

Foundry: Distilling 3D Foundation Models for the Edge

Guillaume Letellier¹

Siddharth Srivastava²

Frederic Jurie¹

Gaurav Sharma³

¹ GREYC, Normandy University, Unicaen, ENSICAEN, UMR CNRS 6072, F-14000 Caen, France

² IIT Delhi, ³ IIT Kanpur

{firstname.lastname}@unicaen.fr

Abstract

Foundation models pre-trained with self-supervised learning (SSL) on large-scale datasets have become powerful general-purpose feature extractors. However, their immense size and computational cost make them prohibitive for deployment on edge devices such as robots and AR/VR headsets. Existing compression techniques like standard knowledge distillation create efficient ‘specialist’ models but sacrifice the crucial, downstream-agnostic generality that makes foundation models so valuable.

In this paper, we introduce Foundation Model Distillation (FMD), a new paradigm for compressing large SSL models into compact, efficient, and faithful proxies that retain their general-purpose representational power. We present Foundry, the first implementation of FMD for 3D point clouds. Our approach, Foundry, trains a student to learn a compressed set of SuperTokens that reconstruct the teacher’s token-level representations, capturing a compact basis of its latent space. A single distilled model maintains strong transferability across diverse downstream tasks—classification, part segmentation, and few-shot scenarios—approaching full foundation-model performance while using significantly fewer tokens and FLOPs, making such models more practical for deployment on resource-constrained hardware.

1. Introduction

The machine learning landscape is increasingly dominated by large-scale foundation models [5, 9, 13]. Pre-trained on vast, unlabeled datasets using self-supervised learning (SSL), these models serve as powerful, general-purpose feature extractors that can be adapted to a wide array of downstream tasks. This paradigm has shown immense success in 3D vision, with models trained on datasets of synthetic objects [41, 45], real-world scans [36, 40], and large-scale scenes [2, 7, 34] becoming the backbone for tasks in robotics, autonomous driving, and AR/VR.

However, the very scale that makes these models powerful also creates a significant deployment bottleneck. With hundreds of millions of parameters and quadratic attention complexity, models like PointTransformer [49] cannot be executed on resource-constrained devices. As we show, even a modern GPU can fail to process a moderately sized point cloud of 300k points as demonstrated in Sec. 4.4, let alone the million-point scenes common in real-world applications. This computational barrier prevents the power of 3D foundation models from reaching the devices where they are often needed most.

Existing compression techniques fall short of this goal because they trade generality for efficiency. The dominant approach, knowledge distillation (KD) [14], typically trains a student to mimic the teacher’s logits on a specific task, creating an efficient ‘specialist’. While recent works have explored distilling feature embeddings from large vision-language models like CLIP [43], these methods focus on preserving a specific cross-modal alignment capability via direct feature mimicry. This still results in a student specialized for tasks like zero-shot classification, rather than a truly general-purpose unimodal backbone. A fundamental gap remains for a method that can distill the entire representational manifold of a unimodal SSL model.

To bridge this gap, we explore a distillation paradigm we call *Foundation Model Distillation (FMD)*. The goal of FMD is not to create a task-specific specialist, but to forge a compact, portable, and efficient proxy of the original foundation model that retains its core identity as a general-purpose feature extractor. We introduce *Foundry*, the first framework to realize FMD for 3D point cloud Transformers. At the heart of Foundry is the concept of learnable *SuperTokens*. We train a lightweight student wrapper to compress the teacher’s dense set of token embeddings into a small, fixed-size set of SuperTokens, and then reconstruct the original embeddings from this compressed representation. This compress-and-reconstruct objective forces the student to learn a highly efficient basis for the teacher’s latent space, capturing its salient semantic and geometric fea-

tures in a compact form.

The result is a standalone student model that acts as a miniature foundation model. It can be cheaply fine-tuned for numerous downstream tasks without ever needing the original teacher again. Our contributions are:

- We propose Foundation Model Distillation (FMD), a new paradigm for creating compact, general-purpose proxies of large SSL models by distilling their entire representation space.
- We introduce Foundry, the first FMD framework for 3D point clouds. Its core novelty is a *compress-and-reconstruct* objective that forces a student to learn a compact set of SuperTokens as an efficient basis for the teacher’s latent space.
- We demonstrate that our distillation strategy is superior to both direct feature mimicry and task-specific KD, creating a student model with significantly better transferability and data efficiency.
- Our approach yields a portable and efficient 3D foundation model proxy, making large-scale 3D perception practical for resource-constrained GPUs.

2. Related Work

We review prior work on data preprocessing for efficiency and methods for compressing large Transformer models.

2.1. Efficient Data Preprocessing

Traditionally, to reduce the number of tokens fed to a Transformer, images can be downsampled to a lower resolution, videos can be processed with temporal frame subsampling and spatial resizing, and 3D data can be downsampled using grid sampling or Farthest Point Sampling (FPS) [12]. Later, manually designed methods used saliency maps [31] for 3D point clouds or quadtree-based spatial splitting and merging in STEP [26] to reduce the input before tokenizing. STTM [15] applies this idea directly to tokenized data. Other techniques are more specific to 3D point cloud compression and exploit geometric properties [1, 19, 23, 24, 46]. Finally, hand-designed algorithms based on graph clustering [28] can reduce input size by creating superpoints [30]. Our work differs fundamentally from these approaches: rather than pre-sampling points, we learn a compression of the dense feature space itself.

2.2. Model Compression for Transformers

Token Merging and Pruning. A popular line of work focuses on reducing the number of tokens processed ‘online’ within a Transformer to accelerate inference. Methods like ToMe [4] and PiToMe [35] progressively merge similar tokens between layers during a single forward pass. Other techniques, such as PatchMerger [29], learn a codebook to merge tokens at specific blocks within the network. For 3D

point clouds, 3DLST [22] also proposes a learnable super-token module for a supervised segmentation task. While these methods share procedural similarities with our work, their fundamental goal is different. They are designed as architectural modifications to accelerate an existing model for a specific, pre-defined task. In contrast, our work uses a token compression mechanism not for online acceleration, but as the core of an *offline distillation framework* whose sole purpose is to create a new, standalone, and general-purpose student model. Our focus is on distilling a transferable representation space, not on optimizing a single forward pass.

Distillation of Foundation Models. Knowledge Distillation (KD) [14] is a powerful technique for model compression. When applied to foundation models, the specific knowledge being transferred is critical. We identify two main categories:

1. *Distillation of Task-Specific Capabilities.* Most prior work distills a specific capability of a large model. In Vision-Language Models (VLMs), this is often the cross-modal alignment. For instance, TinyCLIP [39] and other works [25] distill the contrastive loss or image-text affinity scores to create a smaller model that is good at zero-shot retrieval or classification. PromptKD [18] focuses on domain-specific distillation. While effective, these methods produce ‘specialist’ students that inherit one key function of the teacher, not its full, general-purpose representational power.

2. *Distillation via Feature Mimicry.* A more direct approach is to force a student’s feature embeddings to match the teacher’s. CLIP-KD [43] provides a comprehensive study of this, using an L2 loss to align the student’s and teacher’s feature outputs. This is a form of direct *feature mimicry*. While this transfers more general information than logit-based KD, it is still fundamentally different from our approach.

Our work, Foundry, establishes a third category: *Representation Space Distillation*. We do not use direct mimicry. Instead, we propose a *compress-and-reconstruct* objective. By forcing the student to learn a compressed representation (the SuperTokens) that can decompress back to the teacher’s full set of feature embeddings, we compel it to learn a robust and efficient basis for the teacher’s entire latent space. This information bottleneck objective is fundamentally different and, as we show experimentally, results in a more versatile and transferable student model, creating a true, unimodal foundation model proxy.

3. Approach

The core of Foundry is a *compress-and-reconstruct* framework designed to implement our Foundation Model Distillation (FMD) paradigm. The objective is to train a lightweight student model to learn a compact, yet powerful, representation that can faithfully approximate the rich latent

space of a large, pre-trained teacher model. We first detail this overarching framework and its learning objective, then describe the novel components that enable it. An overview is provided in Fig. 1.

3.1. The FMD Framework: Compressing and Reconstructing the Latent Space

Our distillation process consists of three conceptual steps, as illustrated in Fig. 1.

1. Teacher Forward Pass (The Ground Truth). First, we establish the ‘ground truth’ representation we aim to distill. For a given input point cloud \mathbb{P} , we follow the standard tokenization process of Point-BERT [47] to produce a set of c initial token embeddings $\mathbf{T} \in \mathbb{R}^{c \times d}$, augmented with positional encodings. These tokens are processed by the powerful, pre-trained teacher encoder, yielding a set of rich output embeddings $\mathbf{Y} \in \mathbb{R}^{c \times d}$. This set \mathbf{Y} represents the target latent space that we want our student to be able to model.

2. Student Forward Pass (Compress and Reconstruct). The student model’s goal is to reproduce \mathbf{Y} without the computational cost of the teacher. It takes the same initial tokens \mathbf{T} and performs a two-stage process:

- *Compress*: The initial tokens \mathbf{T} are fed into our *Dynamic Supertoken Optimization (DSO)* module. This module compresses the information from all c tokens into a much smaller, learnable set of s *Supertokens* ($\mathbf{S} \in \mathbb{R}^{s \times d}$), where $s \ll c$. These Supertokens form the compressed basis for the teacher’s latent space.
- *Reconstruct*: The compressed Supertokens \mathbf{S} are then processed by a lightweight student encoder. The updated Supertokens, along with the original input tokens \mathbf{T} , are passed to our *Cross-Attention Upsampling (CAU)* module. This module uses the compressed representation to reconstruct the teacher’s full latent space, producing the final student output $\hat{\mathbf{Y}} \in \mathbb{R}^{c \times d}$.

3. The Distillation Objective. The entire student model is trained end-to-end with a simple, direct objective: the reconstructed output $\hat{\mathbf{Y}}$ must match the teacher’s ground truth output \mathbf{Y} . We use a Smooth L1 loss¹ for this reconstruction objective:

$$\mathcal{L}_{\text{distillation}} = \text{SmoothL1}(\hat{\mathbf{Y}}, \mathbf{Y}). \quad (1)$$

This loss function directly enforces our FMD goal. By successfully minimizing this reconstruction error, the student is forced to learn Supertokens that are an efficient and information-rich basis for the teacher’s general-purpose representation space.

¹We also experimented with MSE and obtained similar results when fine-tuning, but L1 is more robust to outliers, so it becomes our default choice.

3.2. Learning the Compressed Basis: Dynamic Supertoken Optimization (DSO)

The compression stage is achieved by our DSO module, inspired by 3DLST [22], which is responsible for learning the Supertokens.

Supertokens as a Learnable Basis. The set of Supertokens, $\mathbf{S} = \{s_i\}_{i=1}^s \in \mathbb{R}^{s \times d}$, is a randomly initialized, learnable parameter of our model with truncated normal distribution. Conceptually, it acts as a small, shared memory or a set of basis vectors that, over the course of distillation, will learn to represent the most salient semantic and geometric concepts found across the entire training dataset.

