ y(u,v) &= s_y \, S_{\epsilon_1}(v) \, S_{\epsilon_2}(u) \\ z(u,v) &= s_z \, S_{\epsilon_1}(v) \end{cases} \qquad \text{Toroid} \quad \begin{cases} x(u,v) &= s_x \left(R + C_{\epsilon_1}(v)\right) C_{\epsilon_2}(u) \\ y(u,v) &= s_y \left(R + S_{\epsilon_1}(v)\right) S_{\epsilon_2}(u) \\ z(u,v) &= s_z \, S_{\epsilon_1}(v) \end{cases} \tag{3}$$

where $u, v \in [-\pi, \pi]$ are the vertex coordinates of the template mesh is spherical (resp. toroidal) coordinates and:

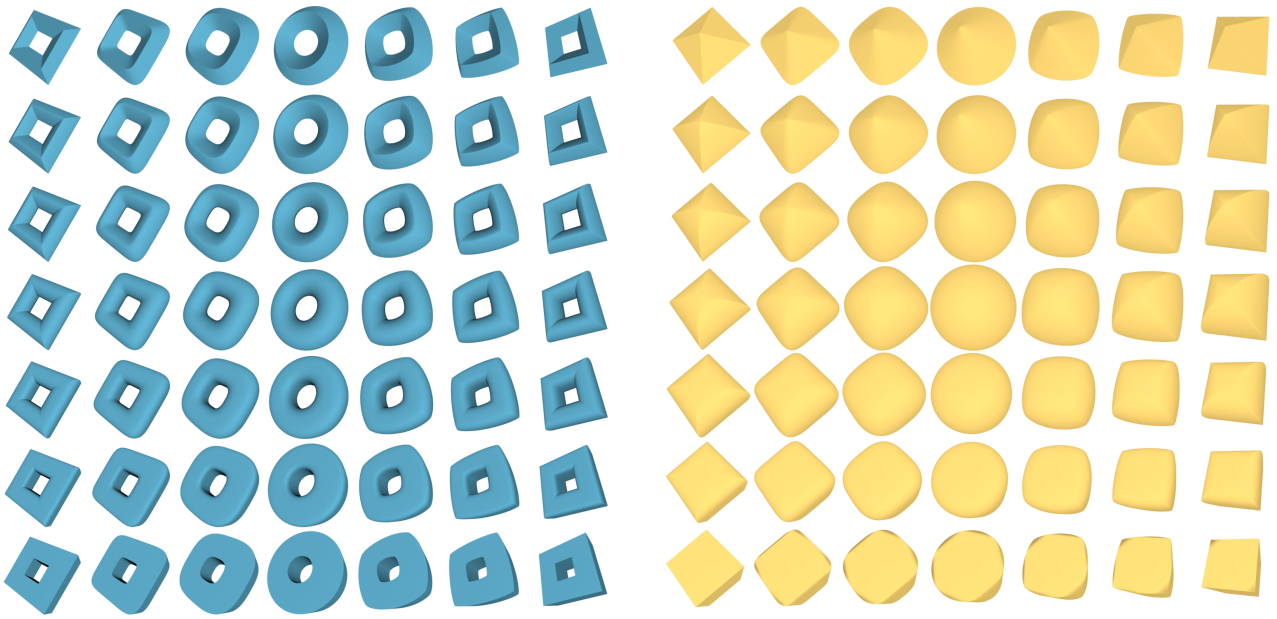
$$\begin{aligned} C_\epsilon(u) &= \text{sign}(\cos(u)) \, |\cos(u)|^\epsilon \\ S_\epsilon(u) &= \text{sign}(\sin(u)) \, |\sin(u)|^\epsilon \end{aligned} \tag{4}$$

K-tori. In the case of higher-genus meshes (Figure 8a), there are no closed-form parametric equations. Instead, we leverage the implicit formulation of tori. A k-torus is generated by (1) evenly distributing tori on a circle, (2) blending their signed distance functions (SDF) through a smooth union operator, and finally (3) extracting the mesh via marching cubes with a grid-size. However, this process is more computationally intensive as it requires applying marching cubes the discretized SDF, whose complexity scales in $O(N^3)$ where N is the grid size. And since we seek to have high quality meshes to preserve fine-grained details (here holes), we use a higher grid resolution.

To smoothly blend the SDF of each torus generated independently, we use the *smooth minimum* operator:

$$\text{softmin}_k(s_1, s_2, \ldots, s_n) = -\frac{1}{k} \log \left(\sum_{i=1}^{n} e^{-k s_i} \right) \tag{5}$$

Where $(s_1, s_2, \ldots, s_n) \in \mathbb{R}^{(N^3)}$ are the SDF of individual tori and k is a hyperparameter controlling the smoothness of the blending. This operator is applied voxel wise. In other words, each voxel of the blended SDF is assigned with the smooth minimum value of the individual SDF.



(a) Supertoroids. (b) Superellipsoids.

Figure 6: Overview of different shapes obtained for fixed values of $a_i, i \in \{1, 2, 3\}$ and increasing values of ϵ_1 (left to right) and ϵ_2 (bottom to top). (a) Different supertoroids. As mentioned in TO ADD, using these shapes for k-tori ($k \geq 2$) is challenging because they may not preserve the genus, for instance if some parts are too thin or sharp. (b) Different superellipsoids.

2.2.4 Topological Consistency and Diversity

Two challenges remain to be addressed. First, when placing components within a sample, we must ensure they don't overlap. At the same time, they need to be close enough to make the samples both topologically consistent and challenging enough for models to correctly identify the number of connected components. In early experiments, we observed that if components are too far apart, the task can be trivially solved using a simple KNN classifier with Chamfer distance. Second, even with careful placement, models tend to overfit, sometimes even when using simple linear heads on top of learned features, showing that this alone is insufficient to evaluate their true topological understanding. To increase variability while preserving topological labels, we apply transformations both at the sample level and the individual component level.

Topological Consistency. We developed a two-stage placement procedure. Suppose there are already some components correctly placed within a sample. To add a new component, we first place it randomly in the sample and then check for intersections with the existing components. Experimentally, we observed that a randomly placed mesh overlaps with at most two other components. If the new component overlaps with only one existing component, we identify the point on the new mesh that is most deeply inside the other component and push the new component along the opposite direction of the surface normal at that point. For intricate shapes, a single adjustment may create new overlaps, so we repeat this step five times (see Figure 7). If overlaps remain after five attempts, the component is discarded and a new one is generated and randomly placed. This iterative procedure improves efficiency: in practice, one or two adjustments are sufficient to resolve overlaps, making it faster than discarding and regenerating components immediately.

Variability. We apply transformations both at the sample level and at the component level. Commonly used rigid transformations, such as rotations and scaling, are applied to both components and full samples. At the component level, we do not apply translations to avoid creating overlaps that would break the overall topology of the sample. To introduce additional geometric variability, we also apply twisting deformations (Figure 8b). Without loss of generality, we assume that a component (or a full sample) lies within the unit sphere, as global scaling can be applied afterward. First, we uniformly sample a direction on the unit sphere, which defines an axis for the twist. Next, we define a smooth scalar function along this axis that determines

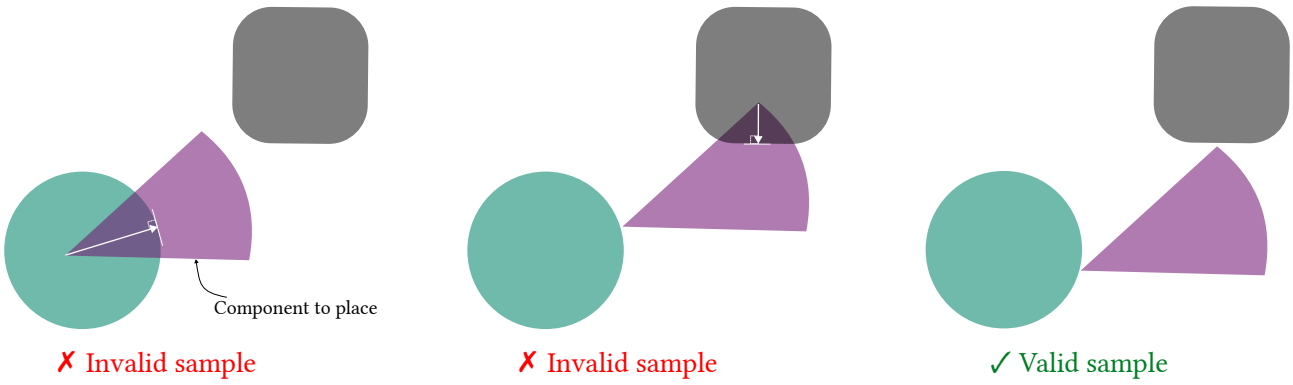


Figure 7: **Component placement procedure.** (1) A new component is randomly placed within the sample. (2) If it overlaps with existing components, we identify the point on the new mesh that is most deeply inside the other component and push the new component along the opposite direction of the surface normal at that point. This step is repeated up to five times. If overlaps remain, the component is discarded and a new one is generated and randomly placed. This procedure ensures that components do not overlap while remaining close enough to preserve topological consistency and challenge models to correctly identify the number of connected components.

the rotation angle. Finally, each vertex is rotated around the axis by the angle given by the scalar function evaluated at the point where the vertex projects along the axis. This procedure introduces non-rigid variability while preserving the genus and connectivity of each component and of the overall sample.

2.3 Analysis

This section provides a detailed characterization of DONUT. We first present dataset statistics (Section 2.3.1) and label distributions. We then assess baseline performance and introduce two complementary analyses to ensure models trained on DONUT truly learn the intended topological signal rather than relying on shortcuts. The first (Section 2.3.2) measures performance drop when models trained on DONUT are evaluated on a distinct dataset. The second (Section 2.3.3), visualizes model attention on template shapes to reveal which regions influence predictions.

2.3.1 General Properties

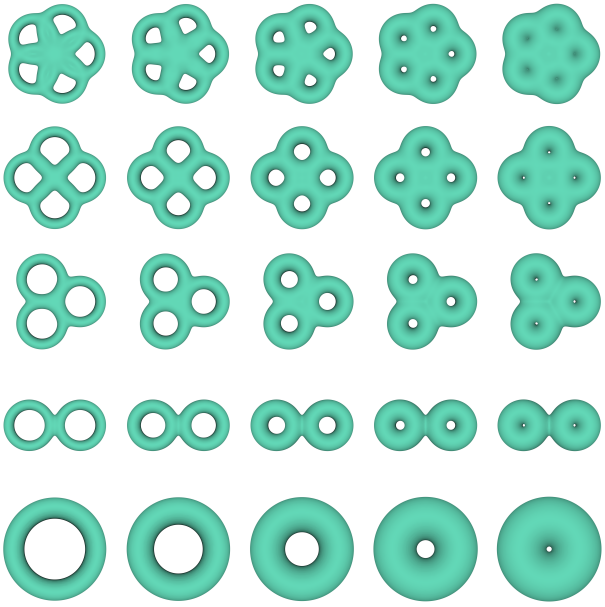
Dataset statistics. DONUT contains 29,517 samples divided into training, validation, and test splits of 80%, 10%, and 10%. Each sample has between 1 and 6 connected components, with the total genus ranging from 0 to 10. A single component has genus at most 5 (corresponding to a 5-torus (Figure 8a)). The dataset size was chosen to balance feasibility and reliability: smaller datasets led to overfitting while much larger datasets made experiments costly. Figure 9 reports the marginal distributions of genera and connected components.

Distributions and scalability. Figure 5 shows that synthetic samples from DONUT occupy the same representation space as training data used by common 3D encoders. This indicates that the dataset lies within the learned distribution of existing models rather than being out-of-distribution. In addition, Figure **??** reports the time required to generate different dataset sizes, showing that large-scale variants remain practical.

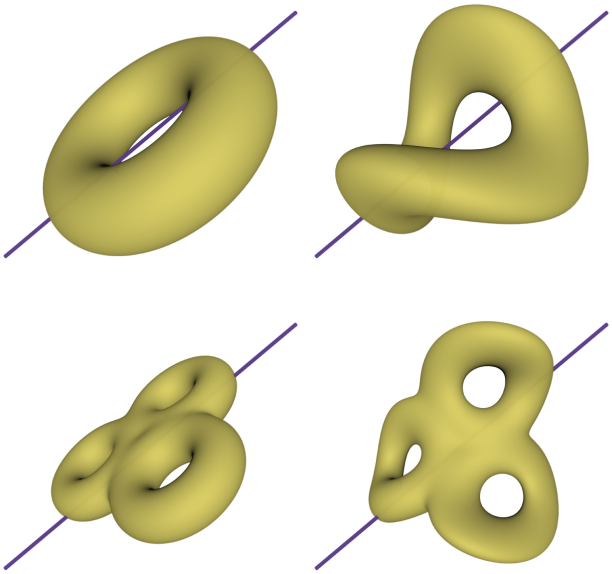
2.3.2 Baselines and Transferability

We trained several classical point-cloud models (PointNet, PointNet++, DGCNN) and recent persistence-based models (PersFormer, xPerT, PersLay) on DONUT. Results are reported in Table 2. Beyond in-distribution accuracy, a key question is whether these models actually capture topology or only rely on geometric characteristics.

