GS-JEPA: SELF-SUPERVISED APPROACH TO LEARNING GAUSSIAN SPLAT REPRESENTATIONS USING JOINT EMBEDDING PREDICTIVE ARCHITECTURE

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce a self-supervised representation learning framework for 3D objects represented as Gaussian Splats based on Joint Embedding Predictive Architectures (JEPA). GS-JEPA represents individual Gaussian primitives as nodes in a local graph and learns their representations through latent prediction, explicitly modeling geometric and appearance-aware relationships between primitives. To isolate the effect of relational modeling, we additionally consider a point-based formulation that treats Gaussian primitives as an unordered set and serves as a baseline. Both formulations learn representations directly from Gaussian splat objects without relying on reconstruction objectives. Across classification and segmentation benchmarks, predictive learning outperforms the reconstruction-based Gaussian Masked Autoencoder. Notably, despite being pretrained exclusively on synthetic objects, the proposed JEPA-based approaches generalize effectively to real reconstructed Gaussian splat data. On MACGS classification, the point-based and graph-based approaches achieve 95.76% and 94.60% accuracy, respectively. On ShapeNetPart segmentation, the graph-based model achieves an mIoU of 85.15%, demonstrating the potential benefits of explicit relational modeling. These results establish JEPA-based predictive learning as an effective approach for transferable Gaussian splat representations and highlight relational modeling as a promising direction for 3D Gaussian representation learning.

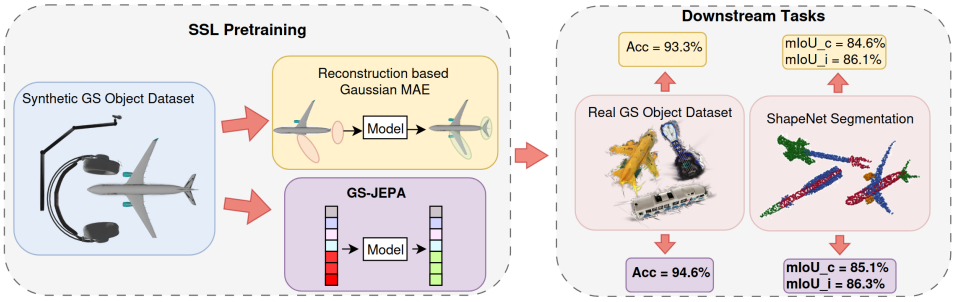

Figure 1: **GS-JEPA**: We present GS-JEPA, a pretraining framework that outperforms the Gaussian Masked Autoencoder (Gaussian-MAE) on downstream tasks such as object classification and segmentation. Pretraining on a synthetic objects dataset followed by linear evaluation on real-world Gaussian splat objects demonstrates that GS-JEPA achieves significantly better performance than Gaussian-MAE. Code is available on GitHub.

1 INTRODUCTION

3D Gaussian Splatting (3DGS) Kerbl et al. (2023) represents scenes as collections of anisotropic Gaussian primitives encoding both geometry and appearance. Compared with conventional point clouds Zhang et al. (2023), Gaussian splats offer a richer representation space that is increasingly

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

relevant for downstream 3D understanding Chen & Wang (2026). Learning effective 3D representations remains challenging due to limited semantic annotations, motivating self-supervised learning (SSL) from unlabeled data Ericsson et al. (2022). Existing 3D SSL methods have largely adopted reconstruction-based objectives, particularly masked autoencoding for point clouds Pang et al. (2022); Yu et al. (2022); Bachmann et al. (2021), with recent extensions to Gaussian splats Ma et al. (2025); Li et al. (2025). However, these methods largely inherit assumptions from point cloud learning and do not explicitly exploit the structural properties of Gaussian primitives. This limitation is particularly relevant because Gaussian representations can contain noisy primitives, irregular local structures, and reconstruction artifacts Yu et al. (2023). Reconstruction objectives may therefore encourage models to encode representation-specific low-level details rather than transferable object-level structure Wang et al. (2022). This motivates an alternative objective that avoids reconstructing the input and instead encourages prediction of meaningful latent representations. Joint Embedding Predictive Architectures (JEPA) Piccoli et al. (2025) offer a self-supervised paradigm that predicts representations of unobserved input regions directly in latent space. By avoiding explicit reconstruction, JEPA-style objectives can emphasize structural dependencies while reducing sensitivity to low-level details. Recent work has extended this paradigm to structured 3D data, including point clouds Saito et al. (2025), motivating its application to Gaussian splat representations.

In this work, we introduce GS-JEPA, a predictive self-supervised framework for learning transferable representations from Gaussian splats. We first establish a point-based formulation as a baseline for evaluating how JEPA-based predictive learning transfers to the Gaussian domain, treating Gaussian primitives as an unordered point set with additional Gaussian attributes. Building on this foundation, GS-JEPA represents individual Gaussian primitives as nodes in a local graph and explicitly models their geometric and appearance-aware relationships during predictive learning. This formulation preserves primitive-level structure that is aggregated in the point-based baseline, enabling GS-JEPA to incorporate relational information directly into the learned representations.

We evaluate GS-JEPA under distribution shifts between pretraining and downstream data, comparing predictive learning with reconstruction-based Gaussian pretraining (Figure 1). Experiments on downstream object understanding tasks, including classification and part segmentation, provide an evaluation benchmark for representation transfer across these distribution shifts.

Our contributions are:

- We introduce GS-JEPA, a predictive self-supervised framework for learning transferable representations directly from 3D Gaussian splat primitives.
- We study two formulations for implementing the JEPA paradigm on Gaussian splats: Point-GS-JEPA formulation that treats Gaussian primitives as an unordered point set with additional attributes, and the novel Graph-GS-JEPA formulation that preserves individual primitives and explicitly models their local geometric and appearance-aware relationships.
- We establish an evaluation benchmark for predictive Gaussian representation learning under distribution shifts between synthetic pretraining and downstream data, comparing against reconstruction-based pretraining across object classification and part segmentation tasks.

2 Related Work

Self-supervised learning for 3D representations. Self-supervised learning (SSL) has become a central paradigm for learning transferable 3D representations without semantic annotations. Early approaches primarily focused on point clouds and employed masked modeling and reconstruction objectives. Point-BERT Yu et al. (2022), Point-MAE Pang et al. (2022), and Point-M2AE Zhang et al. (2022) demonstrated that recovering masked geometric observations can produce representations that transfer effectively to downstream tasks such as object classification and segmentation. More recently, SSL has been extended to Gaussian-based 3D representations, exploiting Gaussian primitives as richer representations of geometry and appearance. GS-PT Liu et al. (2024) and GaussianPretrain Xu et al. (2024), for example, incorporate Gaussian representations into self-supervised pretraining through geometry- and appearance-aware objectives.

Masked modeling of Gaussian representations. Recent work has further established masked autoencoding as a promising paradigm for self-supervised learning directly in Gaussian space. ShapeSplat Ma et al. (2025) introduced a large-scale dataset of 3D Gaussian splats together with Gaussian-MAE, which masks Gaussian primitives and learns to reconstruct their missing geometric and appearance information. This demonstrates that Gaussian-aware masked modeling can produce transferable representations for downstream 3D understanding. Concurrently, SceneSplat Li et al. (2025) extended Gaussian pretraining to scene-level representations by combining masked Gaussian modeling with self-distillation and vision-language alignment.

Predictive representation learning and JEPA. Joint Embedding Predictive Architectures (JEPA) formulate self-supervised learning as prediction in latent space rather than explicit reconstruction LeCun & Courant (2022). I-JEPA Assran et al. (2023) demonstrated the effectiveness of this paradigm for image representation learning, while subsequent extensions to video and multimodal learning, including V-JEPA Assran et al. (2025) and VL-JEPA Chen et al. (2026), showed that latent prediction can be applied across different modalities and conditioning schemes. In 3D, JEPA-based representation learning remains comparatively underexplored, with recent work applying predictive learning to point clouds Saito et al. (2025).

Our work extends predictive self-supervised learning to 3D Gaussian splat representations. While point clouds primarily represent objects as sets of spatial points, Gaussian splats associate each primitive with additional geometric and appearance attributes. Treating Gaussian splats only as point sets therefore does not fully exploit the structure encoded by the representation. We investigate whether latent predictive learning can leverage these richer Gaussian representations to learn transferable features while avoiding explicit reconstruction of potentially noisy low-level Gaussian attributes.

3 GS-JEPA

We introduce GS-JEPA, a predictive self-supervised framework for learning representations directly from 3D Gaussian splats. We first establish **Point-GS-JEPA** as a baseline, treating Gaussian primitives as an unordered set with their associated attributes. Building on this formulation, **Graph-GS-JEPA** represents individual Gaussian primitives as nodes in a local graph and explicitly models their geometric and appearance-aware relationships during predictive learning. Both formulations learn by predicting latent representations of masked regions from their visible context rather than reconstructing Gaussian attributes.

JEPA Formulation JEPA learns representations by predicting the latent representation of a target region from a partially observed context. Given a context region c and a target region t, the context and target encoders produce

$$H_{ctx} = f_\theta(c), \qquad H_{tgt} = f_\xi(t), \tag{1}$$

where f_θ and f_ξ denote the context and target encoders, respectively. The target encoder parameters ξ are updated as an exponential moving average (EMA) of θ, following student-teacher self-supervised learning frameworks Tarvainen & Valpola (2018); Grill et al. (2020); He et al. (2020). The predictor estimates the target representation from the context representation and target positional information p_{tgt} as $\tilde{H}_{tgt} = p_\phi(H_{ctx}, p_{tgt})$, where p_ϕ denotes the predictor. Training minimizes

$$\mathcal{L} = D(\tilde{H}_{tgt}, H_{tgt}), \tag{2}$$

where D measures the distance between the predicted and target representations, typically cosine similarity Grill et al. (2020) or L2 distance Chen et al. (2026).

Point-based baseline We instantiate the JEPA formulation for Gaussian splats by treating the primitives as an unordered point set augmented with Gaussian attributes. Following the point-based predictive learning formulation of Point-JEPA Saito et al. (2025), Gaussian primitives are organized into local regions and converted into tokens, which are then processed by Transformer-based context and target encoders f_θ and f_ξ, respectively, together with a Transformer predictor p_ϕ. This provides a direct point-based baseline and establishes the token-level formulation used as the basis for the Graph-GS-JEPA.