Information Aggregation via Cross-Attention. To compress the input tokens \mathbf{T} into the Supertokens \mathbf{S} , we use a cross-attention mechanism [38]. The Supertokens act as queries, “attending” to the input tokens (keys and values) to gather the most relevant information. This is formulated as:

$$\mathbf{Q} = \mathbf{S}\mathbf{W}_Q, \mathbf{K} = \mathbf{T}\mathbf{W}_K, \mathbf{V} = \mathbf{T}\mathbf{W}_V, \quad (2)$$

where $\mathbf{W}_{Q,K,V}$ are learnable projection matrices. We then compute a hard assignment² by identifying the most similar Supertoken for each of the c input tokens. This produces a one-hot **Cross-Attention Map (CAM)** of size $c \times s$, where each row has a single ‘1’ indicating the chosen Supertoken:

$$\text{CAM}_{j,i} = \begin{cases} 1 & \text{if } i = \arg \max_k \left(\frac{\mathbf{q}_k \cdot \mathbf{k}_j}{\sqrt{d}} \right) \\ 0 & \text{otherwise} \end{cases}, \quad (3)$$

for each input token $j \in \{1, \dots, c\}$ and Supertoken $i \in \{1, \dots, s\}$. This map is then used to perform a grouped average, updating each Supertoken with the mean of the value vectors of all input tokens assigned to it:

$$\mathbf{S}_{\text{updated}} = \frac{\text{CAM}^T \mathbf{V}}{\text{sum}(\text{CAM}^T, \text{axis}=1)}. \quad (4)$$

Here, $\text{CAM}^T \mathbf{V}$ (a matrix of size $s \times d$) sums the value vectors for each group, and the division by the token counts (a vector of size s) normalizes each sum to produce the mean. This updated set of Supertokens is then passed to the student encoder. Notably, we perform this semantic grouping before incorporating positional embeddings, ensuring the Supertokens learn to represent features based on content rather than location. The positional embeddings are merged in the same way separately and added to the Supertokens before the student encoder.

3.3. Reconstructing the Latent Space: Cross-Attention Upsampling (CAU)

After the lightweight student encoder has processed and refined the Supertokens, the reconstruction stage is handled by the CAU module, adapted from 3DLST.

²Using Gumbel-Softmax [16]

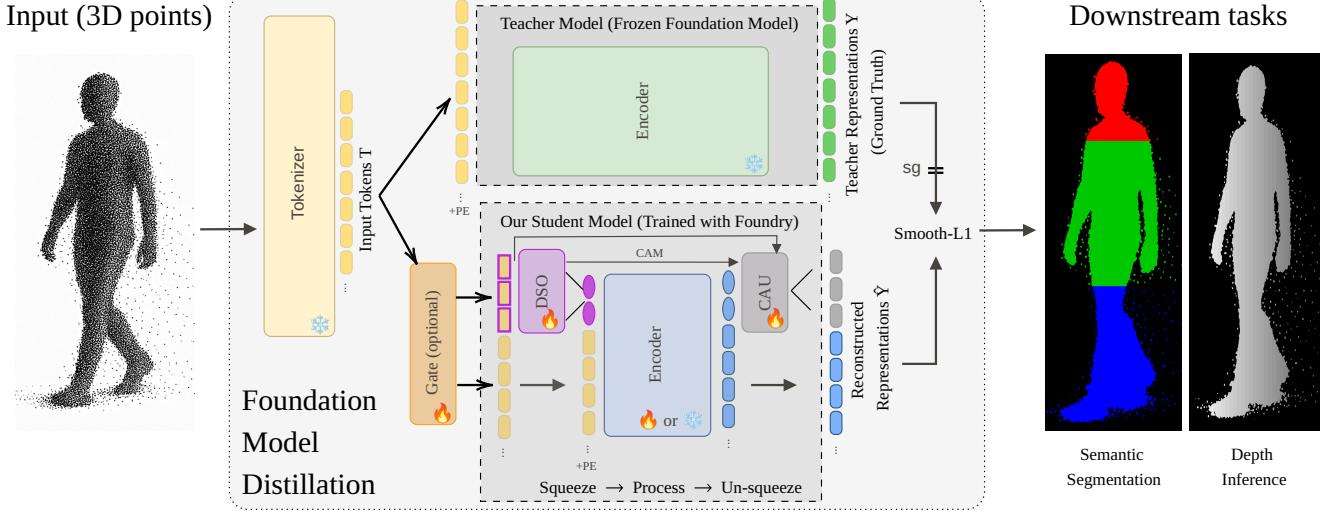


Figure 1. Overview of the Foundry Framework. Our distillation strategy follows a compress-and-reconstruct pipeline. The student model uses a Dynamic Supertoken Optimization (DSO) module to compress the input tokens into a small set of learnable SuperTokens. After processing by a lightweight encoder, a Cross-Attention Upsampling (CAU) module reconstructs an approximation of the teacher’s latent space. The entire student is trained to minimize the reconstruction error, forcing the SuperTokens to become a powerful, compact basis for the teacher’s representations.

Upsampling via Information Look-up. To compute the distillation loss, we must return from the compressed s -token space to the original c -token space. The CAU module achieves this by allowing each original token location to ‘look up’ the relevant refined information from the updated SuperTokens. We reuse the CAM computed during the DSO step (Eq. (3)) as a routing mechanism. The reconstruction is formulated as:

$$\hat{\mathbf{Y}} = \text{MLP}(\mathbf{T} + \text{CAM} \cdot \mathbf{S}_{\text{encoder,out}}) \in \mathbb{R}^{c \times d}. \quad (5)$$

A residual connection with the initial tokens \mathbf{T} is crucial, as it reinjects local, high-frequency details that may have been lost during the compression, ensuring a high-fidelity reconstruction.

3.4. Budget-Aware Inference via Gated Compression

While our core framework creates a statically efficient model, we can extend it with a gating mechanism to enable dynamic, on-the-fly budget control at inference time.

Gate Mechanism. We add a simple 2-layer MLP gate that takes the input tokens \mathbf{T} and predicts a fusion probability π_i for each token. At inference time, only tokens with $\pi_i > r$ (where r is a user-defined threshold) are compressed by the DSO module. The remaining tokens bypass the compression and are processed alongside the SuperTokens in the student encoder. This allows a user to trade accuracy for performance by simply adjusting the threshold r .

Regularization Objective. To train the gate to make meaningful selections, we add a regularization term to the distil-

lation loss, which encourages the gate to activate:

$$\mathcal{L}_{\text{gate}} = -\lambda_{\text{gate}} \sum_{i=1}^c \pi_i. \quad (6)$$

The final objective function when the gate is enabled is $\mathcal{L} = \mathcal{L}_{\text{distillation}} + \mathcal{L}_{\text{gate}}$.

3.5. Complexity Analysis

We denote by c the number of input tokens, by s the maximum number of SuperTokens that can be selected (with $s \ll c$), by d the embedding dimension, by l the number of layers of the Transformer encoder, and by r the ratio of selected tokens when the gate is used. Table 1 shows complexity of each encoder and wrapper. When l is very large, the computational overhead induced by the wrapper becomes marginal.

4. Experimental Results

We present experiments designed to validate our Foundation Model Distillation paradigm and the effectiveness of our Foundry framework. Our evaluation is structured to answer three key questions in order of importance:

1. **Is the FMD paradigm effective?** We first demonstrate that distilling a general-purpose student is superior to creating task-specific specialists, especially for transfer learning and data efficiency (Sec. 4.1).

2. **Is our SuperToken mechanism novel and necessary?**

We then show that our compress-and-reconstruct ob-

Method	Encoder complexity	Wrapper complexity	Total
Transformer	$\mathcal{O}(lc^2d + lcd^2)$	-	$\mathcal{O}(lc^2d + lcd^2)$
Foundry (Ours)	$\mathcal{O}(ls^2d + lsd^2)$	$\mathcal{O}(2scd + cd^2)$	$\mathcal{O}(scd + ls^2d + lsd^2 + cd^2)$
Foundry-Gate (Ours)	$\mathcal{O}(l(s + (1 - r)c)^2d + l(s + (1 - r)c)d^2)$	$\mathcal{O}(cd + 2srd + cd^2)$	$\mathcal{O}(srcd + l(s + (1 - r)c)^2d + l(s + (1 - r)c)d^2 + cd^2)$

Table 1. **Complexity analysis.** For a small s , we see Foundry needs less computation to perform a forward pass compared to a regular Transformer. Foundry-Gate takes another parameter r which can be fixed by the user or dynamic by the gate itself. In the worst case $r = 0$ (and $s = 0$ because no token selected for merging), we match Transformer complexity and best complexity of Foundry.

jective with learnable SuperTokens outperforms simpler compression baselines (Sec. 4.2).

3. **How does Foundry perform?** Finally, we benchmark our validated method against other token compression techniques and analyze its performance, budget-awareness, and computational cost (Sec. 4.3 and beyond).

All experiments were performed on a ViT-S teacher (12 layers, 384 embedding dimensions) pre-trained with Point-JEPA [32]. Results with ViT-T are specified.

4.1. Validating the Foundation Model Distillation Paradigm

The core claim of our work is that creating a general-purpose student is more valuable than distilling specialists for each task. We validate this by comparing our single Foundry student against specialist students trained with traditional, task-specific knowledge distillation.

Experimental Setup. We compare two approaches:

- **FMD-Student (Ours):** A single Foundry student with 16 supertokens, distilled once on the general-purpose ShapeNet55 dataset [6]. This is our “generalist” model.
- **Specialist-Students:** Two students with an identical architecture to ours. The first (*Specialist-SEG*) is trained via traditional KD to mimic the teacher’s outputs on the ShapeNetPart [44] segmentation task. The second (*Specialist-CLS*) is trained similarly on the ShapeNet55 classification task. Both are only using KL divergence loss [17] for aligning the student and teacher output distributions without using ground truth labels.

Generalist vs. Specialist on Downstream Tasks. We evaluate both native-task performance and transferability. As shown in Tab. 2, our generalist FMD-Student is highly competitive with the specialist models on their own area. For example, it achieves 89.95% accuracy on ShapeNet55 classification, close to the specialist trained for that task. However, the true advantage of the FMD paradigm is revealed in transfer learning. When the specialist students are distilled, their performance collapses (e.g., the classification specialist’s accuracy drops by nearly 15%). In contrast, our single generalist student demonstrates robust and stable performance across both tasks, proving it has successfully inherited the teacher’s general-purpose representational power.

Few-Shot Generalization. The robustness of a general-

Method	SN55 (Acc. \uparrow)	SNP (mIoU _C /mIoU _I \uparrow)
Teacher (Point-JEPA)	90.54	83.91/85.73
Foundry (Ours)	89.95	81.85/84.77
Specialist-CLS	75.09*	-
Specialist-SEG	-	61.88/65.72*

Table 2. **Generalist vs. Specialist Distillation.** Our single FMD-Student, distilled once on ShapeNet55, maintains high performance when fine-tuned on both classification and segmentation. In contrast, small specialist students (marked with *), when distilled on the outputs of their respective specialist teachers to compress inputs to 16 supertokens, fail to perform well even on their native task, underscoring their limited effectiveness.

purpose model is most evident in low-data regimes. We fine-tune our distilled student on few-shot subsets of ModelNet40 following Sharma and Kaul protocol [33]. As shown in Tab. 3, our student retains a remarkable degree of the teacher’s few-shot learning capability, even with extreme compression down to a single SuperToken. For example, it achieves 91.8% top-1 accuracy in a 10-shot setting, demonstrating that the distilled representations provide a strong semantic prior for learning from scarce data.

Method	5-way		10-way	
	10-shot	20-shot	10-shot	20-shot
Point-JEPA	96.1 ± 3.45	98.2 ± 1.32	93.4 ± 4.23	95.3 ± 3.35
Foundry (Ours) ($s = 16$)	92.5 ± 4.35	95.2 ± 2.82	89.8 ± 4.93	92.2 ± 4.96
Foundry (Ours) ($s = 1$)	91.8 ± 4.02	95.1 ± 3.18	88.3 ± 6.56	91.5 ± 4.79

Table 3. **Few-shot transfer performance on ModelNet40.** Even with a single SuperToken, the gateless student achieves 91.8% (10-shot) and 95.1% (20-shot) accuracy, approaching the Point-JEPA teacher.