Transferability provides a natural probe. To test this, we evaluated models on a curated subset of ABC, without fine-tuning. This set contains shapes whose geometry differs significantly from the training distri-



(a) **Degrees of freedom allowed on k-tori.** Prior to the sequence of transformations applied to each individual component (e.g. rotation, twisting) we allow some variability during the generation. The x-axis represents the ratio *major radius / minor radius*. The y-axis shows how the template shape is modified as we increase the value of k (i.e. the genus). *Note:* In the actual dataset, 1-tori are generated with the parametric representation of supertoroids, since a regular torus is a particular case of supertoroid. However, for $k \geq 2$, using a composition of 1-tori is more reliable. Some parameter sets can lead to undesirable holes in the final shape.



(b) **Effect of twisting.** Original template shapes (left) are twisted along the purple axis (right). Twisting deformations introduce non-rigid variability while preserving the genus and connectivity of each component. Here the scalar function defined along the axis is affine between $-\pi/6$ and $\pi/3$. We apply this augmentation at the component and sample level.

bution, but whose topology is well characterized and within the label space. If models have learned topology, they should retain reasonable performance on ABC despite geometric differences.

Results. Table 2 shows that DGCNN and PointNet++ achieve competitive results on predicting the number of connected components of DONUT. However, it also highlights that raw knowledge transfer, without fine-tuning, to ABC is very limited. It suggests that models remain highly sensitive to geometric distribution shifts, even when the task is purely topological.

2.3.3 Saliency Analysis

To further understand model behavior, we visualize in Figure 10 saliency maps on toy shapes. These maps highlight which regions contribute most to predictions. When trained to predict genus, models tend to focus on cycles, while for connected components they concentrate around contact points between shapes. Such results suggest that models attend to meaningful topological structures rather than exploiting geometric shortcuts.

Definition. Let a point cloud be $X = [x_1, \ldots, x_N] \in \mathbb{R}^{N\times 3}$. Let $f_\theta(X) \in \mathbb{R}^C$ be the vector of class logits for C classes. The saliency of point x_i for class c is the norm of the gradient of the class score with respect to the point coordinates:

$$S_{i,c} = \left\| \frac{\partial f_\theta(X)_c}{\partial x_i} \right\|_2 \tag{6}$$

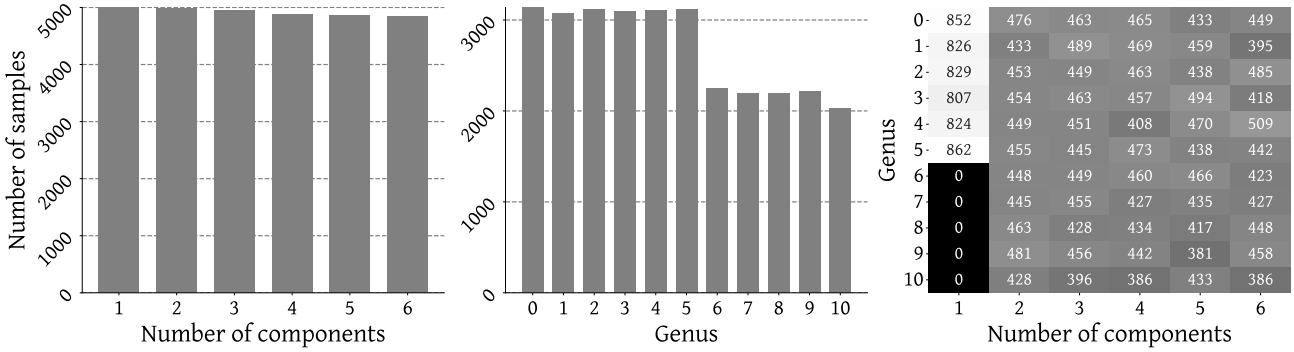


Figure 9: **Label distributions for genus and connected components.** The marginal distributions of genus and connected components (left and middle histograms) are approximately uniform, ensuring balanced supervision across topological classes. The right plot shows the joint distribution. The absence of samples with one connected component and genus greater than 5 stems from the constraints of our generation process, which limits individual components to a maximum genus of 5. The only way to achieve a genus greater than 5 is by combining multiple components.

		Genus			**Connected Components**		
Dataset	Model	MSE↓	Acc.↑	F1↑	MSE↓	Acc.↑	F1↑
				Point-based models			
DONUT	PointNet [38]	14.2	20.9	17.4	0.92	56.7	55.5
	PointNet++ [37]	1.38	**59.5**	59.3	0.17	84.2	84.1
	DGCNN [49]	1.05	51.5	51.3	0.11	**89.1**	89.1
ABC	PointNet [38]	9.35	$12.7_{-8.2}$	$12.1_{-5.3}$	0.93	$21.6_{-35.1}$	$22.0_{-33.5}$
	PointNet++ [37]	9.91	$16.6_{-42.9}$	$15.0_{-44.3}$	1.16	$24.6_{-59.6}$	$23.1_{-61.0}$
	DGCNN [49]	11.7	$18.1_{-33.4}$	$15.2_{-36.1}$	1.04	$27.7_{-61.4}$	$25.4_{-63.7}$

Table 2: **Performance of baseline models trained on DONUT and evaluated both in-distribution and on ABC.** Values are reported as *DONUT/ABC*. We report mean squared error (MSE), balanced accuracy (Acc.), and balanced F1-score (F1). MSE is reported to capture how far off predictions are from the ground-truth on average, which is particularly relevant given the natural hierarchy of the labels (genus and connected components): misclassifying by a larger margin is more severe than a closer miss. Training setup is detailed in Table 6

Interpretation on point clouds. Saliency quantifies which points most influence the class score under small coordinate changes. Points with larger $S_{i,c}$ contribute more to the decision, because an infinitesimal displacement of those coordinates changes $f_\theta(X)_c$ the most. In our topological setting this yields a clear reading. For genus prediction, high saliency concentrates along cycles whose perturbation would alter the number of one dimensional holes. For connected components, high saliency appears near thin links and contact regions where minor displacements can merge or split components. These observations are consistent with prior work that adapts saliency to point clouds to estimate pointwise importance for recognition.

Practical computation. We backpropagate the gradient of the selected logit with respect to the input coordinates and compute $S_{i,c}$ for all points. Raw gradients can be noisy. Before visualization we therefore post-process raw maps by adapting [44]. Concretely, we clip $S_{i,c}$ at the τ-th percentile q_τ to suppress extreme outliers, then scale by $\widetilde{S_{i,c}} = \min(S_{i,c}, q_\tau)/q_t au \in [0,1]$. . We set all values below a small fraction of the maximum to zero to improve visual clarity. Smoothing strategies such as averaging saliency over noisy perturbations of the input are known to reduce noise further, but we keep our pipeline simple and report thresholded and normalized maps.

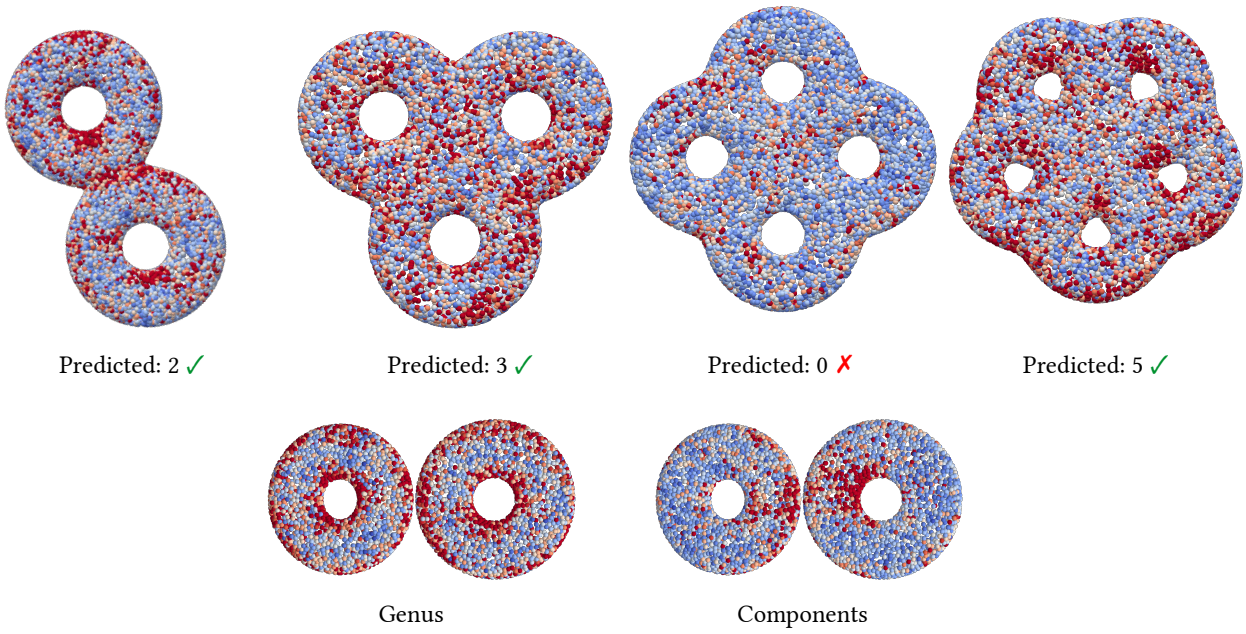


Figure 10: **Saliency maps of toy shapes with varying topology.** We show saliency for DGCNN as it's the best performing model on DONUT (Table 2). Top row shows saliency maps on genus prediction for samples with a single connected component and increasing genus. We also specify the genus predicted by the model. Bottom row highlights how saliency maps change between genus (left) and connected component (right) prediction for the same sample with two connected components.

Results. Figure 10 highlights that when the prediction is correct, saliency sometimes captures the expected topological structures, as seen in the top row for genus 2 and 5 shapes. However, even for successful predictions, interpretation is not always straightforward. The points that contribute most to the prediction do not always follow a clear or consistent spatial pattern. When the model fails, as for genus 4, the saliency distribution becomes even less interpretable.

The second row of Figure 10 provides more insight. For genus prediction, the salient points tend to cluster around the main topological structures, while for component prediction, they concentrate near the contact region between connected components. This suggests that the model distinguishes between tasks by attending to different geometric cues.

It is also worth noting that the saliency patterns vary with rotation, scale, and the prominence of the topological structures themselves, such as the size of a hole or the distance between components. *These observations indicate that the model's predictions remain strongly influenced by geometric properties, even when the task requires reasoning about topology.*

2.4 Limitations and Further Improvements

Topological Variety. The current dataset has limited topological variety. It only encodes two characteristics: genus and number of connected components. This is mainly because all meshes are watertight and manifold. In this setting, topological invariants are trivial to deduce ($\beta_1 = 2 \times (\text{"}genus\text{"})$, $\beta_2 = \beta_0$). A more challenging extension would include surfaces with boundaries, non-watertight meshes, or nested shapes. While the current generation process allows nested components in theory, no such cases are present in practice. Another direction is to incorporate families of parametric shapes with non-trivial topology, such as catenoids, helicoids, or Möbius strips.

Geometric Complexity. The geometric diversity is also limited. It could be extended by adding parametric families that introduce richer geometric structures. For example, the Superformula introduced by Gielis (2003) [17] generalizes superellipses and provides a broad range of shapes. Although we did not add such families here, since the dataset is mainly used for evaluation and no overfitting issues were observed with the current

configuration, this would be valuable in other settings. In particular, if the dataset is used for pretraining, as in Point-MAE-Zero, introducing more geometric complexity would improve its utility.

3 Used Models

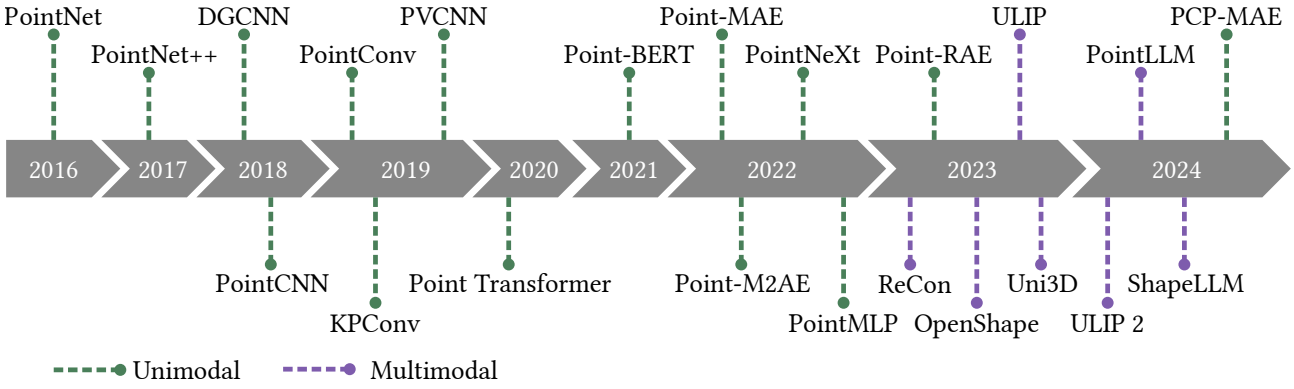


Figure 11: **Timeline of models used as 3D point-cloud encoders**

This section provides an extensive description of the models used in our experiments. We focused on three main types of models: point cloud-based models, transformer-based models, and persistence-based models. Since the latter require background on persistence homology, they're further detailed in Section 4.2. On the one hand, point cloud-based models are used as baselines for specific tasks (see Table 2). This choice stems from the widely known limitations in terms of robustness to input corruption and generalization [30] of such models. On the other hand, transformers have demonstrated state-of-the-art performance on various 3D shape understanding tasks [57, 36, 59, 60], and we are therefore insterested here in their ability to learn rich representations from data.