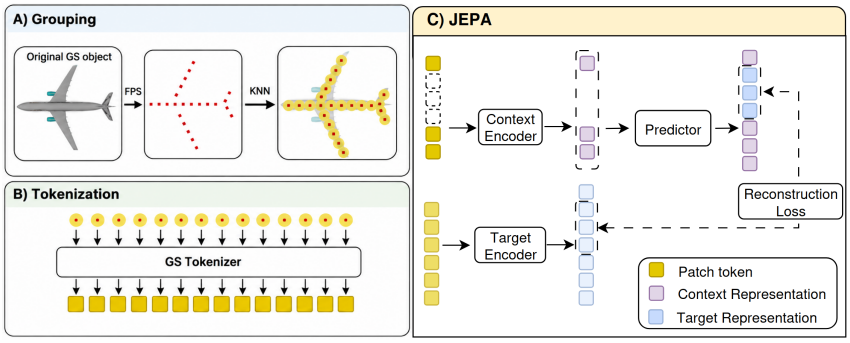

Figure 2: Overview of the proposed Point-GS-JEPA framework. The pipeline consists of three main stages: (A) *Grouping*, where Gaussian primitives are divided into local regions using FPS and KNN grouping; (B) *Tokenization*, where each local Gaussian group is independently encoded into a latent token representation using a modified PointNet architecture adapted for Gaussian attributes; and (C) *JEPA pretraining*, where transformer-based context and target encoders together with the predictor module learn latent representations through predictive learning.

As illustrated in Figure 2, Gaussian primitives are first partitioned into local regions using Farthest Point Sampling (FPS) followed by K-Nearest Neighbors (KNN) grouping Qi et al. (2017b). Let a Gaussian object be $\mathcal{G} = \{g_i\}_{i=1}^{M}$, where each primitive g_i is described by its Gaussian attributes. FPS selects N representative primitives as group centroids, and KNN identifies k neighboring primitives around each centroid, producing local groups $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$, where $P_j \in \mathbb{R}^{k \times d}$ contains the primitives assigned to the j-th region. Different grouping configurations and their resulting local regions are illustrated in Appendix E.

Each local Gaussian region is independently transformed into a fixed-dimensional latent token using a PointNet-based tokenizer Qi et al. (2017a); Saito et al. (2025). The tokenizer uses the Gaussian descriptor, consisting of position $\mathbf{x} \in \mathbb{R}^3$, opacity $o \in \mathbb{R}$, scale $\mathbf{s} \in \mathbb{R}^3$, rotation $\mathbf{r} \in \mathbb{R}^4$, and spherical harmonic coefficients $\mathbf{h} \in \mathbb{R}^3$. The resulting local embeddings form a token sequence for Transformer-based predictive learning. Details about mathematical formulation are provided in Appendix C.

Since Gaussian splats have no canonical ordering, the local regions are sequenced using the greedy traversal strategy introduced by Point-JEPA Saito et al. (2025), which selects the next region according to the proximity of its centroid to that of the previously selected region. The resulting sequence is partitioned into context and target groups for predictive learning, with target groups corresponding to masked regions and the remaining groups forming the visible context.

The context and target tokens are processed by the Transformer-based context and target encoders, while the predictor Transformer estimates the latent target representations from the visible context. Positional information for target groups is obtained by projecting their centroid positions $\mathbf{x}$ into the positional embedding space using a dedicated MLP Ramasinghe & Lucey (2021). The model is optimized using a Smooth L1 loss between the predicted and target embeddings Girshick (2015). Additional architectural details are provided in Appendix F.

Graph-GS-JEPA Building on the point-based baseline, we formulate GS-JEPA as a graph over individual Gaussian primitives, preserving primitive-level structure while explicitly modeling local geometric and appearance-aware relationships. Gaussian splats exhibit local dependencies between neighboring primitives through their spatial arrangement, anisotropic support, and appearance attributes Kerbl et al. (2023), which can be represented through localized graph message passing Kipf & Welling (2017). As shown in Figure 3, Gaussian primitives form graph nodes and local neighboring primitives are connected by edges, providing the structural basis for context encoding and target selection. Unlike the point-based formulation, this representation allows a graph neural network to operate directly on individual Gaussian primitives while incorporating their local relationships.

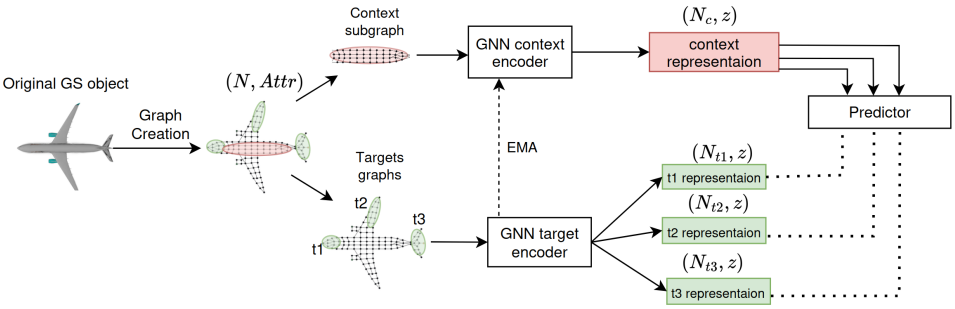

Figure 3: Overview of the proposed Graph-GS-JEPA framework. The pipeline consists of four main stages: (1) construction of a graph representation of a Gaussian object, (2) sampling of target and context subgraphs, (3) encoding of the visible context graph using a GNN backbone and (4) predictive estimation of target node representations from context embeddings.

Graph Construction from Gaussian Object Given a Gaussian object $\mathcal{G} = \{g_i\}_{i=1}^{M}$, we construct a KNN graph Wang et al. (2019), where the neighborhood of node i is defined as

$$\mathcal{N}_k(i) = \mathrm{KNN}(g_i, k), \tag{3}$$

using a selected subset of Gaussian attributes to compute pairwise distances and determine graph connectivity. Different attribute configurations can therefore define different local neighborhoods. The resulting topology determines the local receptive field over which information is propagated during graph encoding.

Each Gaussian node is independently projected into a 384-dimensional latent representation using a lightweight MLP Han et al. (2023). The attributes used for node encoding can be selected independently from those used to construct the graph, allowing the topology and initial node representation to capture complementary information. The resulting embeddings remain associated with individual Gaussian primitives, enabling subsequent graph layers to explicitly propagate information between neighboring nodes.

In addition to graph connectivity, we encode pairwise relationships between neighboring Gaussian primitives. For an edge (i, j) connecting Gaussians g_i and g_j, we define the edge attributes as

$$\mathbf{e}_{ij} = \mathrm{concat}\left(\mathbf{e}_{ij}^{\mathrm{dist}}, \mathbf{e}_{ij}^{\mathrm{dir}}, \mathbf{e}_{ij}^{\mathrm{opa}}, \mathbf{e}_{ij}^{\mathrm{scale}}, \mathbf{e}_{ij}^{\mathrm{rot}}, \mathbf{e}_{ij}^{\mathrm{app}}\right).$$

The individual components capture complementary pairwise relationships between the two Gaussians: $\mathbf{e}_{ij}^{\mathrm{dist}}$ encodes their centroid distance, $\mathbf{e}_{ij}^{\mathrm{dir}}$ their relative direction, $\mathbf{e}_{ij}^{\mathrm{opa}}$ their opacity relationship, $\mathbf{e}_{ij}^{\mathrm{scale}}$ their relative scale, $\mathbf{e}_{ij}^{\mathrm{rot}}$ their relative orientation, and $\mathbf{e}_{ij}^{\mathrm{app}}$ their appearance similarity based on spherical-harmonic features. Continuous distance and angular quantities are represented using Gaussian radial basis function (RBF) embeddings. Depending on the experimental configuration, individual attribute groups can be enabled or omitted, with the selected components concatenated to form the final edge feature. The precise definitions, normalization and encoding of each attribute are provided in Appendix D.

Graph-aware context and target sampling. We sample target regions as connected subgraphs over the KNN graph, combining k-hop neighborhood Skenderi et al. (2025) expansion with random selection of nodes from the final hop to match the desired target size. Multiple non-overlapping target subgraphs are sampled per object, while the remaining visible nodes form the context graph. This preserves local graph structure within the target regions while providing a consistent number of nodes for context-target prediction.

Crucially, the target nodes and all incident context-target edges are removed before context encoding. This pruning prevents information leakage through message passing and ensures that the context encoder has access only to visible Gaussian primitives Zhu et al. (2024). The resulting context graph is dynamically remapped to retain only valid context-to-context connections.

This formulation combines learned feature similarity with the explicit geometric and appearance-aware relationships encoded by the graph. Stacking multiple graph-attention layers progressively expands the receptive field while preserving the locality defined by the graph.

Graph encoder Since GS-JEPA represents a Gaussian object as a graph, the context graph is encoded using a Graph Neural Network (GNN)-based model. We adopt a Graphormer-style backbone Ying et al. (2021), in which unlike global self-attention, attention is restricted to KNN neighbors, while edge features provide explicit relational biases. Given node representations $\mathbf{h}_i$ and $\mathbf{h}_j$, learned linear projections produce the query $\mathbf{q}_i$, key $\mathbf{k}_j$, and value $\mathbf{v}_j$. The attention weights for node i are then computed over its neighbors $\mathcal{N}(i)$ as

$$\alpha_{ij} = \mathrm{softmax}_{j\in\mathcal{N}(i)}\left(\frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d}} + b_{ij}\right), \qquad b_{ij} = \mathrm{MLP}(\mathbf{e}_{ij}), \tag{4}$$

where $\mathbf{e}_{ij}$ denotes the edge representation between nodes i and j, and b_{ij} provides a learnable edge-dependent attention bias. The updated node representation aggregates the value features of its neighboring nodes as $\mathbf{h}_i' = \sum_{j\in\mathcal{N}(i)} \alpha_{ij}\mathbf{v}_j$. Stacking graph-attention layers progressively expands the receptive field.