4.2. The Importance of the SuperToken Mechanism

Having established the value of the FMD paradigm, we now prove that our specific compress-and-reconstruct mechanism with learnable SuperTokens is the key to its success. We compare Foundry against two strong baselines designed to create a compressed student via simpler means.

Baselines.

- **KMeans-Student:** Replaces our learnable DSO/CAU modules with a static K-Means [20] clustering algorithm.

We first compute K-Means centroids on the tokenizer’s output embeddings and then fine-tune a student to target task. This tests the value of end-to-end learning.

- **FPS-Student (New Baseline):** Replaces our mechanism with naive pre-sampling. We use Farthest Point Sampling (FPS) [11] to select a sparse subset of input tokens *before* they are processed, and distill a student on this sparse representation. This tests if a learned compression is better than a simple geometric sampling.

As shown in Tab. 4, our full Foundry model significantly outperforms both baselines. Substituting the learnable DSO with static K-Means causes a massive accuracy drop of over 13% on ShapeNet55. This proves that learning the Super-Token basis vectors *jointly* with the distillation process is critical. Furthermore, our method also surpasses the FPS-Student, demonstrating that our learned semantic compression captures richer information than a simple geometric pre-sampling of the input space.

Method / Token Grouping Strategy	ShapeNet55 (Acc. \uparrow)
Foundry (Ours, learnable DSO and CAU)	89.68
KMeans-Student (static clustering)	76.08
FPS-Student (pre-sampling)	87.56

Table 4. **Effect of the token-grouping strategy.** Our learnable DSO mechanism significantly outperforms baselines using static K-Means clustering or naive Farthest Point Sampling. This validates that our end-to-end, attention-based aggregation is essential for creating a high-fidelity student. All model encoders are completely frozen during distillations and fine-tunings using 16 super-tokens, prototypes or centers.

4.3. Benchmark Performance and Analysis

Overall gateless performance. After validating our core claims, we now benchmark the performance of our gateless Foundry variant against the teacher and other token compression methods across a wide range of datasets. Six classification datasets (ModelNet40 [41] (MN40), OmniObject3D [8] (OO3D), the three ScanObjectNN [36] (SONN) splits, and ShapeNet55 [6] (SN55)) and one part segmentation (ShapeNetPart [44] (SNP)) are benchmarked.

As shown in Tab. 5, despite compressing the latent representation to a small basis of $s \ll c$ tokens, Foundry maintains accuracy within 1-2% of the full teacher model on most datasets, mainly on synthetics and within a few points on challenging real-world datasets. While acknowledging that averaging different metrics (Accuracy and mIoUs) is not strictly rigorous, the ‘Avg’ column provides a useful high-level indicator of overall performance trends. This summary view suggests that our method consistently outperforms naive baselines like random sampling and exhibits more stable cross-task performance than methods like ToMe and PiToMe. For instance, with just $s = 4$ SuperTo-

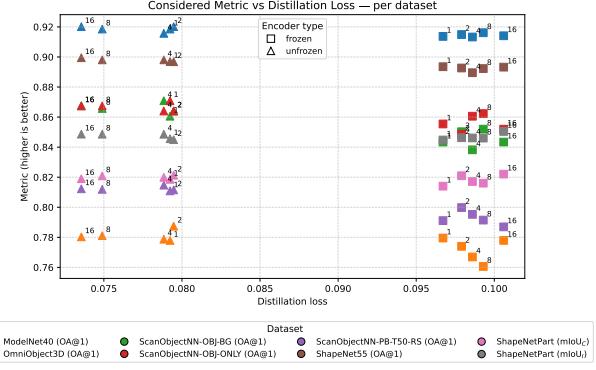


Figure 2. **Relationship between distillation loss and fine-tuning accuracy.** Each point corresponds to a distilled model with a different number of SuperTokens (s) across multiple datasets. A clear inverse correlation is observed, with diminishing improvements beyond $s \leq 4$, indicating that reconstruction fidelity saturates once the student sufficiently spans the teacher’s latent space.

kens, the student achieves 91.25% accuracy on ModelNet40 and 84.67/84.81 mIoU on ShapeNetPart. This indicates that our learned SuperTokens capture a truly reusable and task-agnostic latent basis. Unlike inference-based methods, we see that our FMD method can be combined via compress-and-reconstruct objective with other methods, such as ToMe or PatchMerger, coming very close to baseline performance while adapting the model to process increasingly compressed information.

Distillation loss as a proxy for transferability. Fig. 2 illustrates the relationship between distillation loss and fine-tuning performance. A clear inverse correlation is observed, with diminishing gains beyond $s \leq 4$, indicating that once the compressed basis sufficiently spans the teacher’s latent space, further capacity increases yield marginal benefit. This suggests that the model has adapted to processing compressed tokens rather than the original, uncompressed tokens.

Selective compression We analyze the dynamic, budget-aware variant Foundry-Gate, which integrates the gating mechanism introduced in Sec. 3.4. The gate regularizer λ_{gate} controls the proportion of tokens that bypass compression, allowing on-the-fly trade-offs between computational cost and representational fidelity. We compute the Spearman correlation between the final fine-tuning performances and the used regularization value. We observe anti-correlation, for both frozen and unfrozen settings. On ModelNet40, the correlation values are -0.45 and -0.13 respectively for frozen and unfrozen distillation. When frozen, anti-correlation is more important because the model is not updated during distillation. Complete results are available in supplementary material. Figure 3 shows that by reducing the number of tokens passed to the core encoder, the perfor-

# supertokens or tokens (c)	Frozen student	Avg	MN40	OO3D	SONN-OBJ-BG	SONN-OBJ-ONLY	SONN-PB-T50-RS	SN55	SNP	
									mIoU _C	mIoU _I
baseline ($c = 64$)										
-	-	87.72	93.02 ± 0.14	82.25 ± 0.37	91.84 ± 0.47	88.38 ± 0.41	86.05 ± 0.42	90.54 ± 0.14	83.91 ± 0.25	85.73 ± 0.15
Foundry (Ours)										
$s = 16$	✓	84.33	91.41 ± 0.32	77.30 ± 0.61	84.74 ± 0.43	84.97 ± 0.72	79.93 ± 1.10	89.56 ± 0.21	81.88 ± 0.47	84.82 ± 0.32
$s = 16$	✗	84.87	91.75 ± 0.25	77.80 ± 0.21	86.23 ± 0.46	86.29 ± 0.43	80.36 ± 0.90	89.87 ± 0.09	81.87 ± 0.04	84.82 ± 0.07
$s = 16$ (ViT-T)	✗	83.75	91.49 ± 0.42	75.98 ± 0.24	83.42 ± 0.87	84.80 ± 0.98	79.38 ± 0.17	88.67 ± 0.33	81.68 ± 0.24	84.56 ± 0.10
$s = 8$	✓	84.38	91.71 ± 0.10	76.48 ± 0.36	85.60 ± 0.55	85.14 ± 0.95	80.28 ± 1.00	89.42 ± 0.17	81.84 ± 0.34	84.57 ± 0.05
$s = 8$	✗	84.76	91.63 ± 0.20	77.32 ± 0.72	86.23 ± 0.34	86.06 ± 1.05	80.46 ± 0.73	89.61 ± 0.24	81.95 ± 0.19	84.85 ± 0.01
$s = 4$	✓	84.25	91.25 ± 0.14	76.96 ± 0.28	84.45 ± 0.60	85.94 ± 1.04	79.61 ± 0.14	89.26 ± 0.27	81.84 ± 0.18	84.67 ± 0.08
$s = 4$	✗	84.64	91.48 ± 0.08	76.69 ± 1.02	86.00 ± 1.22	85.89 ± 0.52	80.70 ± 0.93	89.74 ± 0.06	81.78 ± 0.30	84.81 ± 0.07
$s = 2$	✓	84.17	91.52 ± 0.53	77.09 ± 0.32	84.51 ± 0.62	84.51 ± 1.08	79.98 ± 0.14	89.34 ± 0.14	81.84 ± 0.36	84.55 ± 0.11
$s = 2$	✗	84.87	91.76 ± 0.22	78.32 ± 0.66	85.83 ± 0.53	86.12 ± 0.50	80.78 ± 0.33	89.66 ± 0.04	81.98 ± 0.21	84.47 ± 0.04
$s = 1$	✓	84.17	91.60 ± 0.30	77.19 ± 0.67	84.80 ± 0.53	85.03 ± 0.75	79.47 ± 0.33	89.29 ± 0.13	81.46 ± 0.08	84.54 ± 0.10
$s = 1$	✗	84.58	91.72 ± 0.14	77.72 ± 0.08	84.57 ± 1.30	86.06 ± 0.96	80.65 ± 0.38	89.65 ± 0.06	81.76 ± 0.14	84.50 ± 0.09
<i>Random sampling - inference from baseline</i>										
$c = 16$	-	75.20	90.57 ± 0.41	65.19 ± 0.81	78.95 ± 1.19	72.70 ± 1.14	69.74 ± 0.91	87.80 ± 0.22	66.96 ± 0.71	69.71 ± 0.16
$c = 1$	-	21.46	18.33 ± 0.66	03.82 ± 0.32	23.22 ± 1.55	21.31 ± 1.35	19.90 ± 0.68	15.54 ± 0.23	33.70 ± 0.30	35.89 ± 0.16
<i>Updating group size to reduce number of generated tokens - inference from baseline</i>										
$c = 16$	-	41.66	63.45 ± 0.47	13.29 ± 0.42	11.76 ± 0.16	22.24 ± 0.29	12.74 ± 0.17	59.99 ± 0.19	72.98 ± 0.21	76.84 ± 0.08
$c = 1$	-	15.78	03.88 ± 0.19	00.47 ± 0.13	09.24 ± 0.63	15.85 ± 0.82	09.37 ± 0.00	04.25 ± 0.09	40.30 ± 0.44	42.90 ± 0.34
<i>ToMe [4] - inference from baseline (no additional training needed)</i>										
$s = 16$	-	73.80	89.10 ± 0.23	50.51 ± 0.39	82.19 ± 0.56	84.22 ± 0.62	73.87 ± 0.50	84.89 ± 0.24	63.09 ± 0.40	62.51 ± 0.08
$s = 8$	-	66.79	84.02 ± 0.38	33.54 ± 0.78	76.87 ± 1.30	79.88 ± 0.96	67.32 ± 0.62	79.68 ± 0.26	56.81 ± 0.43	56.23 ± 0.14
$s = 4$	-	59.88	74.92 ± 0.65	20.91 ± 0.59	71.57 ± 1.05	74.29 ± 0.54	61.52 ± 0.49	72.05 ± 0.30	51.59 ± 0.52	52.19 ± 0.12
$s = 2$	-	53.19	62.76 ± 0.44	11.21 ± 0.36	67.76 ± 1.03	68.49 ± 0.84	56.30 ± 0.62	61.66 ± 0.30	47.47 ± 0.45	49.88 ± 0.12
$s = 1$	-	46.15	48.35 ± 0.31	07.20 ± 0.20	61.20 ± 0.89	62.87 ± 0.95	51.69 ± 0.46	42.15 ± 0.18	45.36 ± 0.19	50.37 ± 0.09
<i>ToMe [4] - Trained using FMD (Ours) + finetuning</i>										
$s = 16$	✗	86.52	93.07	80.95	90.02	87.44	83.66	89.84	82.41	84.76
$s = 1$	✗	85.87	92.14	80.08	89.85	86.40	83.00	89.84	81.14	84.50
<i>PiToMe [35] - Trained using FMD (Ours) + finetuning</i>										
$s = 16$	✗	86.51	92.83	81.97	89.33	87.95	83.66	89.88	81.77	84.65
$s = 1$	✗	86.14	92.55	80.95	90.19	86.23	82.89	89.76	81.84	84.74
<i>PatchMerger [29] - Trained using FMD (Ours) + finetuning</i>										
$s = 16$	✗	87.59	92.83	82.76	91.39	88.99	85.53	90.11	83.59	85.55
$s = 1$	✗	87.14	92.18	82.13	90.71	88.12	85.05	89.71	83.64	85.54