3.1 Baselines Models

We compare results obtained with transformer-based models to several baselines widely used in the literature. These models directly act on individual points without any tokenization step. They are all invariant to point permutations and can handle varying point cloud sizes. A final feature vector is obtained through a global pooling operation, which is then used for downstream tasks.

3.1.1 PointNet

PointNet [38] introduced the first deep learning architecture that operates directly on raw 3D points. Each point is independently transformed through shared multilayer perceptrons (MLPs) to extract per-point features, which are then aggregated using a symmetric pooling function—typically max pooling—to obtain a global shape descriptor. This design ensures invariance to input permutation and robustness to geometric transformations through a learned spatial transformer network. However, since PointNet treats each point independently before pooling, it lacks the ability to explicitly capture local geometric structures and neighborhood relations within the point cloud.

3.1.2 PointNet++

PointNet++ [37] extends PointNet by introducing a hierarchical feature learning mechanism that captures local geometric patterns at multiple scales. It partitions the point cloud into overlapping local regions using sampling and grouping operations, applies PointNet within each region to extract local features, and recursively aggregates them to form higher-level representations. This hierarchical structure allows PointNet++ to model fine-grained local geometry while maintaining the permutation invariance of PointNet. The introduction of radius-based grouping and multi-scale feature extraction significantly improves robustness to non-uniform sampling and enhances generalization to complex shapes.

3.1.3 DGCNN (Dynamic Graph CNN)

DGCNN [49] further advances local structure modeling by representing the point cloud as a dynamic graph, where edges connect each point to its k-nearest neighbors in the feature space. Unlike PointNet++'s static grouping, DGCNN updates the neighborhood graph at each layer, allowing feature dependencies to evolve adaptively as the representation becomes more abstract. The model applies an edge convolution operator that learns features based on both point attributes and relative positional differences, effectively capturing local geometric relations and contextual information. This dynamic graph formulation enables DGCNN to achieve strong performance on tasks requiring fine spatial reasoning, such as segmentation and shape classification. Even though earlier but also some recent models [31, 51, 45, 33] have shown competitive performance, it is admitted that performance on DGCNN account for all the other possible baselines.

3.2 Transformer-based Models

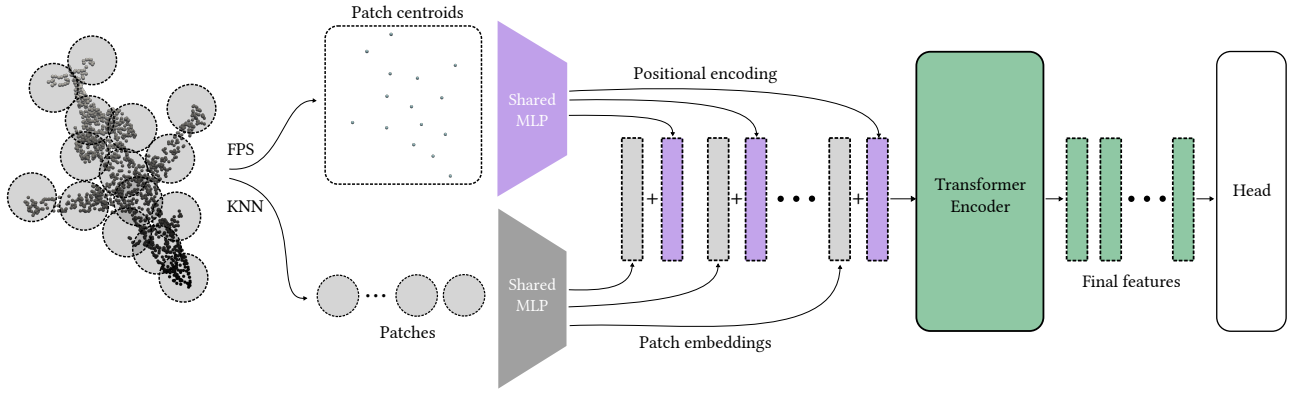


Figure 12: **Transformer Overview for 3D point-clouds.** A point cloud is first partitioned into local patches using farthest point sampling (FPS) and K nearest neighbors (KNN). Patches are then embedded and augmented with positional encodings. The resulting sequence of patch embeddings is processed by a Transformer encoder to produce a global shape representation.

Recent advances in transformer architectures for 3D shape understanding have largely built on the intuition that ideas from NLP and vision can be transferred to point cloud data. While the geometry of 3D shapes is fundamentally different from text or images, most models share a similar formal structure that follows three main stages: patchification, embedding, and token processing through a transformer encoder.

Let a point cloud be represented as a set of N points:

$$\mathcal{P} = \{p_i \in \mathbb{R}^3\}_{i=1}^{N}. \tag{7}$$

The patchification step partitions $\mathcal{P}$ into M local regions $\{\mathcal{P}_j\}_{j=1}^{M}$, each containing K points. A common approach is to apply Farthest Point Sampling (FPS) to select M center points $\{\mathbf{c}_j\}_{j=1}^{M}$, and then group each center with its K nearest neighbors using KNN:

$$\mathbf{c}_j = \text{FPS}(\mathcal{P}), \quad \mathcal{P}_j = \text{KNN}(\mathbf{c}_j, \mathcal{P}) = \{p_{j,1}, \ldots, p_{j,K}\} \subset \mathbb{R}^3. \tag{8}$$

Each patch $\mathcal{P}_j$ is thus a small local subcloud capturing fine-scale geometry around $\mathbf{c}_j$.

A lightweight local encoder f_{loc} (often a PointNet variant) maps each patch to a feature vector:

$$\mathbf{x}_j = f_{\text{loc}}(\mathcal{P}_j) \in \mathbb{R}^d, \tag{9}$$

This produces a sequence of patch embeddings $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_M]^\top \in \mathbb{R}^{M \times d}$, which serves as the input to the transformer. The embedding dimension d is fixed for all patches.

To provide geometric context, positional encodings are added to the embeddings. Each patch center $\mathbf{c}j$ is projected through a multilayer perceptron $fpos$:

$$\mathbf{p}_j^{\text{pos}} = f_{\text{pos}}(\mathbf{c}_j) \in \mathbb{R}^d, \tag{10}$$

The final input tokens are obtained as

$$\mathbf{z}_j^{(0)} = \mathbf{x}_j + \mathbf{p}_j^{\text{pos}} \tag{11}$$

Optionally, a learnable global token $\mathbf{z}_{\text{cls}}^{(0)} \in \mathbb{R}^d$ is prepended to the sequence to aggregate global information, yielding:

$$\mathbf{Z}^{(0)} = [\mathbf{z}_{\text{cls}}^{(0)}; \mathbf{z}_1^{(0)}, \ldots, \mathbf{z}_M^{(0)}] \in \mathbb{R}^{(M+1)\times d}. \tag{12}$$

The transformer encoder then applies a stack of L layers, each composed of multi-head self-attention (MSA) and feed-forward networks (FFN):

$$\mathbf{Z}^{(l+1)} = \text{FFN}(\text{MSA}(\mathbf{Z}^{(l)})) + \mathbf{Z}^{(l)}, \quad l = 0, \ldots, L-1. \tag{13}$$

The final output $\mathbf{Z}^{(L)}$ contains contextualized representations of all patches, where each token attends to others to model long-range dependencies across the shape. The representation of the class token $\mathbf{z}_{\text{cls}}^{(L)}$ (if used) or a global pooling of patch embeddings is then employed as the final shape descriptor.

This framework mirrors the architecture of Vision Transformers, with the key difference that the tokens correspond to irregular geometric neighborhoods rather than fixed image patches. Most recent models differ only in how the tokens are defined, how positional encodings are implemented, and what self-supervised pretraining objectives are used. These design variations lead to distinct inductive biases and training dynamics, as detailed in the following subsections.

3.2.1 Point-BERT: Discrete Tokenization and Masked Modeling

Point-BERT [57] was among the first methods to transfer the BERT-style masked pretraining paradigm to point clouds. Its core idea is to convert each patch embedding $\mathbf{x}_j \in \mathbb{R}^d$ into a discrete token index, enabling the use of a masked language modeling objective analogous to NLP.

To achieve this, a discrete variational autoencoder (dVAE) is first trained to quantize the continuous patch embeddings. The encoder of the dVAE, $f_{\text{enc}}^{\text{dVAE}}$, maps a patch $\mathcal{P}_j$ to a latent code:

$$\mathbf{h}_j = f_{\text{enc}}^{\text{dVAE}}(\mathcal{P}_j) \in \mathbb{R}^{d'} \tag{14}$$

This latent code is then discretized by assigning it to its nearest entry in a learnable codebook $\mathcal{C} = \mathbf{c}1, \ldots, \mathbf{c}|\mathcal{C}| \subset \mathbb{R}^d$:

$$\tilde{\mathbf{h}}_j = \mathbf{c}_{k_j}, k_j = \arg\min_k ||\mathbf{h}_j - \mathbf{c}_k||_2 \tag{15}$$

The decoder $f_{\text{dec}}^{\text{dVAE}}$ reconstructs the original patch from $\tilde{\mathbf{h}}_j$, ensuring that each codeword captures meaningful local geometry.

Once the vocabulary is trained, the transformer is pretrained with Masked Point Modeling (MPM). A subset $\mathcal{M} \subset 1, \ldots, M$ of patches is masked, and the model receives as input only the unmasked tokens and their positional encodings. The objective is to predict the discrete indices $k_{j\,j\in\mathcal{M}}$ of the masked patches:

$$\mathcal{L}_{\text{MPM}} = -\sum_{j\in\mathcal{M}} \log p_\theta(k_j | \{\mathbf{z}_i^{(0)}\}_{i\notin\mathcal{M}}) \tag{16}$$

where p_θ is parameterized by the transformer. A learnable global token $\mathbf{z}_{\text{cls}}^{(0)}$ is updated jointly with the patch tokens during pretraining, and its final state $\mathbf{z}_{\text{cls}}^{(L)}$ is used as the shape embedding for downstream tasks.

3.2.2 Point-MAE: Continuous Tokens and Masked Autoencoding

Point-MAE [36] simplifies this framework by removing discrete tokenization and operating directly on continuous patch embeddings. Each patch $\mathcal{P}_j$ is encoded as a continuous vector $\mathbf{x}_j = f_{\text{loc}}(\mathcal{P}_j) \in \mathbb{R}^d$, and a random subset $\mathcal{M}$ of them is masked. Only the unmasked patches are passed to the transformer encoder.

A shared learnable vector $\mathbf{t}_{\text{mask}} \in \mathbb{R}^d$ replaces the embeddings of masked patches. The transformer encoder processes only unmasked embeddings:

$$\mathbf{Z}_{\text{enc}} = \text{Encoder}(\{\text{x}_j + p_j^{\text{pos}}\}_{j \notin \mathcal{M}}) \tag{17}$$

Then a lightweight decoder reconstructs the masked patches from the concatenation of encoder outputs and mask tokens:

$$\mathbf{Z}_{\text{dec}} = \text{Decoder}([\mathbf{Z}_{\text{enc}}; \{\mathbf{t}_{\text{mask}} + p_j^{\text{pos}}\}_{j \in \mathcal{M}}]) \tag{18}$$

The reconstruction target is the set of 3D coordinates $\mathcal{P}_j j \in \mathcal{M}$, and the loss is usually a Chamfer distance:

$$\mathcal{L}_{\text{MAE}} = \sum_{j \in \mathcal{M}} \text{Chamfer}(\hat{\mathcal{P}}_j, \mathcal{P}_j) \tag{19}$$

This continuous formulation avoids the need for a pretrained tokenizer and allows more efficient computation since only unmasked patches go through the heavy encoder. It also directly learns a geometric reconstruction objective rather than a discrete prediction one. Unlike Point-BERT, the class token is not used during pretraining but introduced only during fine-tuning for downstream tasks. This makes pretraining focus on local geometric recovery, leaving semantic aggregation to task-specific adaptation.

3.2.3 Hierarchical Extensions: Multi-Scale Representations

Point-M2AE [59] extends Point-MAE by introducing multi-scale feature processing to better capture hierarchical shape structure. Given the initial point cloud $\mathcal{P}$, successive downsampling layers produce coarser point sets $\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots$, with decreasing sizes $N_1 > N_2 > \ldots$ and corresponding patch sets $\{\mathcal{P}_j^{(\ell)}\}_{j=1}^{M_\ell}$. Each scale has its own embedding function $f_{\text{loc}}^{(\ell)}$, producing patch features $\mathbf{X}^{(\ell)} \in \mathbb{R}^{M_\ell \times d_\ell}$.

$$\mathbf{Z}^{(\ell+1)} = \text{Transformer}^{(\ell)}(\mathbf{X}^{(\ell)}, \mathbf{Z}^{(\ell-1)}) \tag{20}$$

This design parallels the feature hierarchies in PointNet++ and U-Net architectures, allowing Point-M2AE to represent both fine-grained details and global context. The masking strategy is also applied at multiple scales to ensure consistency across abstraction levels.