Our main model uses the Graphormer-style backbone, while the same graph representation can also be paired with more lightweight graph backbones that directly incorporate edge attributes, such as GINE (Appendix J).

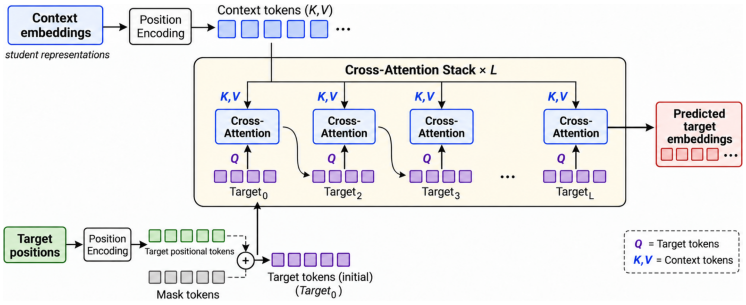

Figure 4: Proposed cross-attention predictor architecture. Target tokens, initialized from mask tokens and target positional encodings, are iteratively refined through stacked cross-attention blocks. At each layer, target tokens act as queries while context tokens derived from the context embeddings serve as fixed keys and values. The final target representations are used to predict the target embeddings

Predictor The predictor p_ϕ operates on the encoded context nodes and estimates representations for the removed target nodes. Since the graph formulation operates directly on individual Gaussian primitives and therefore produces substantially more tokens than the point-based formulation, we use six cross-attention blocks rather than a standard Transformer predictor. This allows target tokens to selectively aggregate information from the visible context while reducing the computational complexity of target prediction, as demonstrated in Appendix B. Figure 4 illustrates the predictor p_ϕ architecture.

Let $\mathbf{H}_{\mathrm{ctx}} \in \mathbb{R}^{N_c\times D}$ denote the encoded context representations, which are projected to the predictor dimension as

$$\mathbf{C} = W_{\mathrm{pred}}\mathbf{H}_{\mathrm{ctx}} \in \mathbb{R}^{N_c\times D_p}. \tag{5}$$

The N_t target tokens $\mathbf{T}^{(0)} \in \mathbb{R}^{N_t\times D_p}$ are initialized from a shared learnable mask token and augmented with positional embeddings. Given the Gaussian positions $\mathbf{x}_{\mathrm{ctx}}$ and $\mathbf{x}_{\mathrm{tgt}}$, a shared positional encoding function f_{pos} maps each position to the predictor dimension:

$$\mathbf{C} \leftarrow \mathbf{C} + f_{\mathrm{pos}}(\mathbf{x}_{\mathrm{ctx}}), \qquad \mathbf{T}^{(0)} \leftarrow \mathbf{T}^{(0)} + f_{\mathrm{pos}}(\mathbf{x}_{\mathrm{tgt}}), \tag{6}$$

where f_{pos} is a lightweight MLP that produces the positional embeddings.

Each cross-attention block updates only the target tokens by attending to the fixed context representation. The core operation is

$$\mathrm{Attn}(\mathbf{T},\mathbf{C}) = \mathrm{softmax}\left(\frac{Q(\mathbf{T})K(\mathbf{C})^\top}{\sqrt{d_h}}\right)V(\mathbf{C}), \tag{7}$$

Table 1: Number of trainable parameters for baseline and our proposed models

	Gaussian-MAE	Point-GS-JEPA	Graph-GS-JEPA (Graphormer)
Pretraining	28.79M	24.68M	13.545M
Full finetuning	21.78	21.78	10.80M

where Q, K, and V are learned linear projections producing queries from target tokens and keys and values from context tokens, and d_h is the per-head dimension. Thus, the attention weights have size $N_t \times N_c$, allowing each target token to aggregate information from all visible context nodes. Each block additionally applies layer normalization, residual connections, and an MLP. After six blocks, the target representations are projected back to the encoder dimension:

$$\hat{\mathbf{H}}_{\text{tgt}} = W_{\text{out}}\mathbf{T}^{(6)} \in \mathbb{R}^{N_t \times D}, \tag{8}$$

and compared with the corresponding target encoder representations using the JEPA objective.

4 Experiments and Results

We evaluate GS-JEPA primarily by measuring the transferability of pretrained representations from synthetic Gaussian data to downstream 3D object understanding tasks, including classification and part segmentation. Following the evaluation protocol introduced for Gaussian representation learning by Gaussian-MAE Ma et al. (2025), we compare the proposed predictive formulations with reconstruction-based pretraining under matched downstream evaluation settings.

Pretraining We pretrain both point-based baseline and graph-based GS-JEPA on the synthetic training split of the ShapeSplat Chang et al. (2015); Ma et al. (2025) dataset, a large-scale synthetic dataset containing objects from 55 classes. All pretraining runs use the same training split to ensure consistent comparisons across methods. During pretraining, the context encoder is optimized using a Smooth L1 loss between the predicted and target latent representations. After pretraining, the context encoder is retained and used to initialize the downstream fine-tuning and evaluation tasks. Detailed pretraining configurations are provided in Appendix G.

Table 1 reports the number of trainable parameters used during pretraining and full fine-tuning. The difference between the two settings for the JEPA-based methods arises from the predictor, which is optimized during pretraining but discarded for downstream fine-tuning; the target encoder is updated via EMA and therefore is not counted as trainable. Gaussian-MAE and Point-GS-JEPA use the same tokenizer and Transformer encoder architecture, providing a controlled comparison of the pretraining objectives without introducing differences in tokenization or encoder architecture. In contrast, Graph-GS-JEPA operates on an explicit graph representation and therefore employs a graph-based encoder (Graphormer), resulting in a distinct architecture and parameter count. Accordingly, we use Point-GS-JEPA as the controlled baseline for evaluating the effect of replacing reconstruction-based pretraining with latent predictive learning, while Graph-GS-JEPA is subsequently introduced to study the additional effect of explicit relational modeling.

4.1 Gaussian object classification.

Setup Classification is the primary evaluation task and is performed on ModelSplat40 and MACGS Zhang et al. (2025). ModelSplat40 provides an object-level benchmark with a distribution closely related to the ShapeSplat pretraining data due to their shared synthetic generation process. In contrast, MACGS contains Gaussian objects reconstructed using a standard 3D Gaussian Splatting reconstruction process, resulting in greater reconstruction variability and noise. Further details on the differences between the synthetic and reconstructed datasets are provided in Appendix A. Together, these datasets allow us to evaluate representation transfer under both relatively similar data distributions and a more challenging distribution shift.

For each pretrained backbone, local representations are aggregated into a global object descriptor, following the representation aggregation strategy of Point2Vec Bachmann et al. (2021). The same aggregation procedure is applied to the point-based baseline and GS-JEPA to provide a consistent downstream evaluation interface.

Table 2: Classification accuracy (%) on ModelSplat40 and MACGS. Our methods are marked with †. The Graph-GS-JEPA variants use either a Graphormer or GINE backbone. Models marked with ⋆ use the corresponding downstream encoder architectures of the self-supervised models, but are trained from random initialization without self-supervised pretraining.

Dataset	Supervised Baselines				Self-Supervised Learning				
	PointNet Qi et al. (2017a)	PointNet++ Qi et al. (2017b)	Point Transformer⋆ Zhao et al. (2021)	Graphormer⋆	Finetuning	Gaussian-MAE Ma et al. (2025)	Point-GS-JEPA†	Graph-GS-JEPA Graphormer †	Graph-GS-JEPA GINE †
ModelSplat40	89.91 ± 0.12	**91.93 ± 0.10**	90.64 ± 0.31	91.36 ± 0.45	**FULL**	92.91 ± 0.31	93.14 ± 0.16	**93.42 ± 0.06**	91.98 ± 0.11
					LINEAR	90.38 ± 0.37	91.57 ± 0.09	**91.77 ± 0.06**	89.10 ± 0.12
					MLP-3	91.57 ± 0.14	92.20 ± 0.05	**92.34 ± 0.14**	91.14 ± 0.08
MACGS	81.52 ± 0.60	86.18 ± 0.07	**89.65 ± 0.31**	87.23 ± 0.55	**FULL**	93.31 ± 0.11	**95.76 ± 0.06**	94.60 ± 0.14	89.24 ± 0.26
					LINEAR	81.55 ± 0.47	**90.52 ± 0.12**	86.67 ± 0.14	79.98 ± 0.18
					MLP-3	88.08 ± 0.09	**93.43 ± 0.07**	92.07 ± 0.11	91.27 ± 0.15

We consider three evaluation protocols: *linear evaluation*, in which the pretrained backbone is frozen and only a linear classifier is trained; *MLP evaluation*, in which the backbone remains frozen and a three-layer MLP classifier is optimized; and *full fine-tuning*, where both the pretrained backbone and classifier are updated. These settings measure, respectively, the quality of the fixed representation, its separability under a nonlinear classifier, and the benefit of pretrained initialization when adaptation to the target task is allowed.

Results Table 2 reports classification accuracy on ModelSplat40 and MACGS. For all self-supervised approaches, we report the mean accuracy and standard deviation over three independent fine-tuning runs. We additionally include architecture-matched supervised baselines, which use the same encoders as the corresponding self-supervised models but are trained from random initialization without self-supervised pretraining. Specifically, Point Transformer has the same architecture as encoders used in Gaussian-MAE and Point-GS-JEPA, while the Graphormer⋆ baseline uses the same Graphormer architecture as Graph-GS-JEPA. These comparisons therefore isolate the contribution of self-supervised pretraining from the downstream encoder architecture.

Overall, both GS-JEPA formulations outperform Gaussian-MAE across the reported evaluation settings. On ModelSplat40, Graph-GS-JEPA with the Graphormer backbone achieves the best self-supervised results, reaching 93.42% under full fine-tuning, 91.77% with linear evaluation, and 92.34% with the MLP-3 protocol. Notably, its full fine-tuning performance improves over the architecture-matched Graphormer⋆ baseline (91.36%) by 2.06 percentage points, directly demonstrating the benefit of self-supervised pretraining. On MACGS, Point-GS-JEPA performs best, achieving 95.76%, 90.52%, and 93.43% under full fine-tuning, linear evaluation, and MLP-3, respectively. Under full fine-tuning, this corresponds to a 2.45 percentage-point improvement over Gaussian-MAE and a 6.11 percentage-point improvement over the architecture-matched Transformer⋆ baseline.