Table 5. Performance of the gateless Foundry student across benchmarks. Each student is distilled from a frozen foundation teacher using the compress–reconstruct objective. Despite compressing to $s \ll c$ latent tokens, the distilled models remain within 1–2 % of teacher accuracy on most classification and segmentation tasks, and within a few points on challenging real-world datasets. We vary the number of SuperTokens $s \in \{1, 2, 4, 8, 16\}$ and report mean ± standard deviation over multiple runs. For Foundry, random token sampling and changing token group size, s and c correspond to the number of tokens at the input and output of the student encoder. For ToMe, PiToMe, and PatchMerger, it is the number of tokens we seek to obtain at the output. Complete per-dataset results and variance analyses are reported in the supplementary material.

mance degradation compared to the baseline model trained with $r = 0.5$ is acceptable. However, when r is increasing (less tokens are fused), performance degrades more significantly (see supplementary material). This suggests that the model has adapted to process compressed tokens rather than on the original, uncompressed tokens.

4.4. Empirical Computational Cost

We evaluate runtime, memory, and computational efficiency on an NVIDIA RTX A3000 6GB Laptop GPU. Table 6 reports results for object-level inference , and Table 7 reports large-scale scene inference (2^{18} points). Reported experiments include tokenization and encoding steps.

Small-scale inference. Relative to the baseline’s 478 GFLOPs, Foundry reduces forward-pass cost to **178 G** ($s = 16$), **156 G** ($s = 8$), and **137 G** ($s = 1$), while maintaining comparable accuracy. Average latency decreases from 0.09 s to 0.05–0.06 s without memory overhead. The ViT-T variant further reduces computation to **144 GFLOPs**.

at similar accuracy, demonstrating the scalability of the framework to smaller backbones.

Large-scale scenes. In the indoor-scene simulation (Tab. 7), both the baseline and ToMe exceed the 6 GB VRAM limit and fail to execute, whereas Foundry completes inference with a **4.0 GB** footprint and **4.6 s** forward time (throughput ≈ 0.22 obj/s). This confirms that the proposed distillation enables practical deployment on edge or mobile-class GPUs, where full foundation models are otherwise infeasible.

Dynamic compute control. As shown in the lower blocks of Tab. 6, Foundry-Gate allows specifying a fixed token budget or directly adjusting the fusion ratio r at inference. Varying r between 0.1 and 0.9 modulates latency predictably while incurring data-sensitive degradation in downstream accuracy, enabling real-time compute adaptation under resource constraints.

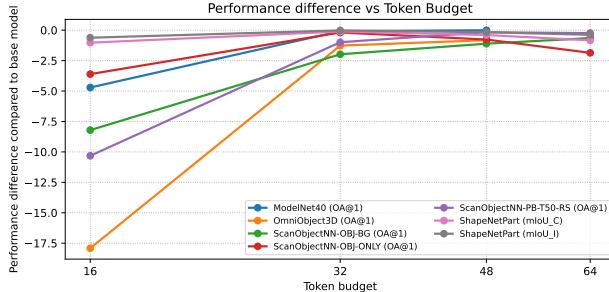


Figure 3. Finetuning results vs token budget for Foundry-Gate. We use the best backbone on each dataset with $\lambda_{\text{gate}} = 10^{-11}$ and no weight decay on the gate module. The token budget represents the maximum number of tokens we accept as input to the core encoder (r is dynamic and changes for each input example). We report top-1 overall accuracy for classification tasks and mIoU per class and per instance for part segmentation tasks. All metrics are computed on the validation set for each dataset.

Method	# supertokens	Token budget	Params (M)	MACs (G)	FLOPs (G)	GPU Mem (MiB)	Avg runtime (s)	Throughput (obj/s)
baseline	-	-	21.84	238.624	478.128	1143.0	0.09	747.11
ToMe [4] ($r_{\text{token}} = 16$)	-	-	21.84	115.412	231.319	1143.0	0.06	1091.42
ToMe [4] ($r_{\text{token}} = 8$)	-	-	21.84	97.8966	196.233	1143.0	0.05	1174.07
ToMe [4] ($r_{\text{token}} = 4$)	-	-	21.84	90.0449	180.505	1143.0	0.05	1213.10
ToMe [4] ($r_{\text{token}} = 2$)	-	-	21.84	86.572	173.548	1143.0	0.05	1219.81
ToMe [4] ($r_{\text{token}} = 1$)	-	-	21.84	85.0621	170.523	1143.0	0.05	1217.40
Foundry (Ours)								
Foundry	16	-	22.73	88.9903	178.394	1135.5	0.06	1160.97
Foundry (ViT-T)	16	-	6.33	71.6618	143.669	1073.0	0.05	1335.26
Foundry	8	-	22.73	78.042	150.664	1135.5	0.05	1271.38
Foundry	4	-	22.73	72.5678	145.498	1135.5	0.05	1309.76
Foundry	2	-	22.73	69.8307	140.016	1135.5	0.05	1349.58
Foundry	1	-	22.73	68.4622	137.274	1135.5	0.05	1365.92
Foundry-Gate	16	-	22.78	157.593	315.812	1135.7	0.11	391.56
Foundry (Ours) - Inference token-budgeted								
Foundry-Gate	16	16	22.78	87.8332	176.076	1135.7	0.07	864.02
Foundry-Gate	16	32	22.78	107.651	215.774	1135.7	0.08	756.92
Foundry-Gate	16	48	22.78	126.073	252.674	1135.7	0.09	779.59
Foundry (Ours) - change fusion ratio r								
Foundry-Gate ($r=0.1$)	16	-	22.78	87.8332	176.076	1135.7	0.07	881.45
Foundry-Gate ($r=0.2$)	16	-	22.78	89.1921	178.798	1135.7	0.07	875.99
Foundry-Gate ($r=0.3$)	16	-	22.78	159.254	319.139	1135.7	0.09	751.83
Foundry-Gate ($r=0.4$)	16	-	22.78	161.028	322.694	1135.7	0.11	588.67
Foundry-Gate ($r=0.5$)	16	-	22.78	157.593	315.812	1135.7	0.11	593.70
Foundry-Gate ($r=0.6$)	16	-	22.78	152.006	304.621	1135.7	0.10	623.29
Foundry-Gate ($r=0.7$)	16	-	22.78	152.006	304.621	1135.7	0.10	623.66
Foundry-Gate ($r=0.8$)	16	-	22.78	152.006	304.621	1135.7	0.10	620.73
Foundry-Gate ($r=0.9$)	16	-	22.78	152.006	304.621	1135.7	0.10	623.58

Table 6. Runtime and memory comparison on 64×1024-point inputs. Foundry reduces FLOPs from 478 G to 178–137 G while maintaining comparable accuracy. Wall-clock latency decreases from 0.09 s to 0.05–0.06 s with reduced used memory. We provide also inference values on CPU in supplementary material.

4.5. Ablation Study

We conduct architectural variations as ablations to isolate the contributions of key architectural components. Other key points are discussed in Secs. 4.1 and 4.2. Tab. 8 compares several architectural alternatives. Replacing the hard arg max assignment with soft attention slightly decreases the reconstruction loss (0.101→0.095). Introducing QKV biases or CAU layer normalization yields similarly marginal changes, indicating that the design is robust to small architectural perturbations. Unfreezing the student during distillation reduces the loss from 0.1006 to 0.0736. This clearly shows that allowing the student to learn enables it to process compressed information more effectively and demonstrates

Method	# supertokens	Token budget	Params (M)	MACs (G)	FLOPs (G)	GPU Mem (MiB)	Avg runtime (s)	Throughput (obj/s)
baseline	-	-	21.84					<i>OOM</i>
ToMe [4]	-	-	21.84					<i>OOM</i>
Foundry (Ours)								
Foundry	16	-	22.73	268.719	538.821	4034.1	4.59	0.22
Foundry (ViT-T)	16	-	6.33	264.837	531.037	3971.5	4.57	0.22
Foundry	8	-	22.73	268.547	538.476	4034.1	4.57	0.22
Foundry	4	-	22.73	268.461	538.304	4034.1	4.58	0.22
Foundry	2	-	22.73	268.418	538.218	4034.1	4.56	0.22
Foundry	1	-	22.73	268.396	538.174	4034.1	4.60	0.22
Foundry (Ours) - Inference token-budgeted								
Foundry-Gate	16	16	22.78	269.526	540.438	4034.3	4.57	0.22
Foundry-Gate	16	32	22.78	269.836	541.058	4034.3	4.57	0.22
Foundry-Gate	16	64	22.78	270.497	542.382	4034.3	4.57	0.22
Foundry-Gate	16	128	22.78	271.818	545.028	4034.3	4.57	0.22
Foundry-Gate	16	256	22.78	274.46	550.321	4034.3	4.57	0.22
Foundry-Gate	16	512	22.78	279.745	560.907	4034.3	4.60	0.22
Foundry-Gate	16	1024	22.78	290.315	582.08	4034.3	4.59	0.22

Table 7. Scalability of Foundry on large indoor scenes (2^{18} points). The baseline and ToMe exceed 6 GB VRAM and fail to execute, whereas Foundry completes inference with a 4.0 GB footprint and ~4.6 s forward time. This demonstrates memory-efficient scalability suitable for edge deployment.

once again that the compressive learning objective that has been put in place is sufficient.

Ablations	Frozen student	Distillation loss
Foundry	✓	0.1006
Foundry	✗	0.0736
arg max → softmax	✗	0.0787
Bias on QKV in DSO	✗	0.0763
Output layer norm at CAU	✗	0.0748

Table 8. Architectural ablations of the distillation framework. Soft assignment, QKV biases, and CAU layer normalization each decrease the reconstruction loss slightly (e.g., 0.101→0.095). Unfreezing the student during distillation further reduces the loss (0.1006→0.0736), indicating that most benefits arise from the compressive learning objective itself and that giving the model the ability to update its weights helps it better process compressed tokens.

5. Conclusion

This paper introduced Foundation Model Distillation (FMD), a new paradigm for making large SSL models practical for the edge. We presented Foundry, the first FMD framework, which successfully distills a general-purpose 3D foundation model into a compact proxy that is a fraction of the size. Our core contribution, a compress-and-reconstruct objective based on learnable SuperTokens, creates a student that retains the teacher’s transferability and data efficiency—a feat that task-specific distillation fails to achieve. By producing a portable, standalone, and efficient model, Foundry paves the way for deploying powerful 3D perception on resource-constrained systems, from autonomous robots to AR headsets. Our results indicate that it is possible to distill a foundation model into a compact backbone that preserves its transferability across classification, segmentation, and few-shot tasks, while reducing computational cost. We focus on a single 3D self-supervised

teacher (Point-JEPA) and leave extending Foundry to other 3D foundation models and modalities for future work; it remains to be seen how well our compress–reconstruct design transfers in those settings.