By combining local spatial self-attention at fine resolutions with global aggregation at coarser ones, Point-M2AE improves generalization and compactness, outperforming single-scale variants on both classification and reconstruction tasks.

3.2.4 Further Improvements

Several recent works refine the MAE framework to better encode geometric priors and invariances. PCP-MAE [60] modifies positional encodings to prevent information leakage: instead of directly providing patch centers $\mathbf{c}_j$, it requires the model to predict them, forcing reliance on relational geometry rather than absolute positions. HFBRI-MAE [56] augments each token with handcrafted rotation-invariant descriptors, ensuring robustness to arbitrary 3D rotations. Point-RAE [32] introduces a regression-before-autoencoding strategy, first predicting geometric features and then reconstructing points. This separation decouples encoder and decoder learning, preventing overfitting of the reconstruction loss.

Together, these improvements show that incorporating geometric inductive biases into the transformer pipeline strengthens both robustness and generalization.

3.2.5 Comparative View

Despite architectural differences, all transformer-based 3D encoders share the same formal backbone described by Figure 12. They differ mainly in how tokens are defined (discrete vs. continuous), the pretraining objective (classification vs. reconstruction), and the presence of hierarchical processing.

Point-BERT's discrete modeling brings a language-like structure but requires a learned tokenizer.
Point-MAE simplifies the pipeline and enables efficient continuous pretraining.
Point-M2AE extends this formulation to multiple scales, encoding richer geometric dependencies.

This report focuses on Point-BERT, Point-MAE, and Point-M2AE as representative models, as they capture the main design paradigms in transformer-based 3D shape understanding. It also considers PCP-MAE as it builds directly on Point-MAE and is the current state-of-the-art on most downstream tasks we further describe.

3.3 Multimodal approaches

Several recent works extend 3D transformers to multimodal settings by aligning point cloud embeddings with image and text features in a shared latent space. Given a 3D encoder f_{3D} producing embeddings $\mathbf{z}_{3D} \in \mathbb{R}^d$, and pretrained encoders f_{img}, f_{text}, the goal is to enforce semantic consistency through a contrastive objective:

$$\mathcal{L}_{align} = -\log \frac{\exp\left(\text{sim}(\mathbf{z}_{3D}, \mathbf{z}_{img})/\tau\right)}{\sum_{(\mathbf{z}'_{3D}, \mathbf{z}'_{img})} \exp\left(\text{sim}(\mathbf{z}'_{3D}, \mathbf{z}'_{img})/\tau\right)}. \tag{21}$$

where sim denotes cosine similarity.

ULIP [54] and ULIP-2 [53] follow this strategy by coupling Point-BERT with image and text encoders. The resulting models learn cross-modal alignment but remain bounded by the geometric expressiveness of the 3D backbone. PointLLM [52] further connects a frozen 3D encoder to a large language model through a lightweight adapter, while ShapeLLM [39] and Uni3D [61] generalize the same principle using multi-view or ViT-based 3D encoders.

Although these systems broaden the representational scope of 3D models, their effectiveness strongly depends on the unimodal encoder. The quality of $\mathbf{z}_{3D}$ determines how well multimodal alignment preserves geometric structure. Our own trials confirmed that replacing or fine-tuning the multimodal layers rarely compensates for weaknesses in the 3D representation itself.

Why we focus on unimodal encoders. Because multimodal models inherit the inductive biases of their 3D backbones, we restrict our analysis to unimodal encoders. This allows us to isolate how transformers process geometric and topological structure within point clouds, without interference from external modalities. In Section 4, we further quantify how such encoders capture non-semantic information using persistence-based analysis.

4 Experiments

This section presents a series of experiments designed to investigate the relationship between learned 3D representations and topology. Our goal is to understand whether and how transformer models capture topological features of shapes, when trained on standard 3D data. We progressively move from a direct evaluation of encoded topology to more indirect analyses. Throughout these experiments, we use DONUT as the main benchmark.

We first conduct a probing analysis to estimate how much topological information can be linearly decoded from the representations of different layers (Section 4.1). This experiment provides an initial view of the presence and organization of topological cues inside the network. We then study the alignment between the learned embedding space and topological structures derived from persistent homology (Section 4.2). This second analysis gives a more geometric perspective on how representations relate to the shape topology. Finally (Section 4.3), we investigate how changing the pretraining data distribution affects these findings, showing that current pretraining corpora do not explicitly encourage the capture of topological structures.

4.1 Model Probing

We start by probing transformer models to assess whether their internal representations encode information about shape topology. Specifically, we test if simple linear classifiers can recover the genus and number of

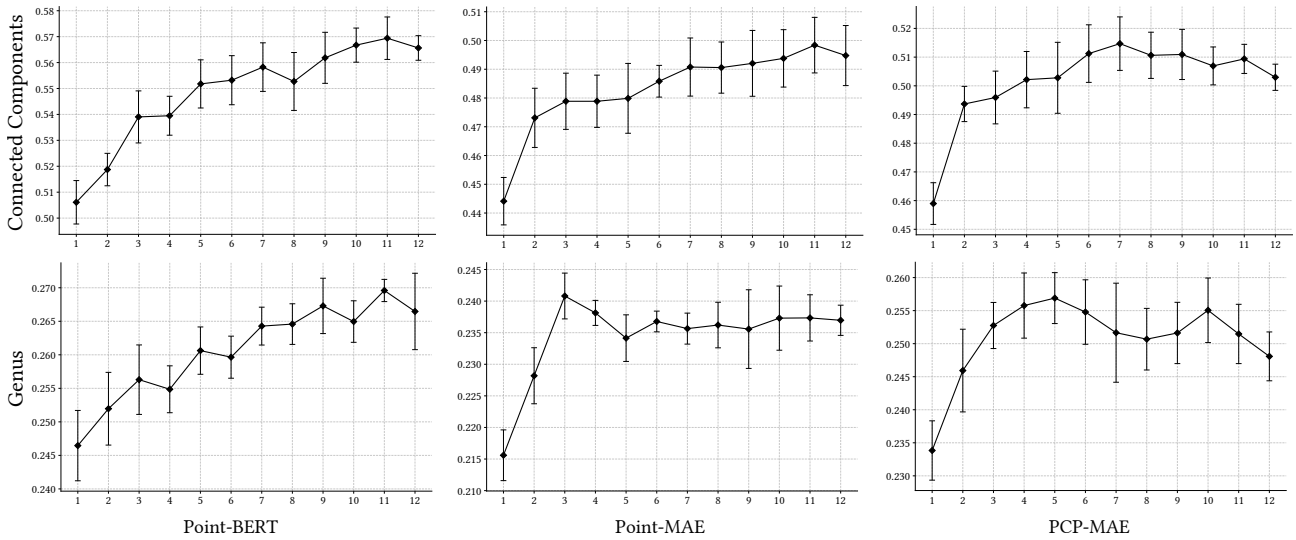


Figure 13: **Linear probing accuracy across transformer layers for Point-BERT (left column) and Point-MAE (right column).** We report probing accuracy for connected components (top row) and genus (bottom row). For each setting, accuracy values are averaged over the validation sets from a 5-fold cross-validation procedure, with error bars representing the corresponding standard deviations. For Point-BERT, probing is applied to the CLS token, which aggregates global information throughout the network. For Point-MAE, which does not include a CLS token during pretraining, we instead use the max-pooled patch embeddings from each layer.

connected components of shapes from the DONUT dataset using the representations of each intermediate layer. This approach provides a first estimate of how topological signals are distributed across the model depth.

Motivation. Linear probing is a standard approach in representation learning to evaluate how much a given factor of variation is linearly accessible from the learned embeddings. It assumes that if a linear classifier performs well, the corresponding information is already represented in a linearly separable manner. Importantly, several works have shown that the most informative representations for downstream tasks are not always located in the final layers of a model. Intermediate layers can encode more general or transferable structures before they are transformed into task-specific representations. This has been established in both vision and language transformers [2, 20, 40]. We therefore analyze not only the final layer, but the entire depth of each model, to capture where topology-related features emerge.

Experimental setup. For each evaluated model, we freeze the pretrained backbone and extract offline activations from all transformer layers on the DONUT dataset. At every layer l, we train two separate linear classifiers on top of the frozen representations to predict (i) the genus and (ii) the number of connected components. We perform 5-fold cross-validation and report averaged accuracy on the validation set.

Results. Overall, Figure 13 probing accuracy tends to increase with network depth for both models, indicating that higher transformer layers capture richer structural information. The improvement is modest for genus prediction, with an average increase of about 3% between the first and last layers. In contrast, connected component prediction improves by roughly 8%, suggesting that the notion of component separation becomes more explicit as features become increasingly global. This aligns with the expectation that transformers progressively aggregate local patch information into holistic shape-level representations.

Interestingly, in Point-MAE, accuracy for genus prediction plateaus after layer 5. This saturation may reflect the limited capacity of shallow hierarchies to encode topological invariants that depend on multi-scale relationships between local patches. Testing deeper or hierarchical variants of Point-MAE could help determine whether such information emerges later in the network.

Despite these relative improvements, absolute scores remain low: around 20% for genus and 50% for

connected components, highlighting that both models only weakly encode topology. This supports the view that current pretraining objectives, which focus on geometric reconstruction or semantic alignment, do not drive models to capture global connectivity. While deeper layers improve aggregation and invariance, they remain sensitive to geometric details rather than structural invariants. These results reinforce that topology-aware objectives or explicit topological supervision may be required to make 3D encoders truly aware of shape structure beyond geometry.

4.2 Subspace Alignment with Persistent Homology

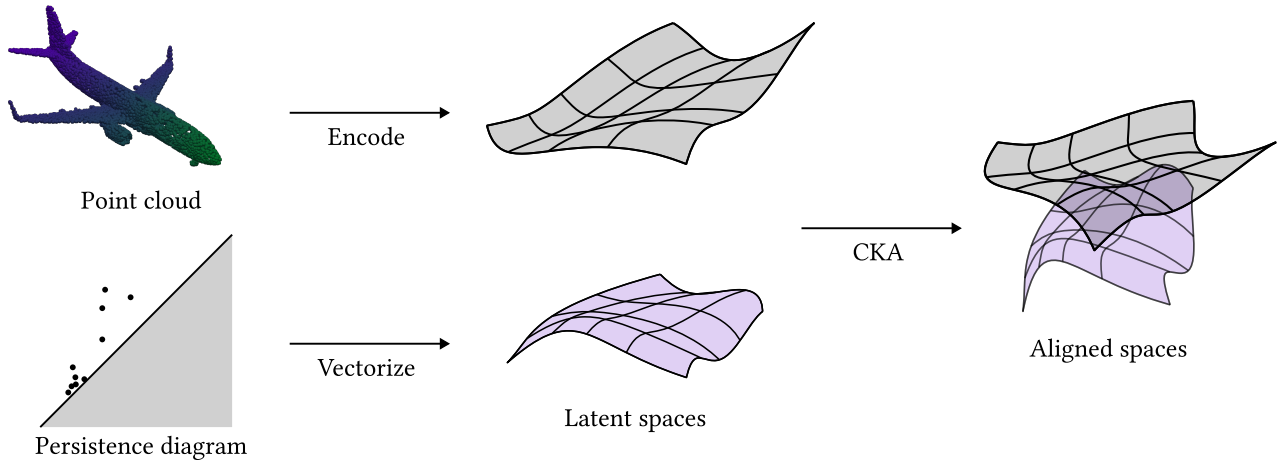


Figure 14: **Alignment procedure overview.** We extract features from a 3D point cloud with the model we seek to evaluate. We also vectorize its corresponding persistence diagram (Section 4.2.2). We then compute the CKA similarity (Section 4.2.3) between the two sets of features to quantify how well the learned representation aligns with topological signatures.

We study the relationship between representations learned by 3D shape encoders and those derived from persistent homology. Following the perspective of the Platonic Representation Hypothesis [19, 25], we view learned embeddings as potentially containing subspaces that capture intrinsic structural information. Information captured by persistent homology can also be treated as a complementary modality. Assuming there's a way to encode topological features into a vector space, we can then ask whether the learned and topological subspaces align. It's worth noticing that the analogy with [25] is not perfect: while it's assumed that representations tend to converge at scale, topological features, even learned ones, aren't the result of large scale training.

We first review background on persistent homology and its vectorization. We then formally define the metrics used to quantify alignment. Finally, we describe our alignment protocol and present results across encoders and datasets.

4.2.1 Persistence Diagrams

Filtrations. Let $(X, \leq)$ be a topological space equipped with a real index. A (real) filtration is a family of subspaces $(X_a)_{a\in\mathbb{R}}$ with $X_a \subseteq X_b$ whenever $a \leq b$ and $\bigcup_a X_a = X$. Typical examples are sublevel sets of a function $f : X \to \mathbb{R}$, namely $X_a = f^{-1}((-\infty, a])$.