The two GS-JEPA formulations exhibit complementary behavior across the datasets. Graph-GS-JEPA consistently outperforms Point-GS-JEPA on ModelSplat40 across all three evaluation protocols, whereas Point-GS-JEPA performs better on MACGS. This suggests that the benefit of explicit graph-based modeling depends on the downstream data distribution: the graph formulation provides stronger representations on the synthetic ModelSplat40 benchmark, while the point-based formulation transfers more effectively to the reconstructed MACGS data.

Figure 5 evaluates the effect of incorporating different edge attributes into Graph-GS-JEPA. Adding relational information based on distance, direction, and opacity similarity generally improves classification accuracy, whereas incorporating additional edge attributes provides smaller gains or can lead to a decrease in performance. These results indicate that geometric and selected appearance relationships are particularly useful for constructing the graph representation.

4.2 Part segmentation.

Setup We further evaluate fine-grained representation quality on ShapeNetPart using the corresponding Gaussian representations provided by ShapeSplat. Following established 3D self-supervised learning protocols Yu et al. (2022); Pang et al. (2022); Bachmann et al. (2021), pretrained representations are used to predict per-point part labels. The Gaussian splats and their corresponding point clouds share the same spatial coordinate system and thus occupy the same geometric space. This correspondence enables Gaussian-derived features to be propagated to the original point locations, where they can be evaluated against the corresponding ShapeNetPart annotations. This

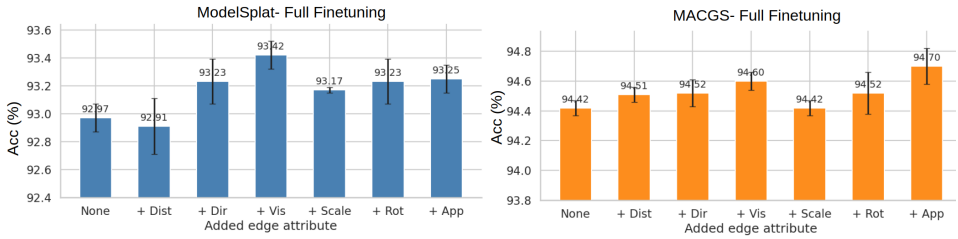

Figure 5: Effect of edge attributes on the graph-based GS-JEPA performance for ModelSplat and MACGS. Each category represents the addition of a specific edge attribute to the graph, with "None" denoting the baseline without additional edge attributes.

protocol follows the Gaussian-domain extension introduced by Gaussian-MAE Ma et al. (2025). Segmentation models are evaluated under full fine-tuning.

Table 3: Part segmentation results on ShapeNetPart. Category mIoU (mIoU_C) measures the average segmentation performance across object categories, while instance mIoU (mIoU_I) averages performance across individual object instances. Our methods are marked with †.

Method	**mIoU$_C$**	**mIoU$_I$**
Point Cloud only		
Transformer-OcCo Yu et al. (2022)	83.4	85.1
Point-MAE Pang et al. (2022)	84.1	86.1
Point-M2AE Zhang et al. (2022)	**84.9**	**86.5**
Point2Vec Knaebel et al. (2023)	84.6	86.3
PointGPT-S Chen et al. (2023)	84.1	86.2
Point-JEPA Saito et al. (2025)	83.9 ± 0.1	85.8 ± 0.1
Gaussian Splat representation		
Gaussian-MAE	84.23 ± 0.20	85.88 ± 0.11
Point-GS-JEPA†	83.99 ± 0.13	85.74 ± 0.22
Graph-GS-JEPA (Graphormer)†	$\mathbf{85.15 \pm 0.14}$	$\mathbf{86.27 \pm 0.07}$
Graph-GS-JEPA (GINE)†	$\mathbf{84.71 \pm 0.09}$	$\mathbf{86.11 \pm 0.05}$

Results Table 3 reports the part segmentation results on ShapeNetPart. The graph-based formulation achieves the strongest performance among the evaluated Gaussian-based methods on both category and instance mIoU, reaching 85.15% and 86.27%, respectively. Its category mIoU also surpasses all compared point-cloud self-supervised methods, while its instance mIoU is competitive with the strongest point-cloud baselines. A possible explanation for this behavior is the primitive-level nature of the graph representation. During pretraining, each Gaussian primitive is represented by an individual node, allowing fine-grained local information to be preserved throughout the encoding process. In contrast, Gaussian-MAE and Point-GS-JEPA aggregate multiple Gaussian primitives into shared patch representations, which may discard local structural details relevant to dense prediction. Since part segmentation requires accurate localization of semantic boundaries, preserving primitive-level information may provide an advantage for this task. Interestingly, point-based baseline achieves results very similar to those reported for Point-JEPA Saito et al. (2025), with nearly identical category and instance mIoU. This observation suggests that the behavior of the original Point-JEPA formulation largely transfers to the Gaussian domain. The results indicate that patch-based predictive representations do not provide the same advantage for dense segmentation as the proposed graph-based formulation.

Additional experimental results are provided in Appendices H-J.

5 Conclusions and Limitations

The results demonstrate that predictive pretraining provides strong transferable representations for Gaussian-based object classification and part segmentation, with the graph-based formulation showing particular benefits for fine-grained prediction. The current evaluation focuses on object-level Gaussian representations, as large-scale datasets of reconstructed Gaussian scenes with corresponding annotations remain limited. Extending GS-JEPA to large-scale scene representations and scene-level tasks is therefore a natural direction for future work.

REFERENCES

Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture, 2023. URL https://arxiv.org/abs/2301.08243.

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba, Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, Francois Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-jepa 2: Self-supervised video models enable understanding, prediction and planning, 2025. URL https://arxiv.org/abs/2506.09985.

Joël Bachmann, Kenneth Blomqvist, Julian Förster, and Roland Siegwart. Points2vec: Unsupervised object-level feature learning from point clouds, 2021. URL https://arxiv.org/abs/2102.04136.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository, 2015. URL https://arxiv.org/abs/1512.03012.

Delong Chen, Mustafa Shukor, Theo Moutakanni, Willy Chung, Jade Yu, Tejaswi Kasarla, Yejin Bang, Allen Bolourchi, Yann LeCun, and Pascale Fung. Vl-jepa: Joint embedding predictive architecture for vision-language, 2026. URL https://arxiv.org/abs/2512.10942.

Guangyan Chen, Meiling Wang, Yi Yang, Kai Yu, Li Yuan, and Yufeng Yue. Pointgpt: Auto-regressively generative pre-training from point clouds, 2023. URL https://arxiv.org/abs/2305.11487.

Guikun Chen and Wenguan Wang. A survey on 3d gaussian splatting, 2026. URL https://arxiv.org/abs/2401.03890.

Linus Ericsson, Henry Gouk, Chen Change Loy, and Timothy M. Hospedales. Self-supervised representation learning: Introduction, advances, and challenges. *IEEE Signal Processing Magazine*, 39(3):42–62, May 2022. ISSN 1558-0792. doi: 10.1109/msp.2021.3134634. URL http://dx.doi.org/10.1109/MSP.2021.3134634.

Ross Girshick. Fast r-cnn, 2015. URL https://arxiv.org/abs/1504.08083.

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

Xiaotian Han, Tong Zhao, Yozen Liu, Xia Hu, and Neil Shah. Mlpinit: Embarrassingly simple gnn training acceleration with mlp initialization, 2023. URL https://arxiv.org/abs/2210.00102.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning, 2020. URL https://arxiv.org/abs/1911.05722.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks, 2020. URL https://arxiv.org/abs/1905.12265.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering, 2023. URL https://arxiv.org/abs/2308.04079.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. URL https://arxiv.org/abs/1609.02907.

Karim Knaebel, Jonas Schult, Alexander Hermans, and Bastian Leibe. Point2vec for self-supervised representation learning on point clouds, 2023. URL https://arxiv.org/abs/2303.16570.

Yann LeCun and Courant. A path towards autonomous machine intelligence version 0.9.2, 2022-06-27. 2022. URL https://api.semanticscholar.org/CorpusID:251881108.

Yue Li, Qi Ma, Runyi Yang, Huapeng Li, Mengjiao Ma, Bin Ren, Nikola Popovic, Nicu Sebe, Ender Konukoglu, Theo Gevers, Luc Van Gool, Martin R. Oswald, and Danda Pani Paudel. Scenesplat: Gaussian splatting-based scene understanding with vision-language pretraining, 2025. URL https://arxiv.org/abs/2503.18052.

Keyi Liu, Yeqi Luo, Weidong Yang, Jingyi Xu, Zhijun Li, Wen-Ming Chen, and Ben Fei. Gs-pt: Exploiting 3d gaussian splatting for comprehensive point cloud understanding via self-supervised learning, 2024. URL https://arxiv.org/abs/2409.04963.

Qi Ma, Yue Li, Bin Ren, Nicu Sebe, Ender Konukoglu, Theo Gevers, Luc Van Gool, and Danda Pani Paudel. Shapesplat: A large-scale dataset of gaussian splats and their self-supervised pretraining, 2025. URL https://arxiv.org/abs/2408.10906.

Yatian Pang, Wenxiao Wang, Francis E. H. Tay, Wei Liu, Yonghong Tian, and Li Yuan. Masked autoencoders for point cloud self-supervised learning, 2022. URL https://arxiv.org/abs/2203.06604.

Francesco Piccoli, Gabriel Vogel, and Jana M. Weber. Joint embedding predictive architecture for self-supervised pretraining on polymer molecular graphs, 2025. URL https://arxiv.org/abs/2506.18194.

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017a. URL https://arxiv.org/abs/1612.00593.

Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017b. URL https://arxiv.org/abs/1706.02413.

Sameera Ramasinghe and Simon Lucey. Learning positional embeddings for coordinate-mlps. Dec 2021. URL http://arxiv.org/abs/2112.11577v2.