Acknowledgement. This work was supported by the Agence Nationale pour la Recherche (ANR) under award number ANR-19-CHIA-0017. This work was performed using HPC resources from GENCI-IDRIS (Grant 2024-AD011014730R1).

References

- [1] Anique Akhtar, Zhu Li, and Geert Van Der Auwera. Inter-frame compression for dynamic point cloud geometry coding. *CoRR*, abs/2207.12554, 2022.
- [2] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *CVPR*, pages 1534–1543, 2016.
- [3] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013.
- [4] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. In *ICLR*. OpenReview.net, 2023.
- [5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 33:1877–1901, 2020.
- [6] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing 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. *CoRR*, abs/1512.03012, 2015.
- [7] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, pages 5828–5839, 2017.
- [8] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *CVPR*, pages 13142–13153, 2023.
- [9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186, 2019.
- [10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.
- [11] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Trans. Image Process.*, 6(9):1305–1315, 1997.
- [12] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y Zeevi. The farthest point strategy for progressive image sampling. *IEEE TIP*, 6(9):1305–1315, 1997.
- [13] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, pages 16000–16009, 2022.
- [14] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.
- [15] Jeongseok Hyun, Sukjun Hwang, Su Ho Han, Taeoh Kim, Inwoong Lee, Dongyoon Wee, Joon-Young Lee, Seon Joo Kim, and Minho Shim. Multi-granular spatio-temporal token merging for training-free acceleration of video llms. *CoRR*, abs/2507.07990, 2025.
- [16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.
- [17] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.
- [18] Zheng Li, Xiang Li, Xinyi Fu, Xin Zhang, Weiqiang Wang, Shuo Chen, and Jian Yang. Promptkd: Unsupervised prompt distillation for vision-language models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 26607–26616. IEEE, 2024.
- [19] Kai Liu, Kang You, and Pan Gao. Diff-pcc: Diffusion-based neural compression for 3d point clouds. *CoRR*, abs/2408.10543, 2024.
- [20] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.
- [21] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*. OpenReview.net, 2019.
- [22] Dening Lu, Linlin Xu, Jun Zhou, Kyle (yilin) Gao, and Jonathan Li. 3dlst: 3d learnable supertoken trans-

- former for lidar point cloud scene segmentation. *Int. J. Appl. Earth Obs. Geoinformation*, 140:104572, 2025.
- [23] Dat Thanh Nguyen, Maurice Quach, Giuseppe Valenzise, and Pierre Duhamel. Learning-based lossless compression of 3d point cloud geometry. In *ICASSP*, pages 4220–4224. IEEE, 2021.
- [24] Jiahao Pang, Kevin Bui, and Dong Tian. Pivot-net: Heterogeneous point-voxel-tree-based framework for point cloud compression. *CoRR*, abs/2402.07243, 2024.
- [25] Renjing Pei, Jianzhuang Liu, Weimian Li, Bin Shao, Songcen Xu, Peng Dai, Juwei Lu, and Youliang Yan. CLIPPING: distilling clip-based models with a student base for video-language retrieval. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 18983–18992. IEEE, 2023.
- [26] Mathilde Proust, Martyna Poreba, Michal Szczepanski, and Karim Haroun. STEP: supertoken and early-pruning for efficient semantic segmentation. In *VISIGRAPP*, pages 50–61. SCITEPRESS, 2025.
- [27] Charles Ruizongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, pages 5099–5108, 2017.
- [28] Hugo Raguet and Loic Landrieu. Parallel cut pursuit for minimization of the graph total variation. *ICML*, 2019.
- [29] Cédric Renggli, André Susano Pinto, Neil Houlsby, Basil Mustafa, Joan Puigcerver, and Carlos Riquelme. Learning to merge tokens in vision transformers. *CoRR*, abs/2202.12015, 2022.
- [30] Damien Robert, Hugo Raguet, and Loïc Landrieu. Efficient 3d semantic segmentation with superpoint transformer. In *ICCV*, pages 17149–17158. IEEE, 2023.
- [31] Pietro Ruiu, Lorenzo Mascia, and Enrico Grossi. Saliency-guided point cloud compression for 3d live reconstruction. *Multimodal Technol. Interact.*, 8(5):36, 2024.
- [32] Ayumu Saito, Prachi Kudeshia, and Jiju Poovvancheri. Point-jepa: A joint embedding predictive architecture for self-supervised learning on point cloud. In *WACV*, pages 7348–7357. IEEE, 2025.
- [33] C. Sharma and M. Kaul. Self-supervised few-shot learning on point clouds. *NeurIPS*, 33:7212–7221, 2020.
- [34] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *CVPR*, pages 2446–2454, 2020.
- [35] Chau Tran, Duy M. H. Nguyen, Manh-Duy Nguyen, TrungTin Nguyen, Ngan Le, Pengtao Xie, Daniel Sonntag, James Y. Zou, Binh Nguyen, and Mathias Niepert. Accelerating transformers with spectrum-preserving token merging. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.
- [36] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *ICCV*, pages 1588–1597, 2019.
- [37] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 9(86):2579–2605, 2008.
- [38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 30, 2017.
- [39] Kan Wu, Houwen Peng, Zhenghong Zhou, Bin Xiao, Mengchen Liu, Lu Yuan, Hong Xuan, Michael Valenzuela, Xi Stephen Chen, Xinggang Wang, Hongyang Chao, and Han Hu. Tinyclip: CLIP distillation via affinity mimicking and weight inheritance. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*, pages 21913–21923. IEEE, 2023.
- [40] Tong Wu, Jiarui Zhang, Xiao Fu, Yuxin Wang, Jiawei Ren, Liang Pan, Wayne Wu, Lei Yang, Jiaqi Wang, Chen Qian, et al. Omniobject3d: Large-vocabulary 3d object dataset for realistic perception, reconstruction and generation. In *CVPR*, pages 803–814, 2023.
- [41] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–1920, 2015.
- [42] Runsen Xu, Xiaolong Wang, Tai Wang, Yilun Chen, Jiangmiao Pang, and Dahua Lin. Pointllm: Empowering large language models to understand point clouds. In *ECCV*, pages 131–147. Springer, 2024.
- [43] Chuanguang Yang, Zhulin An, Libo Huang, Junyu Bi, Xinqiang Yu, Han Yang, Boyu Diao, and Yongjun Xu. CLIP-KD: an empirical study of CLIP model distillation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 15952–15962. IEEE, 2024.
- [44] Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM TOG*, 35(6):1–12, 2016.

- [45] Li Yi, Lin Shao, Manolis Savva, Haibin Huang, Yang Zhou, Qirui Wang, Benjamin Graham, Martin Engelcke, Roman Klokov, Victor S. Lempitsky, Yuan Gan, Pengyu Wang, Kun Liu, Fenggen Yu, Panpan Shui, Bingyang Hu, Yan Zhang, Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Minki Jeong, Jaehoon Choi, Changick Kim, Angom Geetendra, Narasimha Murthy, Bhargava Ramu, Bharadwaj Manda, M. Ramanathan, Gautam Kumar, P. Preetham, Siddharth Srivastava, Swati Bhugra, Brejesh Lall, Christian Häne, Shubham Tulsiani, Jitendra Malik, Jared Lafer, Ramsey Jones, Siyuan Li, Jie Lu, Shi Jin, Jingyi Yu, Qixing Huang, Evangelos Kalogerakis, Silvio Savarese, Pat Hanrahan, Thomas A. Funkhouser, Hao Su, and Leonidas J. Guibas. Large-scale 3d shape reconstruction and segmentation from shapenet core55. *CoRR*, abs/1710.06104, 2017.
- [46] Kang You, Tong Chen, Dandan Ding, M. Salman Asif, and Zhan Ma. RENO: real-time neural compression for 3d lidar point clouds. *CoRR*, abs/2503.12382, 2025.
- [47] 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. In *CVPR*, pages 19313–19322, 2022.
- [48] Karim Abou Zeid, Jonas Schult, Alexander Hermans, and Bastian Leibe. Point2vec for self-supervised representation learning on point clouds. In *GCPR*, pages 131–146. Springer, 2023.
- [49] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *ICCV*, pages 16259–16268, 2021.

Foundry: Distilling 3D Foundation Models for the Edge

Supplementary Material

A. Datasets and metrics

This section briefly reports information for each used dataset in this paper.

A.1. Datasets

ShapeNet55 [6] (SN55). The dataset contains synthetic 3D meshes from 55 categories, split into 41,952 shapes for training and 10,518 for validation and testing. Below is the list of shape names: *airplane, trash bin, bag, basket, bathtub, bed, bench, birdhouse, bookshelf, bottle, bowl, bus, cabinet, camera, can, cap, car, cellphone, chair, clock, keyboard, dishwasher, display, earphone, faucet, file cabinet, guitar, helmet, jar, knife, lamp, laptop, loudspeaker, mailbox, microphone, microwaves, motorbike, mug, piano, pillow, pistol, flowerpot, printer, remote, rifle, rocket, skateboard, sofa, stove, table, telephone, tower, train, watercraft, washer*.

ModelNet40 [41] (MN40). The dataset consists of synthetic 3D meshes from 40 distinct classes, with 9,840 samples for training and 2,468 for validation and testing. The shape names are: *airplane, bathtub, bed, bench, bookshelf, bottle, bowl, car, chair, cone, cup, curtain, desk, door, dresser, flower pot, glass box, guitar, keyboard, lamp, laptop, mantel, monitor, night stand, person, piano, plant, radio, range hood, sink, sofa, stairs, stool, table, tent, toilet, tv stand, vase, wardrobe, xbox*.

ScanObjectNN [36] (SONN). ScanObjectNN comprises 2,304 training examples and 581 validation/test samples from 15 semantic classes. The samples are real-scanned objects divided into three splits: (i) OBJ-BG includes both object and background, (ii) OBJ-ONLY contains only the object, and PB-T50-RS objects and backgrounds perturbed samples with challenging random augmentations to increase classification complexity. Below is the list of shape names: *bag, bin, box, cabinet, chair, desk, display, door, shelf, table, bed, pillow, sink, sofa, toilet*.