Homology and persistent maps. Fix a field $\Bbbk$ and compute singular or simplicial homology with coefficients in $\Bbbk$. For $k \geq 0$ and $a \leq b$, the inclusion $X_a \hookrightarrow X_b$ induces a linear map

$$i_a^b : H_k(X_a; \Bbbk) \longrightarrow H_k(X_b; \Bbbk). \tag{22}$$

A k-dimensional class $\alpha \in H_k(X_b)$ is *born* at the smallest a for which it has a representative in $H_k(X_a)$, and it *dies* at the smallest $d > b$ where it maps to zero in $H_k(X_d)$ under i_b^d. Classes that never die are called *essential*.

Persistence modules, barcodes and diagrams. The collection $\{H_k(X_a), i_a^b\}_{a \le b}$ is a persistence module. Under standard finiteness conditions (e.g. f tame, or finite type filtrations), it decomposes into interval modules. This gives a multiset of intervals (the barcode) or equivalently a multiset of points (b_i, d_i) in the open half plane $\{(x, y) \in \mathbb{R}^2 : x < y\}$ together with the diagonal $\{(t, t)\}$ available for matching with infinite multiplicity. This multiset is the k-th persistence diagram $\mathrm{Dgm}_k(X)$.

Why a persistence diagram is not a Euclidean vector object. A persistence diagram is a collection of points in the plane, where each point (b, d) records the birth and death of a topological feature during a filtration. Points on the diagonal $(b = d)$ correspond to features that disappear immediately, and we assume there are infinitely many of them so that every feature in one diagram can be matched to a trivial one in another when comparing diagrams. Unlike vectors in Euclidean space, persistence diagrams cannot be naturally added or scaled, since such operations would destroy their topological meaning. Even computing an average of several diagrams can give multiple different results. In some cases, a diagram may even have infinitely many points away from the diagonal if the underlying space behaves badly. Therefore, persistence diagrams are treated as elements of a metric space, with distances defined by measures such as the bottleneck or Wasserstein metric, rather than as vectors in a linear space.

Distances between diagrams. Let D and E be diagrams. We allow matchings to the diagonal. The *bottleneck distance* is

$$d_B(D, E) \;=\; \inf_{\gamma : D \to E} \sup_{x \in D} \|x - \gamma(x)\|_\infty. \tag{23}$$

For $1 \le p < \infty$, the *p-Wasserstein distance* is

$$d_{W,p}(D, E) \;=\; \left(\inf_{\gamma : D \to E} \sum_{x \in D} \|x - \gamma(x)\|_\infty^p \right)^{1/p}. \tag{24}$$

These are the standard and stable distances on persistence diagrams. The plain Hausdorff distance between the underlying point sets does not account for multiplicities and diagonal matchings and is not used in modern stability theory.

Stability theorems. For sublevel set filtrations of functions $f, g : X \to \mathbb{R}$ on a common triangulable space, the diagrams are stable under perturbations of f:

$$d_B\big(\mathrm{Dgm}_k(f), \mathrm{Dgm}_k(g)\big) \;\le\; \|f - g\|_\infty. \tag{25}$$

Related bounds hold for $d_{W,p}$. In the algebraic language of persistence modules, the Isometry Theorem states that for q-tame modules M, N,

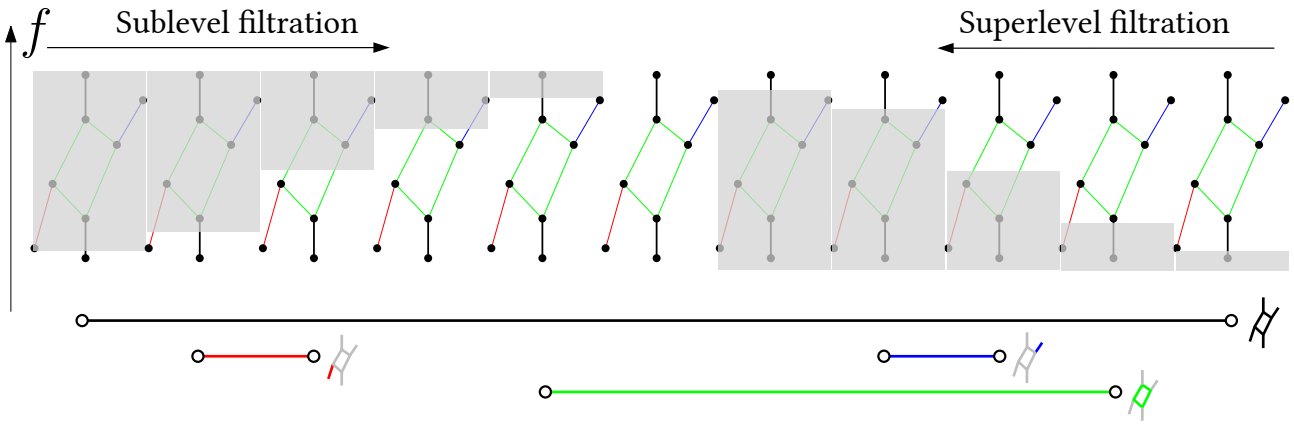
$$d_B\big(\mathrm{Dgm}(M), \mathrm{Dgm}(N)\big) \;=\; d_I(M, N), \tag{26}$$

where d_I is the interleaving distance.

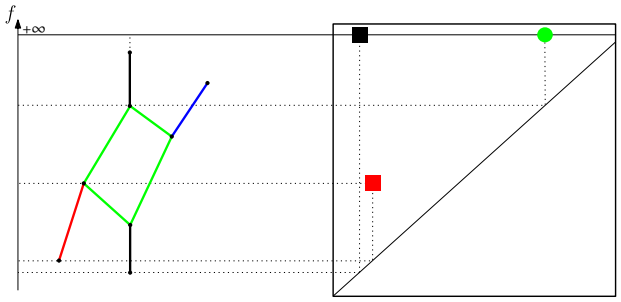
Extended persistence. Essential classes can lead to pairs with infinite lifetime. Extended persistence remedies this by combining sublevel and superlevel filtrations and using relative homology. For a function $f : X \to \mathbb{R}$ on a compact space, consider the sublevel filtration (X_a) and the superlevel filtration $(X^a = f^{-1}([a, \infty)))$. One constructs a zigzag that passes from absolute to relative homology so that every class is paired at a finite parameter value. The resulting *extended persistence diagram* has only finite pairs and encodes essential features as finite points. Figure 15 illustrates the filtration process as well as the difference between ordinary and extended persistence on a simple graph.

4.2.2 Vectorization of Persistence Diagrams

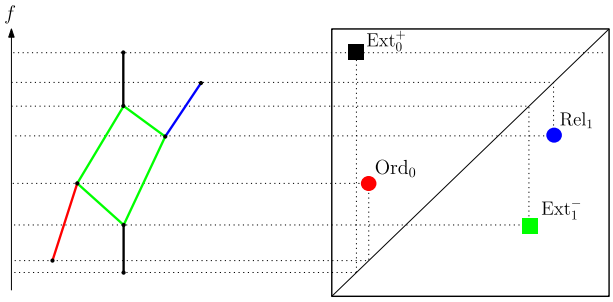
Vectorizing persistence diagrams is crucial for incorporating topological signals in standard learning pipelines. A persistence diagram is a multiset of points in the plane. It does not live in a vector space and is usually



(a) Sub- (resp. super-) level set filtration.



(b) Ordinary persistence diagram.



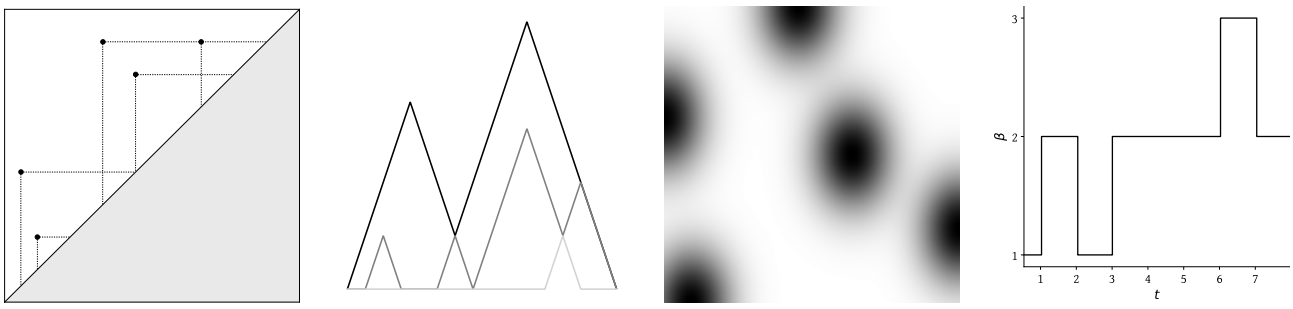
(c) Extended persistence diagram.

Figure 15: **Ordinary vs extended persistence.** (Figures adapted from [9]). In 15a, each node of the graph is assigned its height. Persistence intervals are shown under the sequence. In 15b, ordinary persistence captures connected components and loops, but essential classes lead to infinite intervals (black and green markers) and the upward branch (blue) isn't captured. In 15c, extended persistence pairs all classes at finite values by combining sublevel and superlevel filtrations with relative homology. Finally, Ext_0^+, Ext_1^-, Ord_0 and Ord_1 denote the different types of pairs in extended persistence.

compared with bottleneck or Wasserstein distances rather than inner products. Most learning algorithms expect fixed size vectors or a Hilbert space structure, so raw diagrams are not a natural input. Diagrams also have variable size and are permutation invariant, which complicates batching and optimization. These issues motivate stable feature maps and kernels that embed diagrams into Euclidean

Prescribed Vectorization. Handcrafted summaries (Figure 16) have long been the dominant strategy for vectorizing persistence diagrams in machine learning. The most common approaches include persistence landscapes [6], Betti curves [18], and persistence images [1] (Figure 16). Landscapes embed diagrams in a Hilbert space and enable classical statistics, but they still require design choices such as the number of landscape levels and sampling resolution. Betti curves collapse a diagram to simple counting functions over the filtration. Persistence images rasterize diagrams with a kernel on a fixed grid. These methods are stable in theory but depend on key hyperparameters such as grid resolution and kernel width. These choices must be tuned per task and can limit expressiveness and generalization. Finally, ATOL [42] proposes an unsupervised alternative. It vectorizes measures, including sets of persistence points, by first clustering with K-means and then summarizing cluster statistics in a fixed length descriptor. The pipeline is simple and fast but remains sensitive to the chosen number of clusters and to the distribution shift between training and test diagrams.

Learned Vectorization. Recent work has moved from handcrafted features to learning based approaches [35],[21] that act on persistence diagrams more directly, often through extended persistence [11] (see Section ...). PersLay [9] introduces a neural layer that learns stable weights and pooling functions over points in a diagram. It was developed for graph tasks and relies on extended persistence to encode informative signatures before the learned aggregation. PLLay [26] follows a hybrid route. It first computes persistence landscapes and then applies a differentiable neural layer that learns how to weight and combine landscape coordinates. This reduces manual feature design while keeping the stability benefits of landscapes. Transformer based designs have also appeared. Persformer [41] treats each persistence point as a token and applies standard



(a) Persistence Diagram (b) Persistence Landscape (c) Persistence Image (d) Betti Curve

Figure 16: **Common prescribed vectorizations of a persistence diagram.** From left to right: (a) A sample persistence diagram with points representing topological features; (b) Persistence landscape capturing the prominence of features across scales; (c) Persistence image providing a smoothed, grid-based representation; *Note:* Instead of considering persistence pairs as a *(birth,death)* tuple, persistence image is fitted on $(birth, death - birth)$ pairs. (d) Betti curve showing the count of features over filtration values.

self attention. This is flexible but scales poorly as the number of points grows. To address scalability, the Multiset Transformer [47] clusters points and modifies attention to account for multiplicities, so that a cluster can be handled as a single item with weight greater than one. This change reduces memory and time while preserving permutation invariance at the multiset level. xPerT [27] goes further by leveraging sparsity with a *pixelized diagram* representation. It is close in spirit to persistence images but does not require a smoothing kernel. The pixeliaation allows efficient tokenization and improves scalability compared with Persformer. xPerT also works with extended persistence; details are deferred to the dedicated section.

Across these learning based methods, a common trait is that the diagram is manipulated explicitly rather than replaced by coarse handcrafted summaries. Most results to date are on graph benchmarks rather than 3D shape understanding. This gap limits conclusions for point cloud encoders trained in a self supervised way, where task independent evaluation is required.

Finally, recent topology aware 3D generation pipelines [22] directly encode persistence pairs. More precisely they feed the top k most persistent pairs as $\{g_i = (b_i, d_i - b_i)\}_{i \in [\![1,k]\!]}$ where b_i is the birth time and d_i is the death time of the i-th pair. These papers report increased diversity or topology control but are typically demonstrated on a few ShapeNet categories or related datasets. A representative example conditions a latent diffusion model on topological features computed from persistence diagrams and validates on ShapeNet and ABC.

4.2.3 Subspace Alignment Protocol

Background To assess whether a learned 3D encoder's internal representations align with topological signatures derived from persistent homology, we adopt Centered Kernel Alignment (CKA) as our similarity metric. CKA has become a standard tool in recent work for comparing internal neural representations, because it overcomes certain limitations of canonical correlation analysis (CCA) in high-dimensional settings and provides a normalized similarity score between two representational spaces [29].