Ayumu Saito, Prachi Kudeshia, and Jiju Poovvancheri. Point-jepa: A joint embedding predictive architecture for self-supervised learning on point cloud, 2025. URL https://arxiv.org/abs/2404.16432.

Geri Skenderi, Hang Li, Jiliang Tang, and Marco Cristani. Graph-level representation learning with joint-embedding predictive architectures, 2025. URL https://arxiv.org/abs/2309.16014.

Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results, 2018. URL https://arxiv.org/abs/1703.01780.

Guangrun Wang, Yansong Tang, Liang Lin, and Philip H.S. Torr. Semantic-aware auto-encoders for self-supervised representation learning. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9654–9665, 2022. doi: 10.1109/CVPR52688.2022.00944.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds, 2019. URL https://arxiv.org/abs/1801.07829.

Shaoqing Xu, Fang Li, Shengyin Jiang, Ziying Song, Li Liu, and Zhi xin Yang. Gaussianpretrain: A simple unified 3d gaussian representation for visual pre-training in autonomous driving, 2024. URL https://arxiv.org/abs/2411.12452.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation?, 2021. URL https://arxiv.org/abs/2106.05234.

Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. Point-bert: Pre-training 3d point cloud transformers with masked point modeling, 2022. URL https://arxiv.org/abs/2111.14819.

Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting, 2023. URL https://arxiv.org/abs/2311.16493.

Huang Zhang, Changshuo Wang, Shengwei Tian, Baoli Lu, Liping Zhang, Xin Ning, and Xiao Bai. Deep learning-based 3d point cloud classification: A systematic survey and outlook. *Displays*, 79:102456, 2023. ISSN 0141-9382. doi: 10.1016/j.displa.2023.102456. URL http://dx.doi.org/10.1016/j.displa.2023.102456.

Renrui Zhang, Ziyu Guo, Rongyao Fang, Bin Zhao, Dong Wang, Yu Qiao, Hongsheng Li, and Peng Gao. Point-m2ae: Multi-scale masked autoencoders for hierarchical point cloud pre-training, 2022. URL https://arxiv.org/abs/2205.14401.

Ruiqi Zhang, Hao Zhu, Jingyi Zhao, Qi Zhang, Xun Cao, and Zhan Ma. Mitigating ambiguities in 3d classification with gaussian splatting, 2025. URL https://arxiv.org/abs/2503.08352.

Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point transformer, 2021. URL https://arxiv.org/abs/2012.09164.

Jing Zhu, Yuhang Zhou, Vassilis N. Ioannidis, Shengyi Qian, Wei Ai, Xiang Song, and Danai Koutra. Pitfalls in link prediction with graph neural networks: Understanding the impact of target-link inclusion & better practices. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, WSDM '24, pp. 994–1002, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703713. doi: 10.1145/3616855.3635786. URL https://doi.org/10.1145/3616855.3635786.

Table 4: Summary statistics for all Gaussian splat datasets used in this work.

Dataset	# Gaussians per Object	# Objects	Train	Test/Val	# Categories
MACGS	112 732 ± 63 384	5 370	4 288	1 082	30
ShapeSplat	24 256 ± 9 471	52 456	41 940	10 516	55
ModelSplat40	22 456 ± 8 358	12 309	9 842	2 467	40

A Synthetic and Real Gaussian Splat Datasets

The two Gaussian datasets used for downstream classification, ModelSplat40 and MACGS, differ significantly in their construction methodology and structural characteristics. ModelSplat40 is substantially more similar to the ShapeSplat dataset used during pretraining, as both datasets were generated using the same synthetic reconstruction pipeline. Consequently, they exhibit comparable primitive distributions, object normalization procedures, and average numbers of Gaussian primitives per object, as shown in Table 4.

In contrast, MACGS contains substantially larger and more complex Gaussian representations, which is typical for reconstructions generated from real image observations. Visual inspection of the datasets, illustrated in Figure 6, reveals several important differences between MACGS and the synthetic datasets:

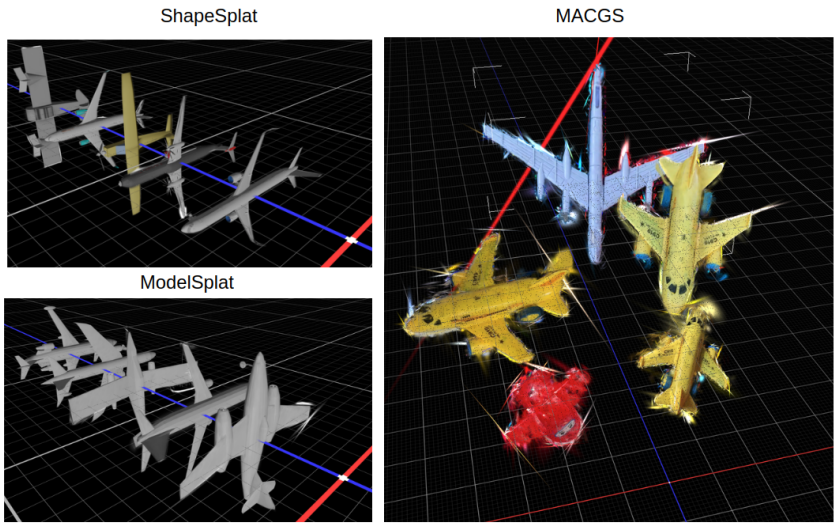

Figure 6: Visualization of airplane objects from the three Gaussian splat datasets used in this work. The left column presents synthetic objects from ShapeSplat (top) and ModelSplat40 (bottom), while the right column presents real reconstructed objects from MACGS. ShapeSplat and ModelSplat40 objects are originally centered and normalized within the coordinate system, whereas MACGS objects preserve their original scale and spatial positioning, illustrating the substantial structural differences between synthetic and real Gaussian splat reconstructions.

B Computational Complexity of Cross-Attention

The main computational difference between a standard Transformer predictor and the proposed cross-attention predictor lies in the size of the attention matrix. Let D_p denote the predictor embedding dimension. For a graph containing $N = 1024$ Gaussian nodes, a standard Transformer operating on all nodes forms the self-attention matrix

$$\mathbf{A}_{\text{self}} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_h}}\right) \in \mathbb{R}^{N\times N},$$

where $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{N \times d_h}$. Computing the attention scores $\mathbf{Q}\mathbf{K}^\top$ therefore has complexity

$$\mathcal{O}(N^2 D_p) = \mathcal{O}(1024^2 D_p),$$

where the factor D_p accounts for the D_p-dimensional token representations across the attention heads.

In the proposed predictor, the nodes are divided into N_c context and N_t target nodes, with $N_c + N_t = N$. The cross-attention operation is

$$\mathbf{A}_{\text{cross}} = \text{softmax}\left(\frac{\mathbf{Q}_{\text{tgt}}\mathbf{K}_{\text{ctx}}^\top}{\sqrt{d_h}}\right) \in \mathbb{R}^{N_t \times N_c},$$

where target tokens provide the queries and context tokens provide the keys and values. Its attention computation therefore has complexity

$$\mathcal{O}(N_t N_c D_p), \qquad N_t + N_c = 1024.$$

Thus, instead of computing interactions between every pair of Gaussian nodes, the predictor computes only target–context interactions, avoiding both context–context and target–target attention. For an equal split, $N_t = N_c = 512$, the cross-attention matrix contains

$$512 \times 512 = 262{,}144$$

entries, compared with

$$1024 \times 1024 = 1{,}048{,}576$$

entries for full self-attention, corresponding to a four-fold reduction in the attention-matrix size.

C FORMULATION OF POINT-GS-JEPA BASELINE

To make the connection to the generic JEPA formulation explicit, we first represent each Gaussian primitive by a finite-dimensional attribute vector. For the i-th Gaussian, we define

$$\mathbf{g}_i = [\mathbf{x}_i,\ o_i,\ \mathbf{s}_i,\ \mathbf{r}_i,\ \mathbf{h}_i] \in \mathbb{R}^{d_g}, \tag{9}$$

where $\mathbf{x}_i \in \mathbb{R}^3$ is the Gaussian position, $o_i \in \mathbb{R}$ is its opacity, $\mathbf{s}_i \in \mathbb{R}^3$ is its scale, $\mathbf{r}_i \in \mathbb{R}^4$ is its rotation quaternion, and $\mathbf{h}_i \in \mathbb{R}^3$ denotes the spherical-harmonic descriptor. Thus, for the attributes listed above,

$$d_g = 3 + 1 + 3 + 4 + 3 = 14, \tag{10}$$

and the complete Gaussian object can be written as

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_1^\top \\ \vdots \\ \mathbf{g}_M^\top \end{bmatrix} \in \mathbb{R}^{M \times d_g}. \tag{11}$$

After FPS and KNN grouping, the Gaussian primitives are partitioned into N local regions, each containing k primitives:

$$P_j = \{\mathbf{g}_{j,1}, \ldots, \mathbf{g}_{j,k}\} \in \mathbb{R}^{k \times d_g}, \qquad j = 1, \ldots, N. \tag{12}$$

The PointNet-based tokenizer τ_ψ maps each local region to a d-dimensional token,

$$\mathbf{z}_j = \tau_\psi(P_j) \in \mathbb{R}^d, \tag{13}$$

The N local tokens are then arranged according to the greedy spatial traversal, yielding

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1^\top \\ \vdots \\ \mathbf{z}_N^\top \end{bmatrix} \in \mathbb{R}^{N \times d}. \tag{14}$$

Let $\mathcal{C}$ and $\mathcal{T}$ denote the sets of context and target indices, respectively, with $\mathcal{C} \cap \mathcal{T} = \varnothing$ and $\mathcal{C} \cup \mathcal{T} = \{1, \ldots, N\}$. The corresponding token sequences are

$$\mathbf{Z}_c \in \mathbb{R}^{N_c \times d}, \qquad \mathbf{Z}_t \in \mathbb{R}^{N_t \times d}, \tag{15}$$

where $N_c = |\mathcal{C}|$ and $N_t = |\mathcal{T}|$.