OmniObject3D [40] (OO3D). The dataset is divided into training and validation/testing sets with a 80/20 split per class. Typically, for a class with 100 elements, the last 20 are used for validation/testing, and the rest for training. If a class has 2 or fewer elements, at least one is used for testing and one for training. The final split counts are 4,641 for training, and 1,270 for validation/testing. The dataset contains 216 categories, including: *anise, antique, apple, asparagus, backpack, ball, bamboo shoots, banana, battery, beauty blender, bed, belt, biscuit, book, bottle, bowl, box, boxed beverage, bread, broad bean, broccoli, broccolini, brush, brussels sprout, bucket noodle, bumbag, bun, burrito, cabinet, cake, calculator, candle, candy, canned beverage, carrot, cauliflower, chair, cheese, cherry, chess, chicken leg, chili, china, chinese chess, chocolate, clock, coconut, conch, corn, cucumber, cup, dice, dinosaur, dish, doll, donut, drawing, drum, dumbbell, dumpling, durian, dustbin, earplug, egg, egg tart, eraser, facial cream, fan, fig, fire extinguisher, flash light, flower pot, flute, fork, frisbee, garage kit, garlic, ginger, glasses, glasses case, gloves, green bean cake, guitar, hair dryer, hairpin, hamburger, ham melon, hammer, hand cream, handbag, hat, haw thorn, hazelnut, helmet, hot dog, house, insole, kennel, kettle, keyboard, kite, kiwifruit, knife, laptop, laundry detergent, lemon, light, lipstick, litchi, longan, loquat, lotus root, magnet, mango, mangosteen, medicine bottle, microwaveoven, monitor, mooncake, mouse, mushroom, nipple, onion, orange, ornament, ornaments, oyster, package, pad, pan, pancake, pastry, peach, peanut, pear, pen, picnic basket, pie, pillow, pineapple, pinecone, pistachio, pitaya, pizza, plant, plug, pomegranate, popcorn, potato, potato chips, power strip, projector, puff, pumpkin, razor, red jujube, red wine glass, remote control, rice cake, ricecooker, rubik cube, sandwich, sausage, scissor, shampoo, shaomai, shoe, shrimp, skateboard, soap, sofa, spanner, speaker, squash, starfish, steamed bun, stool, strawberry, suitcase, sushi, sweet potato, sword bean, table, table tennis bat, tape measure, teapot, teddy bear, thermos, thimble, timer, tissue, tomato, tooth brush, tooth paste, toothpick box, toy animals, toy boat, toy bus, toy car, toy motorcycle, toy plane, toy plant, toy train, toy truck, tvstand, umbrella, vase, waffle, wallet, walnut, watch, water chestnut, watermelon, whistle, yam, zongzi*.

ShapeNetPart [44] (SNP). This part segmentation dataset contains 16 object types and 50 part categories, comprising 16,881 synthetic 3D shapes. The data is split into a training set of 13,998 shapes and a validation/testing set of 2,874 shapes.

Objaverse [8]. This unannotated dataset contains 661,575 objects in the train set and 3000 in the validation set. PointLLM [42] provides annotations via description on each object but it was not utilized in our framework.

A.2. Metrics

Classification task. We use top-1 overall accuracy which can be sometimes abbreviated as OA@1.

Part segmentation task. We use mean intersection over union abbreviated as mIoU for classes and instances. We denote them respectively by mIoU_C and mIoU_I. When unspecified, mIoU_C is used as mIoU.

B. Implementation details

We give some details about the implementation of each tested method.

B.1. Pre-trainings

Point-JEPA [32]. We pre-train a Transformer [38] encoder using the Point-JEPA method. The size of the encoder has the same hyper-parameters as ViT-S [10]. We use the exact same hyper-parameters and architecture for pre-training except floating-point precision (float32 vs float16), and we apply truncated normal as weight initialization function [9] with rescaling w.r.t. layer depth (GPT-2 layer-wise weight rescaling). Results can diverge a bit from the initial paper due to these additions. This model is called the *teacher* or *baseline*.

B.2. Distillations

Foundry. Figure 1 in the main paper illustrates the core idea. We use the architecture described before with our pre-training. As said in the main paper, for the DSO block we use Gumbel-Softmax [16] for differentiability. In contrast to 3DLST [22], we do not double the output channel of CAU for recovering the teacher token dimension.

Foundry-Gate. The Gate is a 2-layer MLP with a fixed hidden dimension of 128. The final gate activation is a sigmoid function to produce a probability for each token. For making it differentiable, we use the straight-through [3] method when training.

Specialists. For the specialists presented in Tab. 2 in main paper, we use fine-tuned models (pre-trained with Point-JEPA) on two datasets, ShapeNet55 (Specialist-CLS) and ShapeNetPart (Specialist-SEG). For each, we have a classical Transformer. We apply distillation only with KL divergence [17] without ground truth attachment to only use the teacher model as we present. We distill on their respective datasets to obtain a set of supertokens for compressing input tokens. We freeze the core encoder, tokenizer, and positional embedder but allow the heads to be updated during the process.

FPS-Student. The idea is to distill an existing backbone with a fixed number of centers for tokenizing the input point cloud to another fixed number of centers with its particular tokenizer. We apply the same distillation strategy as Foundry while changing the architecture so that the new student may be compatible with distillation. Instead of using one single grouping function with $c = 64$ and $k = 32$, we use a second tokenizer with another grouping function with $c_2 = 16$ and $k = 2 \cdot p/c_2$ with p being the number of the initial point cloud (we set 2 for adding overlap between groups). Note that the data augmentation is shared for the two branches as well as the first c sampled centers up to c_2 which are sampled by the Farthest Point Sampling (FPS) [11] algorithm. As the number of output tokens differs between teacher and student, we use a feature upsampler [27] to recover the last tokens of c by interpolating existing c_2 to c .

ToMe [4] and PiToMe [35]. We use our strategy to distill both methods. For obtaining 16 output tokens, we set $r_{tome} = [32, 16]$ which after the first attention block, removes 32 tokens and at the second, removes another 16. Same for PiToMe but using token ratio instead of a fixed number of tokens to end with. The ten last layers process only 16 tokens. When we have 1 single output token, $r_{tome} = [32, 16, 8, 4, 2, 1]$. We track the source to be able to place information of merged tokens. For merged tokens, we put information of the corresponding remaining token. We also set proportional attention to True to add a bias in the attention matrix based on already fused tokens. We track and add proportional bias during distillation and fine-tunings.

PatchMerger [29]. Once again, we use our strategy to distill the encoder with PatchMerger. For obtaining 16 output tokens, we use the default hyper-parameter by placing the module between the 6th and 7th Transformer layers. We track the source to be able to place information of merged tokens. For merged tokens, we put information of the corresponding remaining token. We track when distilling and fine-tuning.

B.3. Fine-tunings

Classification. We use the architecture and method defined in Point-JEPA for all models except KMeans-Student. For this method, we tokenize all training samples and use them for creating s prototypes with K-Means [20] clustering algorithm. After, we encode these prototypes into a frozen pre-trained backbone. Finally, we retrieve all tokens using the assignation matrix and we pool tokens by concatenating mean+max as Point2Vec [48] does for probing and classifying to obtain one single token to use as input for our head.

Part segmentation. We use the architecture and method defined in Point-JEPA.

Zero-Shot Classification. We use the architecture and method defined in Point-JEPA.

Linear Probing. We use the architecture and method defined in Point-JEPA.

B.4. Inferences

Random Sampling. We use fine-tuned models and change at inference-time the number of tokens given to the encoder. We tokenize the input point cloud and obtain c centers. We randomly subsample c to c_{rs} which we set beforehand to match the number of supertokens s we want to match.

Larger group size. We use fine-tuned models and change at inference-time the c and k values. We reduce c , and k is obtained via the following calculation $k = 2 \cdot p/c$.

Foundry-Gate. For a fixed r value and dynamic setting with token-budget, we use fine-tuned models and we change r at inference-time without making any additional training. Specifically, when we budget the number of tokens to pass to the encoder, we select a specific r value to have $c - s$ tokens to merge into s supertokens maximum. We provide the code in Listing 1.

```

1  class BudgetAwareMLPTokenSelector(MLPTokenSelector):
2      def __init__(self, token_budget: int, num_supertokens: int, embed_dim: int, hidden_dim:
3          int = 128, act: Callable = nn.GELU()) -> None:
4          super().__init__(embed_dim, hidden_dim, act)
5          self.token_budget = token_budget
6          self.num_supertokens = num_supertokens
7
8      def _gate_forward(self, x: torch.Tensor) -> torch.Tensor:
9          B, T, _ = x.shape
10
11         h = self.act(self.fc1(x))
12         logits = self.fc2(h)  # (B, T, 1)
13         probs = torch.sigmoid(logits).view(B, T, 1)
14         return probs
15
16     def _select_ratio(self, probs: torch.Tensor):
17         B, T, _ = probs.shape
18
19         if T <= self.token_budget:
20             return 1.0
21
22         sorted_vals, _ = probs[:, :, 0].sort(dim=1, descending=True)
```

```

23     num_to_select = T - self.token_budget + self.num_supertokens
24     if num_to_select == T:
25         fusion_ratio = probs.new_full((B,), -torch.finfo(probs.dtype).eps)
26     else:
27         fusion_ratio = sorted_vals[:, num_to_select+1] # we put +1 because in forward
28             we use > (if)
29
30     return fusion_ratio.reshape(B, 1, 1).repeat(1, T, 1)
31
32 def forward(self, x: torch.Tensor, inference: bool = False, **kwargs) ->
33     MLPTokenSelectorOutput:
34     _, T, _ = x.shape
35
36     probs = self._gate_forward(x)
37     fusion_ratio = self._select_ratio(probs)
38
39     selection_hard = (probs > fusion_ratio).float()
40     if inference: # non differentiable
41         selection = selection_hard
42     else: # straight-through method
43         selection = probs + (selection_hard - probs).detach()
44
45     mask = (selection > fusion_ratio).squeeze(-1) # (B, T), boolean
46     seqlens_selected = mask.sum(dim=1)
47
48     final_num_tokens = (T - seqlens_selected + self.num_supertokens).max().item()
49     if final_num_tokens > self.token_budget:
50         print("more_tokens_used_than_token_budget, diff:", final_num_tokens - self.
51             token_budget)
52
53     outputs = MLPTokenSelectorOutput(
54         weighted_tokens=x * selection + x * (1 - selection),
55         selected_tokens=x[mask],
56         selected_seqlens=seqlens_selected,
57         unselected_tokens=x[~mask],
58         unselected_seqlens=T - seqlens_selected,
59         selected_indices=torch.nonzero(mask.flatten(), as_tuple=False).flatten(),
60         unselected_indices=torch.nonzero(~mask).flatten(), as_tuple=False).flatten(),
61         probs=probs,
62         selection=selection,
63         mask=mask,
64     )
65
66     return outputs
67
68 @staticmethod
69 def from_mlp_token_selector(gate: MLPTokenSelector, token_budget: int, num_supertokens:
70     int) -> "BudgetAwareMLPTokenSelector":
71     new_gate = BudgetAwareMLPTokenSelector(
72         token_budget,
73         num_supertokens,
74         embed_dim=gate.embed_dim,
75         hidden_dim=gate.hidden_dim,
76         act=gate.act,
77     )
78     new_gate.load_state_dict(gate.state_dict())
79
80     if gate.training:

```

```

77     new_gate.train()
78
79     return new_gate

```

Listing 1. Implementation of the token-budgeted Foundry-Gate. We use PyTorch as the deep learning framework. For quick prototyping, we implement only for one single input sample.

ToMe. We use fine-tuned models and change at inference-time the number of tokens at the encoder output. We choose r_{tome} accordingly to the desired final number of output tokens. We track source tokens only when resolving the part segmentation task.

C. Theoretical FLOPs Computation

This section presents the theoretical FLOPs computation for our modules and final models. All expressed formulas are for 32-bit floating point operations. We denote S, N, D, nh, hd, d, L for the maximum number of supertokens, input number of tokens, embedding dimension, number of heads, head dimension, gate hidden dimension, and number of Transformer layers respectively.