Let $X \in \mathbb{R}^{n \times d_X}$ and $Y \in \mathbb{R}^{n \times d_Y}$ be two sets of centered features across n examples (rows). We form Gram (kernel) matrices $K = XX^\top$ and $L = YY^\top$. Let $H = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ be the centering matrix. The (biased) Hilbert–Schmidt independence measure is:

$$\mathrm{HSIC}(K, L) = \frac{1}{(n-1)^2}\mathrm{trace}(KHLH). \tag{27}$$

The CKA similarity is then defined as:

$$\mathrm{CKA}(X, Y) = \frac{\mathrm{HSIC}(K, L)}{\sqrt{\mathrm{HSIC}(K, K)\mathrm{HSIC}(L, L)}}. \tag{28}$$

This normalization ensures that $\mathrm{CKA}(X, Y) \in [0, 1]$ and is invariant to isotropic scaling of features. An unbiased estimator of HSIC can also be used to mitigate bias induced by finite samples or mini-batching.

We choose CKA for several reasons. First, its normalization makes the similarity more interpretable across different layer sizes or vectorization dimensions. Second, in empirical studies, CKA has proven effective at identifying correspondences between layers of differently initialized or architected networks (while pure CCA-type metrics sometimes fail). Third, because it works by comparing inner-product (kernel) structures, it is well suited to capture alignment between representational geometries rather than mere coordinate-wise correlation. Finally, in our context, since the topological encoding (via vectorized persistence diagrams) and the learned features live in distinct vector spaces, using a kernel-based alignment provides a flexible and robust way to detect shared structure, even if the bases or dimensions differ.

Thus in our protocol, we treat a vectorization of the persistence diagram as one representational matrix Y, and the activations extracted from a given layer of the 3D encoder as X. Then we compute $\mathrm{CKA}(X, Y)$ to quantify how well that encoder subspace aligns with the topological modality.

Experimental setting.

Results To be completed.

4.3 Changing the Pretraining Data Distribution

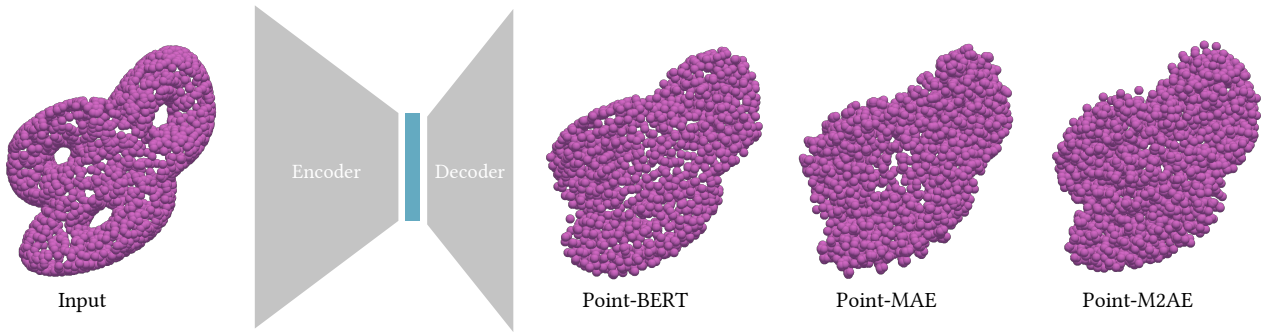


Figure 17: **Point-cloud reconstruction for different encoders.** Each pretrained model is fed with a point cloud with 2048 points. We reconstruct it with the decoder used during pretraining with a masking ratio of zero. To ensure topological structures of the input are captured, we patchify the input point cloud into overlapping local regions before encoding, and decode to a reconstructed point cloud of 4096 points. Reconstructed point-clouds don't exhibit the 3 characteristic holes of the input shape.

Previous experiments showed that current 3D encoders have a limited understanding of topological structures. This section investigates a simple hypothesis. In data-driven approaches, models learn to extract information that is relevant to the pretraining objective. All the considered encoders are pretrained on reconstruction tasks using the Chamfer loss. These tasks rely only on geometric proximity between predicted and target point clouds, without explicitly rewarding structural consistency. Moreover, the pretraining data itself lacks topological diversity. Datasets such as ShapeNet mostly contain single, compact objects with few or no holes. As a result, encoders can achieve low reconstruction errors without learning to capture non-trivial topological properties. This experiment is further motivated by Figure 17, which shows that pretrained encoders struggle to reconstruct even simple topological structures.

We hypothesize that changing the pretraining data distribution can improve the model's structural understanding. By exposing encoders to shapes with richer and more diverse topology, they may learn internal representations that better capture global structure. To test this idea, we pretrain all the previously evaluated encoders on the ABC dataset. This dataset contains CAD models with a wide range of geometric and topological configurations. For a fair comparison with models pretrained on ShapeNet, which contains 51K samples, we randomly select 55K shapes from ABC. We strictly follow the same pretraining procedure, including the reconstruction objective and optimization settings.

To evaluate this hypothesis, we assess two key aspects: (1) whether the models retain their performance on standard downstream tasks, and (2) whether they show improved structural understanding. We follow the evaluation protocols introduced in the original papers of each encoder.

Method	OBJ-BG	OBJ-ONLY	PB-T50-RS
Point-BERT [57]	87.43	88.12	83.07
Point-BERT (ABC)	88.30	88.47	83.55
difference	+0.87	+0.35	+0.48
Point-MAE [36]	90.02	88.29	85.18
Point-MAE (ABC)	88.64	88.47	85.92
difference	−1.38	+0.18	+0.74
PM2AE [59]	91.22	88.81	86.43
PM2AE (ABC)	90.02	89.67	–
difference	−1.2	+0.86	–
PCP-MAE [60]	95.52	94.32	90.35
PCP-MAE (ABC)	95.08	94.32	89.80
difference	−0.44	–	−0.55

Table 3: **Classification results on ScanObjectNN.** For each backbone model, we compare performance when pretrained on ShapeNet (first row) versus ABC (second row). The third row shows the difference, with green indicating improvement and orange indicating decline when using ABC.

Object Classification. We first verify that pretraining on ABC does not degrade general performance on standard semantic tasks. Object classification is performed on the ScanObjectNN dataset, a real-world benchmark built from scanned indoor objects. It contains 15,000 samples across 15 categories, organized into three variants of increasing difficulty: obj only, which includes cleanly segmented objects; obj bg, where background points are kept; and PB T50 RS, which introduces heavy noise and random perturbations. This dataset is considered challenging because it captures real-world imperfections such as occlusions and partial views, unlike synthetic datasets like ShapeNet.

As shown in Table 3, models pretrained on ABC perform similarly or slightly better than those pretrained on ShapeNet. This suggests that changing the pretraining distribution to CAD models whose semantic content differs from the finetuning dataset does not harm performance on semantic classification. In fact, exposure to a more topologically varied pretraining set may even promote better generalization.

Few-Shot Classification. We further evaluate the learned representations under few-shot settings. Following the standard protocol, we perform few-shot classification on ModelNet40, which contains 12,311 clean CAD models from 40 categories, and on DONUT, which includes point clouds with richer topology, such as multiple connected components and non-trivial holes. Few-shot experiments are conducted with 5 or 10 ways (number of selected categories), 10 or 20 shots (number of samples per class), and tested on 20 samples per class.

Results in Table 4 show that performance on ModelNet40 remains close across both pretraining datasets, while models pretrained on ABC consistently achieve better results on DONUT (except for the 10-way 20-shot setting on Point-MAE). The overall low accuracy on DONUT, however, confirms that this task remains non-trivial and highlights the current models' limited structural understanding.

5 Conclusion

This work investigated whether modern transformer-based 3D encoders capture the topological structure of shapes. Using DONUT, a controlled benchmark we specifically designed for this purpose, we were able to quantify how much these models understand intrinsic topological features such as connected components and genus. Across all analyses and architectures, results consistently showed that current 3D transformers lack a clear understanding of topology. Their representations reflect geometric and semantic cues but fail to encode structural invariants that define the global organization of a shape.

This work investigated whether modern transformer-based 3D encoders capture the topological structure

	ModelNet				DONUT (genus)			
	5-way		10-way		5-way		10-way	
Methods	10-shot	20-shot	10-shot	20-shot	10-shot	20-shot	10-shot	20-shot
Point-BERT [57]	$94.6_{\pm 3.1}$	$96.3_{\pm 2.7}$	$91.0_{\pm 5.4}$	$92.7_{\pm 5.1}$	$34.4_{\pm 4.4}$	$36.5_{\pm 4.0}$	$19.2_{\pm 2.0}$	$21.8_{\pm 2.0}$
Point-BERT (ABC)	$96.9_{\pm 2.5}$	$98.5_{\pm 1.4}$	$91.6_{\pm 4.6}$	$94.3_{\pm 3.9}$	$34.9_{\pm 5.4}$	$37.6_{\pm 5.0}$	$20.2_{\pm 2.3}$	$22.4_{\pm 1.5}$
difference	+2.3	+2.2	+0.6	+1.6	+0.5	+1.1	+1.0	+0.6
Point-MAE [36]	$96.3_{\pm 2.5}$	$97.8_{\pm 1.8}$	$92.6_{\pm 4.1}$	$95.0_{\pm 3.0}$	$32.9_{\pm 3.5}$	$38.1_{\pm 3.9}$	$20.4_{\pm 2.1}$	$23.8_{\pm 1.7}$
Point-MAE (ABC)	$96.2_{\pm 2.7}$	$97.6_{\pm 1.9}$	$92.3_{\pm 4.4}$	$95.2_{\pm 2.9}$	$35.1_{\pm 6.0}$	$40.1_{\pm 4.9}$	$21.4_{\pm 2.7}$	$22.8_{\pm 2.4}$
difference	−0.1	−0.2	+0.3	+0.2	+2.2	+2.0	+1.0	−1.0
PM2AE [59]	$96.8_{\pm 1.8}$	$98.3_{\pm 1.4}$	$92.3_{\pm 4.5}$	$95.0_{\pm 3.0}$	$34.7_{\pm 4.2}$	$41.8_{\pm 6.3}$	$21.2_{\pm 2.2}$	$25.0_{\pm 1.3}$
PM2AE (ABC)	$95.9_{\pm 2.4}$	$97.7_{\pm 1.8}$	$91.1_{\pm 4.6}$	$94.4_{\pm 3.4}$	$37.0_{\pm 4.7}$	$43.4_{\pm 5.3}$	$22.8_{\pm 1.7}$	$26.3_{\pm 2.3}$
difference	−0.9	−0.6	−1.2	−0.6	+2.3	+1.8	+1.6	+1.3
PCP-MAE [60]	$97.4_{\pm 2.3}$	$99.1_{\pm 0.8}$	$93.5_{\pm 3.7}$	$95.9_{\pm 2.7}$	$36.6_{\pm 5.4}$	$41.8_{\pm 4.0}$	$21.5_{\pm 2.0}$	$24.6_{\pm 1.7}$
PCP-MAE (ABC)	$96.1_{\pm 3.1}$	$98.3_{\pm 4.3}$	$91.7_{\pm 4.3}$	$94.9_{\pm 2.9}$	$38.7_{\pm 4.2}$	$44.5_{\pm 5.4}$	$22.4_{\pm 1.9}$	$26.0_{\pm 1.4}$
difference	−1.3	−0.8	−1.8	−1.0	+2.1	+2.7	+0.9	+1.4

Table 4: **Few-shot object classification on ModelNet40 and DONUT (genus).** We conduct 10 independent experiments for each setting and report mean accuracy (%) with standard deviation. For each backbone model, we compare performance when pretrained on ShapeNet (first row) versus ABC (second row). The third row shows the difference, with green indicating improvement and orange indicating decline when using ABC.

of shapes. Using DONUT, a controlled benchmark we specifically designed for this purpose, we were able to quantify how much these models understand intrinsic topological features such as connected components and genus. Across all analyses and architectures, results consistently showed that current 3D transformers lack a clear understanding of topology. Their representations reflect geometric and semantic cues but fail to encode structural invariants that define the global organization of a shape.

These findings point to a broader limitation of current 3D representation learning approaches. When moving beyond curated benchmarks to real-world data, we often encounter shapes that do not have clear semantic labels or stable geometric parametrizations but share common topological patterns. In such cases, models that ignore topology risk missing essential structural information. Bridging topological data analysis with deep learning, as explored here, offers a principled way to address this gap and build representations that capture not only what a shape looks like but also how it is fundamentally organized.

A Supplementary Material for DONUT

This section first provides an in-depth analysis of the EuLearn dataset [15], to justify why it's not reliable for our experiments. Then, it details the algorithms and proofs related to the DONUT dataset generation process.

A.1 EuLearn Dataset Analysis

A key issue is the entanglement between topology and orientation. As shown in Figure 19, a simple k-nearest neighbor classifier trained on point clouds in their canonical orientation achieves near-perfect accuracy. However, accuracy collapses when test samples are randomly rotated, despite genus being a topological invariant. This indicates that classifiers exploit orientation-dependent cues instead of genuine topological structure.

The same effect appears at the representation level. Figure 18 shows UMAP projections of Point-BERT embeddings. When evaluated on canonical orientations, embeddings form tight clusters by genus. After applying random rotations, this structure disappears entirely. If embeddings reflected topology, genus separation should persist under rigid transformations; instead, the disappearance of clusters confirms that the genus signal in EuLearn is largely a proxy for global pose and correlated geometric features.