The context and target Transformer encoders then map these token sequences into contextual latent spaces:

$$\mathbf{S}_c = F_\theta(\mathbf{Z}_c) \in \mathbb{R}^{N_c \times d_e}, \tag{16}$$

and

$$\mathbf{S}_t = F_\xi(\mathbf{Z}_t) \in \mathbb{R}^{N_t \times d_e}, \tag{17}$$

where d_e is the Transformer embedding dimension. Thus, the generic JEPA quantities S_c and S_t correspond here to sequences of contextualized local-region embeddings rather than individual Gaussian attributes.

For each target region $j \in \mathcal{T}$, its centroid position $\mathbf{c}_j \in \mathbb{R}^3$ is projected into the Transformer embedding space using a positional MLP,

$$\mathbf{p}_j = \mathrm{MLP}_p(\mathbf{c}_j) \in \mathbb{R}^{d_e}. \tag{18}$$

The target positional embeddings are therefore

$$\mathbf{P}_t = \begin{bmatrix} \mathbf{p}_{t_1}^\top \\ \vdots \\ \mathbf{p}_{t_{N_t}}^\top \end{bmatrix} \in \mathbb{R}^{N_t \times d_e}. \tag{19}$$

The JEPA predictor receives the contextual representations together with the positional information of the masked target regions and predicts the corresponding target embeddings:

$$\widetilde{\mathbf{S}}_t = P_\phi\left(\mathbf{S}_c, \mathbf{P}_t\right) \in \mathbb{R}^{N_t \times d_e}. \tag{20}$$

Consequently, the complete point-based Gaussian JEPA mapping can be summarized as

$$\underbrace{\mathbb{R}^{M \times d_g}}_{\text{Gaussian attributes}} \xrightarrow{\text{FPS+KNN}} \underbrace{\mathbb{R}^{N \times k \times d_g}}_{\text{local Gaussian regions}} \xrightarrow{\tau_\psi} \underbrace{\mathbb{R}^{N \times d}}_{\text{Gaussian tokens}} \xrightarrow{F_\theta, F_\xi} \underbrace{\mathbb{R}^{N \times d_e}}_{\text{context/target embeddings}} . \tag{21}$$

The JEPA prediction step then operates entirely in the latent embedding space:

$$\underbrace{\mathbf{S}_c}_{N_c \times d_e}, \underbrace{\mathbf{P}_t}_{N_t \times d_e} \xrightarrow{P_\phi} \underbrace{\widetilde{\mathbf{S}}_t}_{N_t \times d_e} \approx \underbrace{\mathbf{S}_t}_{N_t \times d_e} . \tag{22}$$

The training objective is therefore applied to the predicted and target latent representations rather than directly to the Gaussian attributes:

$$\mathcal{L}_{\mathrm{JEPA}} = \frac{1}{N_t} \sum_{j \in \mathcal{T}} \mathrm{SmoothL1}\left(\widetilde{\mathbf{s}}_{t,j}, \mathbf{s}_{t,j}\right), \tag{23}$$

where $\widetilde{\mathbf{s}}_{t,j}, \mathbf{s}_{t,j} \in \mathbb{R}^{d_e}$.

D EDGE ATTRIBUTES CALCULATIONS

Depending on the selected configuration, the following edge attributes can be included:

$$\mathbf{e}_{ij} = \left[\mathbf{e}_{ij}^{\text{dist}}, \mathbf{e}_{ij}^{\text{vis}}, \mathbf{e}_{ij}^{\text{dir}}, \mathbf{e}_{ij}^{\text{scale}}, \mathbf{e}_{ij}^{\text{rot}}, \mathbf{e}_{ij}^{\text{app}}\right],$$

where

$$\mathbf{e}_{ij}^{\text{dist}} \in \mathbb{R}^{16}, \qquad \mathbf{e}_{ij}^{\text{vis}} \in \mathbb{R}^{2}, \qquad \mathbf{e}_{ij}^{\text{dir}} \in \mathbb{R}^{3},$$

$$\mathbf{e}_{ij}^{\text{scale}} \in \mathbb{R}^{3}, \qquad \mathbf{e}_{ij}^{\text{rot}} \in \mathbb{R}^{16}, \qquad \mathbf{e}_{ij}^{\text{app}} \in \mathbb{R}^{1}.$$

The final edge representation is obtained by concatenating all enabled attribute groups. Consequently, the dimensionality of the edge feature vector varies depending on the selected configuration.

To improve representational capacity for continuous geometric quantities, scalar distances and angular differences are expanded using Gaussian radial basis function (RBF) kernels.

Given a scalar quantity x, the RBF encoding is defined as

$$\phi_m(x) = \exp\bigl(-\gamma(x-\mu_m)^2\bigr), \qquad m = 1, \ldots, M,$$

where $\{\mu_m\}_{m=1}^{M}$ are uniformly spaced kernel centers and γ controls the kernel width. The resulting feature vector is

$$\boldsymbol{\phi}(x) = [\phi_1(x), \ldots, \phi_M(x)].$$

In all experiments, $M = 16$ kernels are used. The investigated edge attributes include:

- **RBF-encoded Euclidean distance**
  Let
  $$d_{ij} = \|\mathbf{x}_j - \mathbf{x}_i\|_2$$
  denote the Euclidean distance between Gaussian centroids. To improve representational capacity and reduce sensitivity to scale variations, distances are first normalized by the average neighborhood distance
  $$\tilde{d}_{ij} = \frac{d_{ij}}{\bar{d} + \varepsilon},$$
  where $\bar{d}$ denotes the mean Euclidean distance within the graph. The normalized distance is subsequently expanded using Gaussian RBF kernels
  $$\mathbf{e}_{ij}^{\text{dist}} = \boldsymbol{\phi}\Bigl(\tilde{d}_{ij}\Bigr).$$
- **Opacity relationship**
  $$o_{ij}^{\text{prod}} = o_i o_j$$
  $$o_{ij}^{\text{diff}} = |o_i - o_j|$$
  where o_i and o_j denote Gaussian opacities.
- **Relative direction vector**
  $$\mathbf{d}_{ij} = \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|_2}$$
  encoding the normalized direction from Gaussian i to Gaussian j.
- **Log-scale difference**
  $$\Delta\mathbf{s}_{ij} = \log(\mathbf{s}_j) - \log(\mathbf{s}_i)$$
  where $\mathbf{s}_i$ and $\mathbf{s}_j$ denote Gaussian scale vectors.
- **RBF-encoded rotational difference**
  Let $\mathbf{q}_i$ and $\mathbf{q}_j$ denote normalized quaternions representing Gaussian orientations. The angular difference between the two rotations is computed as
  $$\theta_{ij} = 2\arccos\left(|\mathbf{q}_i \cdot \mathbf{q}_j|\right).$$

The resulting angle is subsequently expanded using Gaussian RBF kernels

$$\mathbf{e}_{ij}^{\text{rot}} = \boldsymbol{\phi}(\theta_{ij})\,.$$

- **Appearance similarity**

$$a_{ij} = \hat{\mathbf{h}}_i \cdot \hat{\mathbf{h}}_j$$

where $\hat{\mathbf{h}}_i$ and $\hat{\mathbf{h}}_j$ are normalized spherical harmonics feature vectors.

E GROUPING ATTRIBUTES

The full Gaussian descriptor is not necessarily used for every stage of processing in either the point-based baseline or graph-based GS-JEPA. We therefore distinguish three attribute subsets according to their role in the representation pipeline. Let A_v denote the *vertex attributes* used to encode Gaussian information, A_g the *grouping attributes* used to determine local relationships, and A_{pe} the attributes used to construct positional embeddings for the predictor.

The vertex attributes A_v specify the Gaussian information used to construct the input representations. In the point-based formulation, these attributes are provided to the PointNet-based tokenizer to encode each local Gaussian patch. In the graph-based formulation, they are used to initialize the representation of each Gaussian node.

The grouping attributes A_g determine the distances used to establish local relationships between Gaussian primitives. For the point-based formulation, they determine the proximity between primitives when forming local patches around the sampled centroids. For graph-based GS-JEPA, they determine the KNN neighborhoods used to construct the Gaussian graph. Thus, different choices of A_g result in different local structures for predictive learning.

Finally, A_{pe} specifies the Gaussian attributes used to compute the positional embeddings provided to the predictor. In both formulations, these attributes are mapped to the positional embedding space and provide spatial or attribute-based information about the target locations during prediction.

The following experiments investigate how different attribute selections for A_v, A_g, and A_{pe} affect representation learning in the point-based and graph-based formulations. Fgure 7 illustrates how different grouping attributes A_g affect the local neighborhoods of Gaussian primitives.

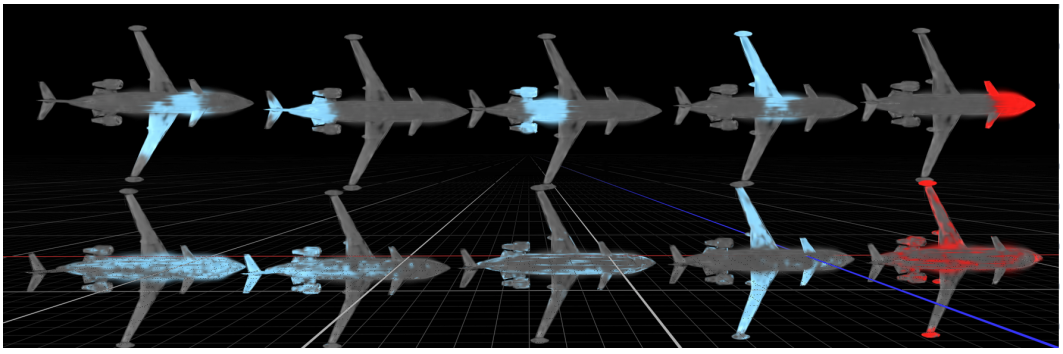

Figure 7: Visualization of the **target subgraphs** (blue) using 4 targets per sample with a ratio of 0.15–0.20, and the **context subgraphs** (red) obtained by selecting the remaining 0.60–0.80 portion of the graph. The first row represents the strategy based solely on the Euclidean distance between Gaussian centers where $A_g = \{x\}$. Second row shows the strategy using the Euclidean distance computed over all attributes $A_g = \{x, o, s, r, h\}$. Position-only approach creates coherent subgraphs that better represent the local structure. In contrast, the approach based on all attributes connects Gaussians that are most similar in their overall feature space, which is reflected in the highlighted patterns shown in the visualization.