C.1. Transformer

$$\text{MLP}_{\text{FLOPs}} = 2 \times (2 \times N \times 4 \times D^2) = 16ND^2 \quad (7)$$

$$\text{SA}_{\text{FLOPs}} = 6ND^2 + 2N^2D + 2N^2D + 2ND^2 = 4N^2D + 8ND^2 \quad (8)$$

$$\text{Transformer}_{\text{FLOPs}} = L \times (\text{SA}_{\text{FLOPs}} + \text{MLP}_{\text{FLOPs}}) = L \times (4N^2D + 8ND^2 + 16ND^2) \quad (9)$$

$$= L \times (4N^2D + 24ND^2) = 4LND(N + 6D) \quad (10)$$

C.2. Foundry and Foundry-Gate modules

Dynamic Supertoken Optimization [22].

$$\text{DSO}_{\text{FLOPs}} = (2SD^2 + 4ND^2) + 2SND + 2nhShdD + 2SD^2 \quad (11)$$

$$= 4SD^2 + 4ND^2 + 4SND = 4D^2(S + N) + 4SND \quad (12)$$

Cross-Attention-guided Upsampling [22].

$$\text{CAU}_{\text{FLOPs}} = nhSN + 2nhSN + nhSN + 2nhSNhd + \text{MLP}_{\text{FLOPs}} \quad (13)$$

$$= 4nhSN + 2SND + 16ND^2 = 2SN(2nh + D) + 16ND^2 \quad (14)$$

Gate.

$$\text{Gate}_{\text{FLOPs}} = 2NDd + 2Ndd = 4NDd \quad (15)$$

C.3. Foundry and Foundry-Gate architectures

$$\text{Foundry}_{\text{FLOPs}} = \text{Transformer}_{\text{FLOPs}}(N = S) + \text{DSO}_{\text{FLOPs}} + \text{CAU}_{\text{FLOPs}} \quad (16)$$

$$= [4LSD(S + 6D)] + (4SD^2 + 4ND^2 + 4SND) + (4nhSN + 2SND + 16ND^2) \quad (17)$$

$$= 4SD^2 + 20ND^2 + 6SND + 4nhSN + 4LSD(S + 6D) \quad (18)$$

$$= 4D^2(S + 5N) + 2SN(3D + 2nh) + 4LSD(S + 6D) \quad (19)$$

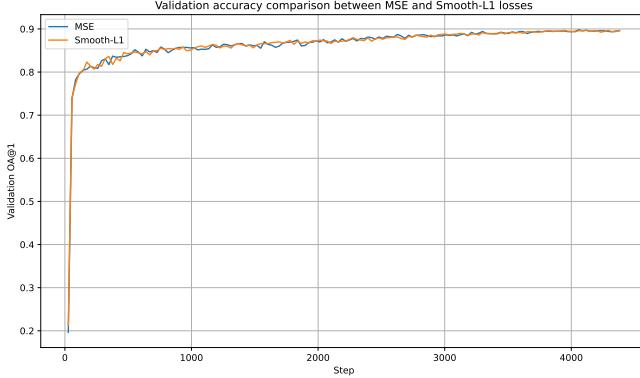


Figure 4. **Comparison on distillation loss.** We use Foundry with a loss update from Smooth-L1 to MSE. To show differences, we show validation top-1 accuracy curves during fine-tuning on ShapeNet55 after distillation using the specified loss.

Let $R = \lfloor rN \rfloor$, $r \in [0, 1]$ be the ratio of fused tokens by the gate and $U = N - R$ the number of unfused tokens.

$$\text{Foundry-Gate}_{\text{FLOPs}}(R) = \text{Gate}_{\text{FLOPs}} + \text{DSO}_{\text{FLOPs}}(N = R) + \text{Transformer}_{\text{FLOPs}}(N = S + U) + \text{CAU}_{\text{FLOPs}}(N = R) \quad (20)$$

$$= (4NDd) + (4SD^2 + 4RD^2 + 4SRD) + [4LD(S + U)(S + U + 6D)] + \\ (4nhSR + 2SRD + 16RD^2) \quad (21)$$

$$= 4NDd + 4SD^2 + 20RD^2 + 6SRD + 4nhSR + [4LD(S + U)(S + U + 6D)] \quad (22)$$

$$= 4D^2(S + 5R) + (2SR)(3D + 2nh) + 4NDd + [4LD(S + U)(S + U + 6D)] \quad (23)$$

D. Additional Results

D.1. Additional ablations

Pre-training with supertokens. For completeness, we provide results on pre-training a Transformer with Point-JEPA method directly using supertokens. The DSO/CAU modules are considered as part of the encoder. Table 9 shows linear probing results at the end of the pre-training. We can clearly see that with the same hyper-parameters we set for SuperTokens,

Method	MN40 (OA@1)	OO3D (OA@1)	SONN-PB-T50-RS (OA@1)
Point-JEPA	93.01 ± 0.18	77.39 ± 0.52	83.24 ± 0.66
Point-JEPA + DSO+CAU	78.12	30.63	51.29

Table 9. **Linear probing results of SuperTokens-Point-JEPA.** We only used datasets for classification task. Mean \pm std over 10 runs for baseline.

the differences in performance are very large. We believe that the tasks of compressing information via supertokens and extracting as much information as possible via the self-supervision method lead to a conflict that limits the final capacity of the pre-trained model with this architecture. This demonstrates that using a pre-trained model on all tokens and then performing a distillation step remains the simplest and, above all, the best solution.

MSE vs Smooth-L1. Figure 4 illustrates the impact of the distillation loss choice during model fine-tunings. During distillations and fine-tunings, the backbone is kept frozen. We observe a similar trend with similar values. No loss is recommended.

D.2. Detailed fine-tuning results

Foundry. Table 10 describes detailed results when fine-tuning the distilled backbone using different numbers of supertokens. The maximum number of epochs for classification and part segmentation tasks are 150 and 300 respectively. In addition, we traditionally unfreeze the backbone respectively at 100 and 0 epoch.

# supertokens	Frozen	FT	Unfreeze	Distillation	Avg	MN40	OO3D	SONN-	SN55	SNP			
	student	epoch		loss		OBJ-BG	OBJ-ONLY	PB-T50-RS	mIoU _C	mIoU _I			
baseline ($c = 64$)	-	100		-	-	93.02 \pm 0.14	82.25 \pm 0.37	91.84 \pm 0.47	88.38 \pm 0.41	86.05 \pm 0.42	90.54 \pm 0.14	-	-
baseline ($c = 64$)	-	0		-	-	-	-	-	-	-	-	83.91 \pm 0.25	85.73 \pm 0.15
baseline ($c = 64$)	-	-		-	87.72	93.02 \pm 0.14	82.25 \pm 0.37	91.84 \pm 0.47	88.38 \pm 0.41	86.05 \pm 0.42	90.54 \pm 0.14	83.91 \pm 0.25	85.73 \pm 0.15
<i>Random sampling - inference from baseline</i>													
$c = 16$	-	-		-	75.20	90.57 \pm 0.41	65.19 \pm 0.81	78.95 \pm 1.19	72.70 \pm 1.14	69.74 \pm 0.91	87.80 \pm 0.22	66.96 \pm 0.71	69.71 \pm 0.16
$c = 8$	-	-		-	63.45	83.72 \pm 0.60	41.20 \pm 1.35	67.40 \pm 1.60	62.67 \pm 1.47	55.78 \pm 0.83	80.72 \pm 0.36	56.47 \pm 0.62	59.60 \pm 0.19
$c = 4$	-	-		-	47.46	65.45 \pm 0.80	19.74 \pm 1.08	50.79 \pm 0.69	45.30 \pm 1.41	39.39 \pm 0.72	62.51 \pm 0.38	46.35 \pm 0.46	50.11 \pm 0.23
$c = 2$	-	-		-	32.10	39.32 \pm 0.55	8.03 \pm 0.52	35.51 \pm 1.52	32.19 \pm 1.78	27.07 \pm 0.62	36.00 \pm 0.29	37.11 \pm 0.27	41.56 \pm 0.10
$c = 1$	-	-		-	21.46	18.33 \pm 0.66	0.382 \pm 0.32	23.22 \pm 1.55	21.31 \pm 1.35	19.90 \pm 0.68	15.54 \pm 0.23	33.70 \pm 0.30	35.89 \pm 0.16
<i>Updating group size to reduce number of generated tokens - inference from baseline</i>													
$c = 16$	-	-		-	41.66	63.45 \pm 0.47	13.29 \pm 0.42	11.76 \pm 0.16	22.24 \pm 0.29	12.74 \pm 0.17	59.99 \pm 0.19	72.98 \pm 0.21	76.84 \pm 0.08
$c = 8$	-	-		-	26.36	20.57 \pm 0.19	0.362 \pm 0.19	09.29 \pm 0.00	17.33 \pm 0.36	09.37 \pm 0.00	23.12 \pm 0.18	61.36 \pm 0.35	66.24 \pm 0.13
$c = 4$	-	-		-	20.23	10.32 \pm 0.25	0.066 \pm 0.14	09.29 \pm 0.00	15.44 \pm 0.34	09.37 \pm 0.00	12.65 \pm 0.21	50.31 \pm 0.36	53.80 \pm 0.13
$c = 2$	-	-		-	15.12	04.79 \pm 0.15	0.028 \pm 0.10	09.29 \pm 0.00	12.67 \pm 0.36	09.37 \pm 0.00	05.53 \pm 0.11	37.08 \pm 0.09	41.95 \pm 0.08
$c = 1$	-	-		-	15.78	03.88 \pm 0.19	0.047 \pm 0.13	09.24 \pm 0.63	15.85 \pm 0.82	09.37 \pm 0.00	04.25 \pm 0.09	40.30 \pm 0.44	42.90 \pm 0.34
<i>ToMe [4] - inference from baseline (no additional training needed)</i>													
$s = 16$	-	-		-	73.80	89.10 \pm 0.23	50.51 \pm 0.39	82.19 \pm 0.56	84.22 \pm 0.62	73.87 \pm 0.50	84.89 \pm 0.24	63.09 \pm 0.40	62.51 \pm 0.08
$s = 8$	-	-		-	66.79	84.02 \pm 0.38	33.54 \pm 0.78	76.87 \pm 1.30	79.88 \pm 0.96	67.32 \pm 0.62	79.68 \pm 0.26	56.81 \pm 0.43	56.23 \pm 0.14
$s = 4$	-	-		-	59.88	74.92 \pm 0.65	20.91 \pm 0.59	71.57 \pm 1.05	74.29 \pm 0.54	61.52 \pm 0.49	72.05 \pm 0.30	51.59 \pm 0.52	52.19 \pm 0.12
$s = 2$	-	-		-	53.19	62.76 \pm 0.44	11.21 \pm 0.36	67.76 \pm 1.03	68.49 \pm 0.84	56.30 \pm 0.62	61.66 \pm 0.30	47.47 \pm 0.45	49.88 \pm 0.12
$s = 1$	-	-		-	46.15	48.35 \pm 0.31	07.20 \pm 0.20	61.20 \pm 0.89	62.87 \pm 0.95	51.69 \pm 0.46	42.15 \pm 0.18	45.36 \pm 0.19	50.37 \pm 0.09
<i>ToMe [4] - Trained using FMD (Ours) + finetuning</i>													
$s = 16$	\times	-		0.0658	86.52	93.07	80.95	90.02	87.44	83.66	89.84	82.41	84.76
$s = 1$	\times	-		0.1509	85.87	92.14	80.08	89.85	86.40	83.00	89.84	81.14	84.50
<i>PiToMe [35] - Trained using FMD (Ours) + finetuning</i>													
$s = 16$	\times	-		0.0643	86.50	92.83	81.97	89.33	87.95	83.66	89.88	81.77	84.65
$s = 1$	\times	-		0.1508	86.14	92.55	80.95	90.19	86.23	82.89	89.76	81.84	84.74
<i>PatchMerger [29] - Trained using FMD (Ours) + finetuning</i>													
$s = 16$	\times	-		0.0326	87.59	92.83	82.76	91.39	88.99	85.53	90.11	83.59	85.55
$s = 1$	\times	-		0.2811	87.14	92.18	82.13	90.71	88.12	85.05	89.71	83.64	85.54
<i>Foundry (Ours)</i>													
$s = 16$	\checkmark	300	0.1006	-	-	-	-	-	-	-	81.55	84.59	
$s = 16$	\checkmark	150	0.1006	-	91.09	76.61	84.68	85.54	80.81	89.68	-	-	
$s = 16$	\checkmark	100	0.1006	-	91.73	77.48	85.20	84.17	80.29	89.67	-	-	
$s = 16$	\checkmark	0	0.1006	-	91.41	77.80	84.34	85.20	78.70	89.32	82.21	85.05	
$s = 16$	\checkmark	-	0.1006	84.33	91.41 \pm 0.32	77.30 \pm 0.61	84.74 \pm 0.43	84.97 \pm 0.72	79.93 \pm 1.10	89.56 \pm 0.21	81.88 \pm 0.47	84.82 \pm 0.32	
$s = 16$	\times	300	0.0736	-	-	-	-	-	-	-	81.90	84.87	
$s = 16$	\times	150	0.0736	-	91.69	77.72	86.06	86.75	80.43	89.88	-	-	
$s = 16$	\times	100	0.0736	-	92.02	78.03	86.75	85.89	81.23	89.95	-	-	
$s = 16$	\times	0	0.0736	-	91.53	77.64	85.89	86.23	79.42	89.78	81.85	84.77	
$s = 16$	\times	-	0.0736	84.87	91.75 \pm 0.25	77.80 \pm 0.21	86.23 \pm 0.46	86.29 \pm 0.43	80.36 \pm 0.90	89.87 \pm 0.09	81.87 \pm 0.04	84.82 \pm 0.07	
$s = 16$ (ViT-T)	\times	300	0.1040	-	-	-	-	-	-	-	81.85	84.49	
$s = 16$ (ViT-T)	\times	150	0.1040	-	91.69	75.75	84.34	83.99	79.49	89.04	-	-	
$s = 16$ (ViT-T)	\times	100	0.1040	-	91.77	76.22	83.30	85.89	79.46	88.56	-	-	
$s = 16$ (ViT-T)	\times	0	0.1040	-	91.00	75.98	82.62	84.51	79.18	88.41	81.51	84.63	
$s = 16$ (ViT-T)	\times	-	0.1040	83.75	91.49 \pm 0.42	75.98 \pm 0.24	83.42 \pm 0.87	84.80 \pm 0.98	79.38 \pm 0.17	88.67 \pm 0.33	81.68 \pm 0.24	84.56 \pm 0.10	
$s = 8$	\checkmark	300	0.0993	-	-	-	-	-	-	-	82.08	84.53	
$s = 8$	\checkmark	150	0.0993	-	91.69	76.69	85.37	84.51	81.05	89.48	-	-	
$s = 8$	\checkmark	100	0.0993	-	91.82	76.69	86.23	84.68	80.64	89.56	-	-	
$s = 8$	\checkmark	0	0.0993	-	91.61	76.06	85.20	86.23	79.15	89.23	81.60	84.60	
$s = 8$	\checkmark	-	0.0993	84.38	91.71 \pm 0.10	76.48 \pm 0.36	85.60 \pm 0.55	85.14 \pm 0.95	80.28 \pm 1.00	89.42 \pm 0.17	81.84 \pm 0.34	84.57 \pm 0.05	
$s = 8$	\times	300	0.0749	-	-	-	-	-	-	-	82.09	84.86	
$s = 8$	\times	150	0.0749	-	91.49	78.11	86.23	86.57	80.46	89.67	-	-	
$s = 8$	\times	100	0.0749	-	91.86	77.17	85.89	84.85	81.19	89.81	-	-	
$s = 8$	\times	0	0.0749	-	91.53	76.69	86.57	86.75	79.74	89.34	81.81	84.84	
$s = 8$	\times	-	0.0749	84.76	91.63 \pm 0.20	77.32 \pm 0.72	86.23 \pm 0.34	86.06 \pm 1.05	80.46 \pm 0.73	89.61 \pm 0.24	81.95 \pm 0.19	84.85 \pm 0.01	
$s = 4$	\checkmark	300	0.0986	-	-	-	-	-	-	-	81.97	84.72	
$s = 4$	\checkmark	150	0.0986	-	91.09	77.24	84.51	84.85	79.53	89.41	-	-	
$s = 4$	\checkmark	100	0.0986	-	91.33	76.93	85.03	86.92	79.77	89.43	-	-	
$s = 4$	\checkmark	0	0.0986	-	91.33	76.69	83.82	86.06	79.53	88.95	81.71	84.61	
$s = 4$	\checkmark	-	0.0986	84.25	91.25 \pm 0.14	76.96 \pm 0.28	84.45 \pm 0.60	85.94 \pm 1.04	79.61 \pm 0.14	89.26 \pm 0.27	81.84 \pm 0.18	84.67 \pm 0.08	
$s = 4$	\times	300	0.0788	-	-	-	-	-	-	-	81.57	84.76	
$s = 4$	\times	150	0.0788	-	91.45	76.14	86.23	85.89	79.67	89.81	-	-	
$s = 4$	\times	100	0.0788	-	91.41	77.87	87.09	86.40	80.95	89.74	-	-	
$s = 4$	\times	0	0.0788	-	91.57	76.06	84.68	85.37	81.47	89.68	81.99	84.86	
$s = 4$	\times	-	0.0788	84.64	91.48 \pm 0.08	76.69 \pm 1.02	86.00 \pm 1.22	85.89 \pm 0.52	80.70 \pm 0.93	89.74 \pm 0.06	81.78 \pm 0.30	84.81 \pm 0.07	
$s = 2$	\checkmark	300	0.0979	-	-	-	-	-	-	-	81.58	84.47	
$s = 2$	\checkmark	150	0.0979	-	92.06	77.09	83.82	83.30	79.84	89.26	-	-	
$s = 2$	\checkmark	100	0.0979	-	91.00	76.77	84.68	85.37	80.12	89.50	-	-	