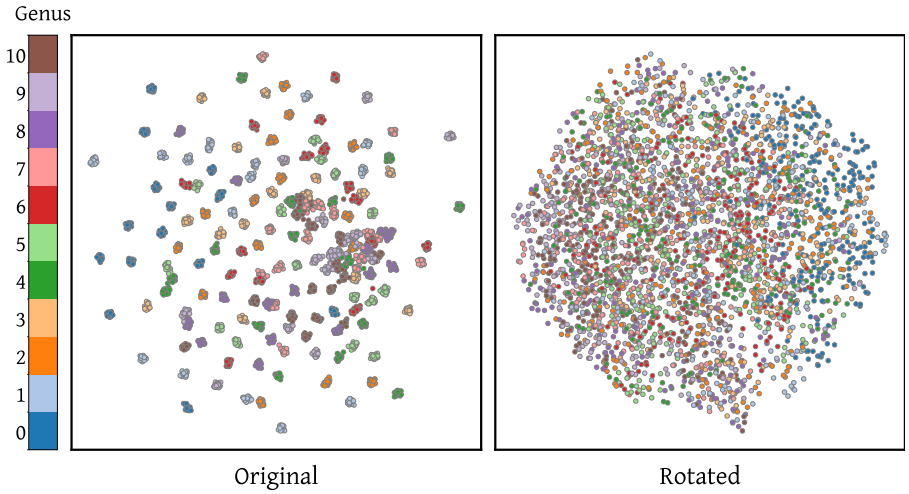


Figure 18: **UMAP projections of Point-BERT embeddings colored by genus.** Clear clusters by genus appear for canonical orientations (left) but vanish under random rotations (right), showing that genus separation in EuLearn is tied to orientation rather than topology.

Linear probing experiments further reinforce this point (Table 20). Probes trained on canonical embeddings generalize only to canonical test shapes, collapsing under rotations. Probes trained on rotated embeddings avoid orientation bias but fail to recover meaningful genus structure, with accuracies barely above random chance. In both cases, the embeddings overfit to spurious correlations, not topology.

Taken together, these results demonstrate that EuLearn's construction introduces systematic biases. The dataset does not allow disentangling true topological reasoning from pose-dependent shortcuts, making it unsuitable for evaluating structural understanding in 3D shape encoders.

A.2 Sampling Labels

As mentioned in Section 2.2.1, labels (number of connected components β_0 and genus g) are sampled prior to generating any mesh. This step is key as it is where we ensure marginal distributions of both labels to be approximately uniform. The sampling procedure is described below. The algorithm takes as input the minimum and maximum number of connected components β_0^{min} and β_0^{max}, the maximum genus per component g^{max}, the maximum genus per sample (group of components) G^{max} and a parameter k that controls the number of samples per value of β_0. It outputs a list of (β_0, g) pairs, with $\beta_0 \sim \mathcal{U}[\![\beta_0^{min}, \beta_0^{max}]\!]$ and $g \sim \mathcal{U}[\![0, G^{max}]\!]$ subject to the constraint that $g \leq \min(G^{max}, \beta_0 \times g^{max})$. The constraint encodes that a single component can't have a genus higher than g^{max}, and that the total genus of a sample cannot exceed G^{max}.

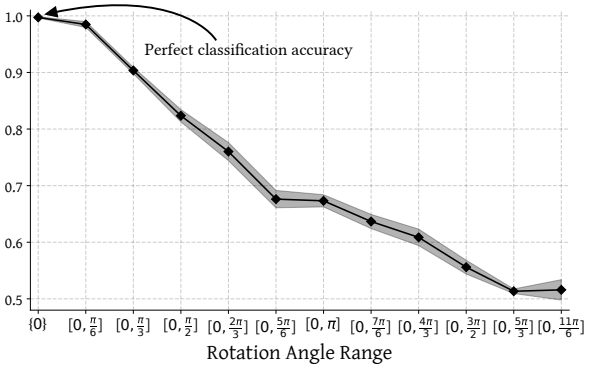


Figure 19: **Classification accuracy of a k-nearest neighbor classifier vs. training rotation range.** Accuracy is perfect when training and testing on canonical orientations but collapses as random rotations along the z-axis are introduced, revealing orientation-dependent bias in EuLearn. *Note:* Results are cross-validated with 5 folds.

Training set	Test set	Training Acc.	Test Acc.
		99.9	$96.4_{(-0.5)}$
			$16.3_{(-83.6)}$
		77.8	$50.1_{(-27.7)}$
			$45.9_{(-31.9)}$

Figure 20: **Linear probing accuracy on Point-BERT embeddings of canonical vs. rotated shapes.** Probes trained on canonical orientations () achieve high accuracy only on canonical test data, collapsing under rotation. (). Probes trained on rotated shapes generalize poorly in both cases, confirming that EuLearn does not provide a stable signal for genus.

Role of k. Choosing exactly k times each value of β_0 ensures that the marginal distribution of β_0 is perfectly uniform. However, since the number of valid genus values depends on β_0, the marginal distribution of g is only approximately uniform.

Algorithm 1 Sampling (β_0, g)
Input: $g^{\max}$, $G^{\max}$, $\beta_0^{\min}$, $\beta_0^{\max}$, k

let $\mathfrak{B}_0 \leftarrow \{\underbrace{\beta_0^{min}, \ldots, \beta_0^{min}}_{\times k}, \underbrace{\beta_0^{min}+1}_{\times k}, \ldots, \underbrace{\beta_0^{max}}_{\times k}\}$
initialize output list $P \leftarrow []$
for all $\beta_0 \in \mathfrak{B}_0$ **do**
  $g_{max} \leftarrow \min\left(G^{max},\ \beta_0 \cdot g^{max}\right)$
  accepted $\leftarrow$ false
  while not accepted **do**
    sample $s \sim \mathcal{U}[\![0, G^{max}]\!]$
    if $s \leq g_{\max}$ **then**
      accepted $\leftarrow$ true
    end if
  end while
  append (β_0, s) to P
end for
return P

A.3 Choosing Components Shapes

Once the labels are sampled, the next step is to choose the actual shapes composing each sample. This is done by decomposing the global genus g into β_0 individual genera $(g_1, \ldots, g_{\beta_0})$ such that $\sum_{i=1}^{\beta_0} g_i = g$. Each g_i is then associated with a parametric family of shapes (Section 2.2.2). However, given a couple (β_0, g) several shapes configurations can lead to the same global labels (Figure ...). We leverage this observation to add more variability within the dataset, by first getting all the possible combinations of shapes for a given label configuration. The problem can be formulated as follows:

Hyperparameter	Value
$g^{\max}$	5
$G^{\max}$	10
$\beta_0^{\min}$	1
$\beta_0^{\max}$	6
k	1000

Table 5: Hyperparameter values used for the DONUT dataset for the experiments. Probing tasks boil down to classifying shapes across $\beta_0^{\max} - \beta_0^{\min} + 1 = 6$ categories for connected components and 11 categories for genera.

Problem statement: *We're given three integers: a (total items), b (target weighted sum) and g^{max} (maximum template index). We also have templates numbered from 0 to g^{max} (included). Template i has a weight of i.*
We want to find all the possible ways to distribute exactly a items across these templates such that the weighted sum $\sum_{i=0}^{g^{max}} i \cdot x_i = b$, where x_i is the number of items assigned to template i.
Input:

- a: Total number of items to distribute
- b: Target weighted sum
- g^{max}: Maximum template index (templates are $0, 1, 2, \ldots, g^{max}$)

Output: A list of all possible distributions $(x_0, x_1, \ldots, x_{g^{max}})$.

Example:
Input: $a = 3,\ b = 5,\ g^{max} = 3$
Output: $\big[[0, 2, 0, 1], [1, 0, 1, 1], [0, 1, 2, 0]\big]$
Explanation:

- $[0, 2, 0, 1]$: $0 \cdot 0 + 2 \cdot 1 + 0 \cdot 2 + 1 \cdot 3 = 3$, and total count $0 + 2 + 0 + 1 = 3$
- $[1, 0, 1, 1]$: $1 \cdot 0 + 0 \cdot 1 + 1 \cdot 2 + 1 \cdot 3 = 5$, and total count $1 + 0 + 1 + 1 = 3$
- $[0, 1, 2, 0]$: $0 \cdot 0 + 1 \cdot 1 + 2 \cdot 2 + 0 \cdot 3 = 5$, and total count $0 + 1 + 2 + 0 = 3$

In the context of DONUT, a corresponds to the number of connected components β_0, b to the total genus g and g^{max} to the maximum genus per component. The output distributions $(x_0, x_1, \ldots, x_{g^{max}})$ indicate how many components of each genus should be used to form a sample with the desired labels. For instance, if $\beta_0 = 3$, $g = 5$ and $g^{max} = 3$, one valid output is $(1, 0, 1, 1)$ which means that the sample should be composed of one genus-0 shape, zero genus-1 shapes, one genus-2 shape and one genus-3 shape. *Note* that G^{max} isn't directly involved here, as it's been used already to choose the values of g (Algorithm 1).

Algorithmic Details. Even though we seek for efficient algorithms to ensure the scalability of DONUT, finding all the combinations requires a tree search algorithm with backtracking. This procedure has exponential () complexity. However, since both β_0^{max} and G^{max} are small, enumerating all the solutions given (β_0, g) requires a negligible amount of time compared to generating the actual meshes and composing them together. Once all the solutions are enumerated, we randomly sample one of them to generate the actual meshes. *Note* that to avoid useless computations, we cache the solutions for each (β_0, g) couple, so that if the same couple is encountered again, we can directly sample from the already computed solutions.

Complexity. A naïve approach of the algorithm's complexity is $O(a^{g^{max}+1})$, since at each of the $g^{max} + 1$ recursion levels, we can have up to a branches. However, this bound is very loose, because the additional

constraint on the weighted sum b prunes most fo the branches: many partial assignments can't possbbly satisfy both the total count and the target sum, and are therefore discarded early. The algorithm actually enumerates *weak combinations* [43]

Algorithm 2 ENUMERATE-SOLUTIONS
Input: a, b, g^{max}
Output: S

$S \leftarrow \emptyset$ ▷ Initialize solution set
if $b < 0$ **or** $b > g^{max} \cdot a$ **then**
  return S
end if
BACKTRACK$(a, b, 0, \emptyset, S)$ ▷ Start recursive enumeration
return S

Algorithm 3 BACKTRACK
Input: r_{count}, r_{sum}, k, $\mathbf{x}$, S
Output: S

if $k > g_{\max}$ **then** ▷ Base case: all template types processed
  if $r_{\text{count}} = 0$ **and** $r_{\text{sum}} = 0$ **then**
    $S \leftarrow S \cup \{\mathbf{x}\}$
  end if
  return
end if ▷ Calculate upper bound for current template type
if $k = 0$ **then**
  $u_k \leftarrow r_{\text{count}}$
else
  $u_k \leftarrow \min(r_{\text{count}}, \lfloor r_{\text{sum}}/k \rfloor)$
end if ▷ Try all feasible counts for template type k
for $n_k = 0$ **to** u_k **do**
  $\mathbf{x}' \leftarrow \mathbf{x} \cup \{n_k\}$
  BACKTRACK$(r_{\text{count}} - n_k,\ r_{\text{sum}} - k \cdot n_k,\ k+1,\ \mathbf{x}',\ S)$
end for

A.4 Baselines Training Details

Table 6 provides the training details for the baselines presented in Table 2. Every model is trained on point clouds containing 4096 points. This number is chosen because each mesh in the DONUT dataset is pre-sampled offline with 8192 points. Using 4096 points provides a dense enough representation to capture the underlying topological structures while enabling random resampling from the original 8192-point clouds. This resampling reduces the risk of overfitting to specific point locations and improves the robustness of the learned representations.

A.5 Additional Results

We complete the results provided in Table 2 with per-class metrics for each baseline.

References

[1] Henry Adams et al. *Persistence Images: A Stable Vector Representation of Persistent Homology*. 2016. arXiv: 1507.06217 [cs.CG]. URL: https://arxiv.org/abs/1507.06217.

Hyperparameter	Point-based models			Topology-based models		
	PointNet	PointNet++	DGCNN	PersFormer	xPerT	PersLay
Batch size	32	32	32	64	–	–
Learning rate	0.001	0.001	0.005	0.0005	–	–
Optimizer	Adam	Adam	Adam	AdamW	–	–
Weight decay	1e-4	1e-4	1e-4	5e-5	–	–
Dropout	0.3	0.3	0.5	0.5	–	–
Training epochs	200	200	250	300	–	–

Table 6: Comparison of typical training hyperparameters for point-based models (PointNet, PointNet++, DGCNN) and topology-based models (PersFormer, xPerT, PersLay). A vertical line separates the two families of architectures.

[2] Guillaume Alain and Yoshua Bengio. *Understanding intermediate layers using linear classifier probes*. 2018. arXiv: 1610.01644 [stat.ML]. URL: https://arxiv.org/abs/1610.01644.

[3] Rubén Ballester et al. *MANTRA: The Manifold Triangulations Assemblage*. 2025. arXiv: 2410.02392 [cs.LG]. URL: https://arxiv.org/abs/2410.02392.