F POINT-GS-JEPA DETAILS

The input representation consists of the Gaussian Splatting attributes

$$\mathcal{A} = \{\mathbf{x}, \mathbf{o}, \mathbf{s}, \mathbf{r}, \mathbf{h}\},$$

corresponding to position (xyz), opacity, scale, rotation, and spherical-harmonics coefficients, respectively. Each sample contains $N = 2048$ points and is processed using a PointNet-Small tokenizer. The tokenizer partitions the input into 128 groups with a group size of 32.

The encoder is a Transformer with embedding dimension

$$d_{\text{enc}} = 384,$$

12 Transformer blocks, and 6 attention heads. Encoder dropout and attention dropout are set to 0.1 and 0.05, respectively, with a stochastic-depth rate of 0.25. Positional information is added at every encoder layer. The predictor uses an embedding dimension of

$$d_{\text{pred}} = 192,$$

with 6 Transformer blocks and 6 attention heads. Its MLP expansion ratio is 4, while dropout, attention dropout, and stochastic-depth rate are set to 0.1, 0.05, and 0.25, respectively.

For the self-supervised objective, target and context groups are sampled contiguously. Each sample contains 4 target groups, with the target sampling ratio uniformly specified in the range

$$r_{\text{target}} \in [0.15,\ 0.20],$$

and the context ratio in

$$r_{\text{context}} \in [0.40,\ 0.75].$$

The target representation is extracted from the final part of encoder layer 11 and normalized using layer normalization.

During training, the point clouds are centered, normalized to the unit sphere, randomly scaled by a factor

$$s \sim [0.8,\ 1.2],$$

and rotated about the second spatial dimension. Translation augmentation is applied with magnitude 0.2. For validation, only centering and unit-sphere normalization are applied. The validation procedure is performed every 10 epochs.

G PRETRAINING CONFIGURATION

The prediction loss is the Smooth-L_1 loss.

Training is performed for 500 epochs using the AdamW optimizer with learning rate

$$\eta_0 = 10^{-3}$$

and weight decay

$$\lambda = 0.05.$$

The learning rate is linearly warmed up for the first 30 epochs, starting from

$$\eta_{\text{warmup}} = 10^{-5},$$

and subsequently follows a cosine decay schedule with minimum learning rate

$$\eta_{\min} = 10^{-6}.$$

The exponential moving average (EMA) coefficient is scheduled from

$$\tau_{\min} = 0.9995 \quad \text{to} \quad \tau_{\max} = 0.99999$$

over the full 500 training epochs. Mixed-precision training with 16-bit precision and synchronized batch normalization are enabled. The effective batch size is 256.

H Hyperparameter Evaluation

During pretraining, we evaluate different hyperparameter configurations to assess their influence on downstream performance and to identify the most relevant design choices. The following experiments provide a more detailed analysis of the graph-based GS-JEPA configuration used in the main experiments.

Influence of edge attributes on classification results. Table 5 evaluates the influence of progressively adding edge attributes to the graph-based GS-JEPA representation. The graph is constructed using Euclidean distance between Gaussian centroid positions, with $A_g = \{x\}$. Under full fine-tuning, incorporating additional edge attributes generally improves performance on both datasets. On ModelSplat40, the highest accuracy is obtained using the complete set of edge attributes, reaching 93.25%, compared with 92.97% without edge attributes. On MACGS, the improvements are more pronounced, with the complete configuration reaching 94.70% compared with 94.42% for the configuration without edge attributes. These results indicate that the additional relational information can be effectively exploited when the backbone is allowed to adapt to the downstream data.

The effect is different under linear probing, where the pretrained backbone remains frozen. On ModelSplat40, the complete edge-attribute configuration achieves the highest accuracy of 91.95%, although several intermediate configurations perform similarly. In contrast, MACGS achieves its highest accuracy of 87.50% without edge attributes, while adding edge information results in slightly lower performance. This difference suggests that the usefulness of edge attributes depends on the downstream evaluation setting: when the backbone can be fine-tuned, the model can adapt its relational representation to the target distribution, whereas a frozen backbone may be more sensitive to the particular relational information encoded during pretraining.

Influence of neighborhood size k. Table 6 evaluates the influence of the neighborhood size k used to construct the KNN graph. Under full fine-tuning, ModelSplat40 performs best with $k = 32$, while MACGS achieves its highest accuracy with $k = 16$. Under linear probing, $k = 32$ gives the best ModelSplat40 performance, whereas MACGS again performs best with $k = 16$. Overall, the results show that larger neighborhoods provide a more favorable configuration for ModelSplat40, while MACGS remains relatively stable across the tested values, with moderate neighborhood sizes providing the strongest performance.

Context-to-target ratio. For each object, we define two hyperparameters that control the construction of the context and target sets: the context ratio, which determine the proportion of patches/nodes assigned to the context, and the number of target patches, which determines how many target sets are sampled per object. First, we randomly sample the target size as a proportion of the object, fixed to 0.15-0.2 of its patches/nodes. For Point-JEPA, the target is constructed by selecting consecutive patches until the desired target proportion is reached. For Graph-GS-JEPA, we instead select a larger connected subgraph using a breadth-first search (BFS) strategy until the same target proportion is reached. This target sampling procedure is repeated according to the specified number of target patches, determining how many target regions are generated for each object.

Table 5: Influence of edge attributes on the graph-based GS-JEPA when the graph is constructed using Euclidean distance computed from Gaussian centroid positions ($A_g = \{x\}$). Dist, Dir, Vis, Scale, Rot, and App denote distance, relative direction, visibility, scale, rotation, and appearance edge attributes, respectively. Results are reported for both full finetuning and linear probing. The best result within each dataset and evaluation protocol is highlighted in bold.

Dist	Dir	Vis	Scale	Rot	App	ModelSplat	MACGS
Full Finetuning							
✗	✗	✗	✗	✗	✗	92.97 ± 0.10	94.42 ± 0.05
✓	✗	✗	✗	✗	✗	92.91 ± 0.20	94.51± 0.05
✓	✓	✗	✗	✗	✗	93.23 ± 0.16	94.52 ± 0.09
✓	✓	✓	✗	✗	✗	**93.42 ± 0.10**	94.60 ± 0.14
✓	✓	✓	✓	✗	✗	93.17 ± 0.02	94.42± 0.05
✓	✓	✓	✓	✓	✗	93.23 ± 0.16	94.52 ± 0.14
✓	✓	✓	✓	✓	✓	93.25 ± 0.10	**94.70 ± 0.32**
Linear Probing							
✗	✗	✗	✗	✗	✗	91.79 ± 0.02	**87.50 ± 0.05**
✓	✗	✗	✗	✗	✗	91.71 ± 0.02	86.44 ± 0.09
✓	✓	✗	✗	✗	✗	91.53 ± 0.09	86.62 ± 0.09
✓	✓	✓	✗	✗	✗	91.77 ± 0.06	86.67 ± 0.14
✓	✓	✓	✓	✗	✗	91.75 ± 0.08	86.53 ± 0.28
✓	✓	✓	✓	✓	✗	91.92 ± 0.06	86.81 ± 0.09
✓	✓	✓	✓	✓	✓	**91.95 ± 0.09**	86.44 ± 0.09

Table 6: Influence of the neighborhood size parameter k on the performance of the graph-based GS-JEPA. Results are reported for graphs constructed using centroid positions (x, y, z) under both full finetuning and linear probing. The best result within each dataset and evaluation protocol is highlighted in bold.

k	ModelSplat	MACGS
Full Finetuning		
32	93.25 ± 0.06	94.51 ± 0.14
24	93.23 ± 0.16	94.37 ± 0.09
16	93.15 ± 0.08	94.65 ± 0.19
8	92.71 ± 0.08	94.51 ± 0.32
Linear Probing		
32	91.94 ± 0.04	86.21 ± 0.05
24	91.53 ± 0.04	86.62 ± 0.09
16	91.57 ± 0.04	86.72 ± 0.28
8	90.76 ± 0.04	86.30 ± 0.05

From the patches/nodes remaining after target selection, we randomly sample the context according to the specified context ratio and use the same respective sampling strategy to construct the context. Thus, the two hyperparameters independently control how large the context is and how many target regions are generated per object. Table 7 evaluates the influence of the context-to-target ratio and the number of target patches used during GS-JEPA pretraining. Across the tested configurations, the 0.6-0.8 context ratio provides consistently competitive performance. Under full fine-tuning, the highest ModelSplat40 accuracy is obtained with four target patches, while MACGS performs best with three target patches. Under linear probing, four target patches give the strongest ModelSplat40 result, whereas MACGS achieves its highest accuracy with two target patches. Overall, the results indicate that performance is relatively robust to the choice of context and target configuration, with moderate context ratios providing a reliable setting across the evaluated tasks.

Gaussian attribute configuration. Table H evaluates the influence of the Gaussian attributes used for node encoding (A_v), graph construction (A_g), and positional encoding (A_{pe}) in graph-based GS-JEPA. Across the evaluated configurations, using the full Gaussian descriptor $\{x, o, s, r, h\}$ for node

Table 7: Influence of the context ratio used during graph-based GS-JEPA pretraining. The context ratio determines the proportion of visible nodes provided to the context encoder during predictive learning and the number of target patches determines how many targets we sample from one object. Each target patch has ratio 0.15-0.20

Context ratio	Target patches	ModelSplat	MACGS
Full Finetuning			
1.0-1.0	3	93.09 ± 0.22	94.33 ± 0.14
0.6-0.8	4	93.31 ± 0.08	94.46 ± 0.09
0.6-0.8	3	93.23 ± 0.16	94.87 ± 0.09
0.6-0.8	2	93.13 ± 0.06	94.74 ± 0.28
0.6-0.8	1	93.09 ± 0.06	94.37 ± 0.02
0.4-0.6	3	93.05 ± 0.26	94.33 ± 0.32
0.8-1.0	3	93.03 ± 0.04	94.45 ± 0.09
Linear Probing			
1.0-1.0	3	91.63 ± 0.14	86.56 ± 0.05
0.6-0.8	4	92.06 ± 0.03	86.85 ± 0.05
0.6-0.8	3	91.53 ± 0.04	86.62 ± 0.09
0.6-0.8	2	91.59 ± 0.14	87.92 ± 0.14
0.6-0.8	1	91.41 ± 0.16	87.45 ± 0.09
0.4-0.6	3	91.61 ± 0.05	86.90 ± 0.09
0.8-1.0	3	91.82 ± 0.04	86.58 ± 0.32

Table 8: Table consists of experiments with detail information about how attributes used in A_g, A_{pe} and A_v influence the results for graph-based GS-JEPA

A_v	A_g	A_{pe}	ModelSplat	MACGS
Full Finetuning				
x,o,s,r,h	x	x,s,r	93.15 ± 0.08	$\mathbf{94.56 \pm 0.09}$
x,o,s,r,h	x	x	$\mathbf{93.23 \pm 0.16}$	94.37 ± 0.09
x,o,s,r,h	x,s,r	x, s, r	92.44 ± 0.10	94.33 ± 0.05
x,o,s,r,h	x,s,r	x	92.77 ± 0.14	94.33 ± 0.14
x	x,s,r	x	91.05 ± 0.07	89.53 ± 0.14
x	x	x, s, r	89.85 ± 0.10	87.87 ± 0.05
x	x	x, s, r	91.24 ± 0.23	89.79 ± 0.31
Linear Probing				
x,o,s,r,h	x	x,s,r	$\mathbf{91.61 \pm 0.24}$	$\mathbf{87.64 \pm 0.37}$
x,o,s,r,h	x	x	91.53 ± 0.04	86.62 ± 0.09
x,o,s,r,h	x,s,r	x, s, r	90.72 ± 0.12	86.21 ± 0.14
x,o,s,r,h	x,s,r	x	91.39 ± 0.10	87.04 ± 0.23
x	x,s,r	x	89.22 ± 0.12	79.20 ± 0.42
x	x	x, s, r	83.63 ± 0.15	71.82 ± 0.05
x	x	x	89.38 ± 0.10	79.89 ± 1.27

encoding provides the strongest overall performance. In contrast, varying the attributes used for graph construction and positional encoding results in comparatively smaller changes in performance, with centroid-based grouping ($A_g = \{x\}$) providing consistently competitive results.

The largest performance differences arise when node encoding is restricted to centroid coordinates only. Compared with the full Gaussian descriptor, this configuration results in a substantial decrease in both ModelSplat40 and MACGS performance, particularly under linear probing. This indicates that retaining additional Gaussian attributes beyond centroid position is important for learning transferable representations, as the node embeddings can exploit information specific to the Gaussian representation that is not available from point coordinates alone.

Table 9: Influence of the context ratio used during GS-JEPA pretraining. Results are reported for both graph construction strategies (xyz and All) under full finetuning and linear probing. The context ratio determines the proportion of visible nodes provided to the context encoder during predictive learning.

Context ratio	Target patches	$mIoU_c$	$mIoU_i$
Full Finetuning			
1.0-1.0	3	83.42 ± 0.26	86.13 ± 0.02
0.6-0.8	4	83.40 ± 0.14	86.01 ± 0.08
0.6-0.8	3	84.17 ± 0.35	85.98 ± 0.12
0.6-0.8	2	83.38 ± 0.02	86.28 ± 0.02
0.6-0.8	1	83.17 ± 0.01	86.14 ± 0.06
0.4-0.6	3	83.26 ± 0.10	86.06 ± 0.02
0.8-1.0	3	83.91 ± 0.27	86.04 ± 0.05

I CENTROID-ONLY PRETRAINING

Table 10 evaluates the effect of restricting all Gaussian attributes to centroid coordinates during pretraining, i.e., $A_g = A_v = A_{pe} = \{x\}$. The values in parentheses report the change relative to the corresponding model pretrained using the full Gaussian descriptor.

Restricting the representation to centroid coordinates generally reduces downstream classification performance, with the effect varying across methods and evaluation protocols. Under full finetuning, the graph-based GS-JEPA exhibits the largest decrease, particularly on MACGS, where performance drops by 4.81 percentage points. The Point-GS-JEPA is more robust to this restriction on ModelSplat40, with a decrease of 0.69 percentage points, while Gaussian-MAE shows a 1.54 percentage-point decrease.

The effect is more pronounced under linear probing, particularly on MACGS, where both GS-JEPA formulations experience substantial performance drops. In contrast, Gaussian-MAE shows a smaller decrease on ModelSplat40 and an improvement on MACGS. Overall, the results indicate that the additional Gaussian attributes contribute substantially to the transferable representations learned by GS-JEPA, particularly for the graph-based formulation and under frozen-backbone evaluation.

Table 10: Classification accuracy (%) obtained when pretraining is performed using only Gaussian centroid coordinates ($A_g = A_v = A_{pe} = x$). Values in parentheses denote the performance difference relative to the corresponding model pretrained using the full Gaussian descriptor. Our methods are marked with †.

Method	**ModelSplat40**	**MACGS**
Self-Supervised Learning (FULL finetuning)		
Gaussian-MAE,	91.37 ± 0.42 (-1.54)	92.69 ± 0.31 (-0.62)
Point-GS-JEPA, †	92.45 ± 0.09 (-0.69)	93.86 ± 0.19 (-1.90)
Graph-GS-JEPA †	91.24 ± 0.23 (-2.18)	89.79 ± 0.31 (-4.81)
Self-Supervised Learning (LINEAR finetuning)		
Gaussian-MAE,	87.84 ± 0.64 (-2.54)	83.10 ± 0.55 (+1.55)
Point-GS-JEPA, †	91.19 ± 0.05 (-0.38)	83.78 ± 0.17 (-6.74)
Graph-GS-JEPA †	89.38 ± 0.91 (-2.39)	79.89 ± 1.27 (-6.78)
Self-Supervised Learning (MLP-3 finetuning)		
Gaussian-MAE,	90.60 ± 0.13 (-0.97)	89.10 ± 0.13 (+1.02)
Point-GS-JEPA, †	91.88 ± 0.07 (-0.32)	89.96 ± 0.07 (-3.47)
Graph-GS-JEPA, †	91.14 ± 0.14 (-1.20)	87.45 ± 0.32 (-4.62)

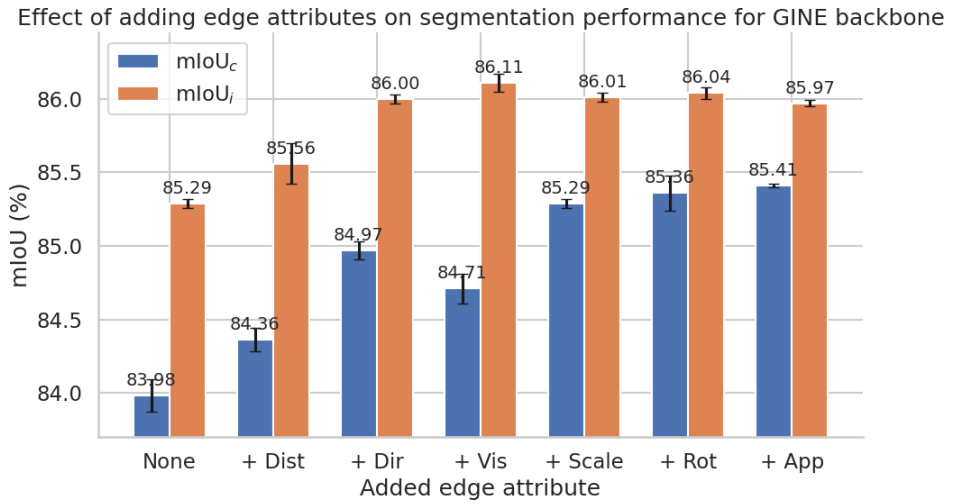

Figure 8: Segmentation results using GINE as backbone for Graph-GS-JEPA.

J GINE BACKBONE

GINE Hu et al. (2020) is evaluated as an alternative graph backbone because its architecture natively incorporates edge attributes into the message-passing operation. This makes it a natural alternative for the proposed Graph-GS-JEPA, where edge features explicitly encode geometric and appearance relationships between neighboring Gaussian primitives. Classification results are reported in Table 11, while the corresponding segmentation results are shown in Figure 8.

For object classification, Graphormer provides substantially stronger and more consistent performance than GINE across the evaluated datasets and protocols. The largest differences are observed under linear probing, where GINE reaches 89.10% and 79.98% on ModelSplat40 and MACGS, respectively, compared with 91.77% and 86.67% for Graphormer. Under full fine-tuning, Graphormer also performs better on both datasets. Overall, Graphormer provides a more robust backbone for the classification setting, although GINE remains competitive under selected evaluation conditions.

For part segmentation, GINE achieves competitive performance, with a clear influence of the incorporated edge attributes. As shown in Figure 8, progressively adding relational information generally improves segmentation performance compared with the configuration without edge attributes. This indicates that the explicit geometric and appearance relationships encoded by the edges can be effectively exploited by GINE for dense prediction, even though the backbone is less effective than Graphormer for the classification task.

Table 11: Comparison of Graph-GS-JEPA on classification task using different backbones including Graphormer and the GINE backbone

Method	**ModelSplat40**	**MACGS**
Self-Supervised Learning (FULL finetuning)		
Graph-GS-JEPA (GINE)†	91.98 ± 0.11	89.24 ± 0.26
Graph-GS-JEPA (Graphormer)†	93.42 ± 0.06	94.60 ± 0.14
Self-Supervised Learning (LINEAR finetuning)		
Graph-GS-JEPA (GINE)†	89.10 ± 0.12	79.98 ± 0.18
Graph-GS-JEPA (Graphormer)†	91.77 ± 0.06	86.67 ± 0.14
Self-Supervised Learning (MLP-3 finetuning)		
Graph-GS-JEPA (GINE)†	91.14 ± 0.08	91.27 ± 0.15
Graph-GS-JEPA (Graphormer)†	92.34 ± 0.14	92.07 ± 0.11