[4] Alan H. Barr. "Superquadrics and Angle-Preserving Transformations". In: *IEEE Computer Graphics and Applications* 1.1 (1981), pp. 11–23. DOI: 10.1109/MCG.1981.1673799.

[5] Rickard Brüel-Gabrielsson et al. *A Topology Layer for Machine Learning*. 2020. arXiv: 1905.12200 [cs.LG]. URL: https://arxiv.org/abs/1905.12200.

[6] Peter Bubenik. *Statistical topological data analysis using persistence landscapes*. 2015. arXiv: 1207.6437 [math.AT]. URL: https://arxiv.org/abs/1207.6437.

[7] Mathieu Carrière, Steve Y. Oudot, and Maks Ovsjanikov. "Stable Topological Signatures for Points on 3D Shapes". In: *Symposium on Geometry Processing*. 2015.

[8] Mathieu Carrière et al. *Optimizing persistent homology based functions*. 2021. arXiv: 2010.08356 [cs.CG]. URL: https://arxiv.org/abs/2010.08356.

[9] Mathieu Carrière et al. *PersLay: A Neural Network Layer for Persistence Diagrams and New Graph Topological Signatures*. 2020. arXiv: 1904.09378 [stat.ML]. URL: https://arxiv.org/abs/1904.09378.

[10] Angel X. Chang et al. *ShapeNet: An Information-Rich 3D Model Repository*. 2015. arXiv: 1512.03012 [cs.GR]. URL: https://arxiv.org/abs/1512.03012.

[11] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. "Extending Persistence Using Poincaré and Lefschetz Duality". In: *Foundations of Computational Mathematics* 9.1 (2009), pp. 79–103. DOI: 10.1007/s10208-008-9027-z.

[12] Matt Deitke et al. *Objaverse: A Universe of Annotated 3D Objects*. 2022. arXiv: 2212.08051 [cs.CV]. URL: https://arxiv.org/abs/2212.08051.

[13] Matt Deitke et al. *Objaverse-XL: A Universe of 10M+ 3D Objects*. 2023. arXiv: 2307.05663 [cs.CV]. URL: https://arxiv.org/abs/2307.05663.

[14] Elisabetta Fedele et al. *SuperDec: 3D Scene Decomposition with Superquadric Primitives*. 2025. arXiv: 2504.00992 [cs.CV]. URL: https://arxiv.org/abs/2504.00992.

[15] Rodrigo Fritz et al. *EuLearn: A 3D database for learning Euler characteristics*. 2025. arXiv: 2505.13539 [cs.CG]. URL: https://arxiv.org/abs/2505.13539.

[16] Thomas Gebhart, Paul Schrater, and Alan Hylton. *Characterizing the Shape of Activation Space in Deep Neural Networks*. 2019. arXiv: 1901.09496 [cs.LG]. URL: https://arxiv.org/abs/1901.09496.

[17] Johan Gielis. "A generic geometric transformation that unifies a wide range of natural and abstract shapes". In: *American Journal of Botany* 90.3 (2003), pp. 333–338. DOI: 10.3732/ajb.90.3.333.

[18] Chad Giusti et al. “Clique topology reveals intrinsic geometric structure in neural correlations”. In: *Proceedings of the National Academy of Sciences* 112.44 (Oct. 2015), 13455–13460. ISSN: 1091-6490. DOI: 10.1073/pnas.1506407112. URL: http://dx.doi.org/10.1073/pnas.1506407112.

[19] Souhail Hadgi et al. *Escaping Plato's Cave: Towards the Alignment of 3D and Text Latent Spaces*. 2025. arXiv: 2503.05283 [cs.CV]. URL: https://arxiv.org/abs/2503.05283.

[20] John Hewitt and Christopher D. Manning. “A Structural Probe for Finding Syntax in Word Representations”. 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)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4129–4138. DOI: 10.18653/v1/N19-1419. URL: https://aclanthology.org/N19-1419/.

[21] Christoph Hofer et al. *Deep Learning with Topological Signatures*. 2018. arXiv: 1707.04041 [cs.CV]. URL: https://arxiv.org/abs/1707.04041.

[22] Jiangbei Hu et al. *Topology-Aware Latent Diffusion for 3D Shape Generation*. 2024. arXiv: 2401.17603 [cs.CV]. URL: https://arxiv.org/abs/2401.17603.

[23] Jingwei Huang, Hao Su, and Leonidas Guibas. *Robust Watertight Manifold Surface Generation Method for ShapeNet Models*. 2018. arXiv: 1802.01698 [cs.CG]. URL: https://arxiv.org/abs/1802.01698.

[24] Jingwei Huang, Yichao Zhou, and Leonidas Guibas. *ManifoldPlus: A Robust and Scalable Watertight Manifold Surface Generation Method for Triangle Soups*. 2020. arXiv: 2005.11621 [cs.GR]. URL: https://arxiv.org/abs/2005.11621.

[25] Minyoung Huh et al. *The Platonic Representation Hypothesis*. 2024. arXiv: 2405.07987 [cs.LG]. URL: https://arxiv.org/abs/2405.07987.

[26] Kwangho Kim et al. *PLLay: Efficient Topological Layer based on Persistence Landscapes*. 2021. arXiv: 2002.02778 [cs.LG]. URL: https://arxiv.org/abs/2002.02778.

[27] Sehun Kim. *xPerT: Extended Persistence Transformer*. 2024. arXiv: 2410.14193 [cs.LG]. URL: https://arxiv.org/abs/2410.14193.

[28] Sebastian Koch et al. *ABC: A Big CAD Model Dataset For Geometric Deep Learning*. 2019. arXiv: 1812.06216 [cs.GR]. URL: https://arxiv.org/abs/1812.06216.

[29] Simon Kornblith et al. *Similarity of Neural Network Representations Revisited*. 2019. arXiv: 1905.00414 [cs.LG]. URL: https://arxiv.org/abs/1905.00414.

[30] Juho Lee et al. *Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks*. 2019. arXiv: 1810.00825 [cs.LG]. URL: https://arxiv.org/abs/1810.00825.

[31] Yangyan Li et al. *PointCNN: Convolution On $\mathcal{X}$-Transformed Points*. 2018. arXiv: 1801.07791 [cs.CV]. URL: https://arxiv.org/abs/1801.07791.

[32] Yang Liu et al. “Regress Before Construct: Regress Autoencoder for Point Cloud Self-supervised Learning”. In: *Proceedings of the 31st ACM International Conference on Multimedia*. MM '23. ACM, Oct. 2023, 1738–1749. DOI: 10.1145/3581783.3612106. URL: http://dx.doi.org/10.1145/3581783.3612106.

[33] Zhijian Liu et al. *Point-Voxel CNN for Efficient 3D Deep Learning*. 2019. arXiv: 1907.03739 [cs.CV]. URL: https://arxiv.org/abs/1907.03739.

[34] Michael Moor et al. *Topological Autoencoders*. 2021. arXiv: 1906.00722 [cs.LG]. URL: https://arxiv.org/abs/1906.00722.

[35] Naoki Nishikawa, Yuichi Ike, and Kenji Yamanishi. *Adaptive Topological Feature via Persistent Homology: Filtration Learning for Point Clouds*. 2023. arXiv: 2307.09259 [cs.LG]. URL: https://arxiv.org/abs/2307.09259.

[36] Yatian Pang et al. *Masked Autoencoders for Point Cloud Self-supervised Learning*. 2022. arXiv: 2203.06604 [cs.CV]. URL: https://arxiv.org/abs/2203.06604.

[37] Charles R. Qi et al. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*. 2017. arXiv: 1706.02413 [cs.CV]. URL: https://arxiv.org/abs/1706.02413.

[38] Charles R. Qi et al. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2017. arXiv: 1612.00593 [cs.CV]. URL: https://arxiv.org/abs/1612.00593.

[39] Zekun Qi et al. *ShapeLLM: Universal 3D Object Understanding for Embodied Interaction*. 2024. arXiv: 2402.17766 [cs.CV]. URL: https://arxiv.org/abs/2402.17766.

[40] Maithra Raghu et al. *Do Vision Transformers See Like Convolutional Neural Networks?* 2022. arXiv: 2108.08810 [cs.CV]. URL: https://arxiv.org/abs/2108.08810.

[41] Raphael Reinauer, Matteo Caorsi, and Nicolas Berkouk. *Persformer: A Transformer Architecture for Topological Machine Learning*. 2022. arXiv: 2112.15210 [cs.LG]. URL: https://arxiv.org/abs/2112.15210.

[42] Martin Royer et al. *ATOL: Measure Vectorization for Automatic Topologically-Oriented Learning*. 2020. arXiv: 1909.13472 [cs.CG]. URL: https://arxiv.org/abs/1909.13472.

[43] Bruce E. Sagan. *Combinatorics: The Art of Counting*. Vol. 210. Graduate Studies in Mathematics. American Mathematical Society, 2020, p. 304. ISBN: 978-1-4704-6032-7.

[44] Daniel Smilkov et al. *SmoothGrad: removing noise by adding noise*. 2017. arXiv: 1706.03825 [cs.LG]. URL: https://arxiv.org/abs/1706.03825.

[45] Hugues Thomas et al. *KPConv: Flexible and Deformable Convolution for Point Clouds*. 2019. arXiv: 1904.08889 [cs.CV]. URL: https://arxiv.org/abs/1904.08889.

[46] Robin Vandaele et al. *Topologically Regularized Data Embeddings*. 2022. arXiv: 2110.09193 [cs.LG]. URL: https://arxiv.org/abs/2110.09193.

[47] Minghua Wang, Ziyun Huang, and Jinhui Xu. *Multiset Transformer: Advancing Representation Learning in Persistence Diagrams*. 2024. arXiv: 2411.14662 [cs.LG]. URL: https://arxiv.org/abs/2411.14662.

[48] Peng-Shuai Wang, Yang Liu, and Xin Tong. "Dual Octree Graph Networks for Learning Adaptive Volumetric Shape Representations". In: *ACM Transactions on Graphics (SIGGRAPH)* 41.4 (2022).

[49] Yue Wang et al. *Dynamic Graph CNN for Learning on Point Clouds*. 2019. arXiv: 1801.07829 [cs.CV]. URL: https://arxiv.org/abs/1801.07829.

[50] Chi-Chong Wong and Chi-Man Vong. "Persistent Homology based Graph Convolution Network for Fine-grained 3D Shape Segmentation". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 7078–7087. DOI: 10.1109/ICCV48922.2021.00701.

[51] Wenxuan Wu, Zhongang Qi, and Li Fuxin. *PointConv: Deep Convolutional Networks on 3D Point Clouds*. 2020. arXiv: 1811.07246 [cs.CV]. URL: https://arxiv.org/abs/1811.07246.

[52] Runsen Xu et al. *PointLLM: Empowering Large Language Models to Understand Point Clouds*. 2024. arXiv: 2308.16911 [cs.CV]. URL: https://arxiv.org/abs/2308.16911.

[53] Le Xue et al. *ULIP-2: Towards Scalable Multimodal Pre-training for 3D Understanding*. 2024. arXiv: 2305.08275 [cs.CV]. URL: https://arxiv.org/abs/2305.08275.

[54] Le Xue et al. *ULIP: Learning a Unified Representation of Language, Images, and Point Clouds for 3D Understanding*. 2023. arXiv: 2212.05171 [cs.CV]. URL: https://arxiv.org/abs/2212.05171.

[55] Zuoyu Yan et al. *Neural Approximation of Graph Topological Features*. 2022. arXiv: 2201.12032 [cs.LG]. URL: https://arxiv.org/abs/2201.12032.

[56] Xuanhua Yin et al. *HFBRI-MAE: Handcrafted Feature Based Rotation-Invariant Masked Autoencoder for 3D Point Cloud Analysis*. 2025. arXiv: 2504.14132 [cs.CV]. URL: https://arxiv.org/abs/2504.14132.

[57] Xumin Yu et al. *Point-BERT: Pre-training 3D Point Cloud Transformers with Masked Point Modeling*. 2022. arXiv: 2111.14819 [cs.CV]. URL: https://arxiv.org/abs/2111.14819.

[58] Longwen Zhang et al. *CLAY: A Controllable Large-scale Generative Model for Creating High-quality 3D Assets*. 2024. arXiv: 2406.13897 [cs.CV]. URL: https://arxiv.org/abs/2406.13897.

[59] Renrui Zhang et al. *Point-M2AE: Multi-scale Masked Autoencoders for Hierarchical Point Cloud Pre-training*. 2022. arXiv: 2205.14401 [cs.CV]. URL: https://arxiv.org/abs/2205.14401.

[60] Xiangdong Zhang, Shaofeng Zhang, and Junchi Yan. *PCP-MAE: Learning to Predict Centers for Point Masked Autoencoders*. 2024. arXiv: 2408.08753 [cs.CV]. URL: https://arxiv.org/abs/2408.08753.

[61] Junsheng Zhou et al. *Uni3D: Exploring Unified 3D Representation at Scale*. 2023. arXiv: 2310.06773 [cs.CV]. URL: https://arxiv.org/abs/2310.06773.

[62] Qingnan Zhou and Alec Jacobson. *Thingi10K: A Dataset of 10,000 3D-Printing Models*. 2016. arXiv: 1605.04797 [cs.GR]. URL: https://arxiv.org/abs/1605.04797.