Table 10 – continued from previous page

$s = 2$	\times	300	0.0795	-	-	-	-	-	-	-	-	82.13	84.50	
$s = 2$	\times	150	0.0795	-	91.65	78.66	85.37	86.40	80.57	89.68	-	-	-	
$s = 2$	\times	100	0.0795	-	92.02	78.74	85.71	86.40	81.16	89.67	-	-	-	
$s = 2$	\times	0	0.0795	-	91.61	77.56	86.40	85.54	80.60	89.62	81.84	84.44	-	
$s = 2$	\times	-	0.0795	84.86	91.76 ± 0.22	78.32 ± 0.66	85.83 ± 0.53	86.12 ± 0.50	80.78 ± 0.33	89.66 ± 0.04	81.98 ± 0.21	84.47 ± 0.04	-	-
$s = 1$	\checkmark	300	0.0967	-	-	-	-	-	-	-	-	81.51	84.61	
$s = 1$	\checkmark	150	0.0967	-	91.94	76.93	84.68	85.37	79.77	89.14	-	-	-	
$s = 1$	\checkmark	100	0.0967	-	91.49	76.69	85.37	84.17	79.53	89.37	-	-	-	
$s = 1$	\checkmark	0	0.0967	-	91.37	77.95	84.34	85.54	79.11	89.35	81.40	84.47	-	-
$s = 1$	\checkmark	-	0.0967	84.17	91.60 ± 0.30	77.19 ± 0.67	84.80 ± 0.53	85.03 ± 0.75	79.47 ± 0.33	89.29 ± 0.13	81.46 ± 0.08	84.54 ± 0.10	-	-
$s = 1$	\times	300	0.0792	-	-	-	-	-	-	-	-	81.66	84.56	
$s = 1$	\times	150	0.0792	-	91.86	77.64	86.06	85.20	81.09	89.67	-	-	-	
$s = 1$	\times	100	0.0792	-	91.57	77.72	83.65	87.09	80.40	89.69	-	-	-	
$s = 1$	\times	0	0.0792	-	91.73	77.80	83.99	85.89	80.46	89.58	81.86	84.43	-	-
$s = 1$	\times	-	0.0792	84.58	91.72 ± 0.14	77.72 ± 0.08	84.57 ± 1.30	86.06 ± 0.96	80.65 ± 0.38	89.65 ± 0.06	81.76 ± 0.14	84.50 ± 0.09	-	-

Table 10. Detailed finetuning results of Foundry. All presented results are finetunings from a frozen or unfrozen backbone during distillation (no distillation stage for the baseline). For the classification task on ModelNet40 (MN40), OmniObject3D (OO3D), the three splits of ScanObjectNN (SONN- $\{\text{PB-T50-RS}, \text{OBJ-BG}, \text{OBJ-ONLY}\}$) and ShapeNet55 (SN55), the used metric is the top-1 accuracy on the validation set. For part segmentation with ShapeNetPart (SNP), we show for both category and instance mIoUs. Mean \pm Std Dev over 10 runs for the baseline. "FT Unfreeze Epoch" denotes for the epoch when we completely unfreeze the ViT backbone during the finetuning stage ('-' in this column means aggregation over the results above). In columns with results, '-' stands for no running was performed. ToMe [4] and PiToMe [35] halve the number of tokens at each layer up to the specified number of tokens (supertokens, to simplify the table columns). For distilled ToMe, PiToMe and PatchMerger [29], the distilled encoder is unfrozen at 100 epochs for classification datasets and at 0 epochs for part segmentation.

Foundry-Gate. Similarly, we have Tab. 11 which details results when using the gate module. For simplifying the results visualization, Fig. 5 plots the metric values of fine-tuned models for each dataset at each regularization value.

λ_{gate}	Frozen student	FT Unfreeze epoch	Distillation loss	Avg	MN40	OO3D	SONN- OBJ-BG	OBJ-ONLY	PB-T50-RS	SN55	SNP	mIoU _C	mIoU _I	
	-	-	-	87.72	93.02 ± 0.14	82.25 ± 0.37	91.84 ± 0.47	88.38 ± 0.41	86.05 ± 0.42	90.54 ± 0.14	83.91 ± 0.25	85.73 ± 0.15	-	-
<i>Foundry-Gate (Ours)</i>														
0	\checkmark	-	0.0191	-	-	-	-	-	-	-	-	-	-	
0	\times	-	0.0152	-	-	-	-	-	-	-	-	-	-	
10^{-15}	\checkmark	300	0.0191	-	-	-	-	-	-	-	82.64	85.24	-	
10^{-15}	\checkmark	150	0.0191	-	93.03	79.37	89.67	87.95	83.10	90.30	-	-	-	
10^{-15}	\checkmark	100	0.0191	-	93.48	81.97	92.77	88.12	86.02	90.75	-	-	-	
10^{-15}	\checkmark	0	0.0191	-	92.22	83.23	90.53	88.30	84.84	90.31	83.77	85.63	-	
10^{-15}	\checkmark	-	0.0191	87.16	92.91 ± 0.64	81.52 ± 1.97	90.99 ± 1.60	88.12 ± 0.17	84.65 ± 1.47	90.45 ± 0.26	83.21 ± 0.79	85.44 ± 0.27	-	-
10^{-15}	\times	300	0.0142	-	-	-	-	-	-	-	82.76	85.06	-	
10^{-15}	\times	150	0.0142	-	93.56	80.24	88.30	87.95	82.86	90.50	-	-	-	
10^{-15}	\times	100	0.0142	-	93.19	83.23	92.25	89.67	86.12	90.55	-	-	-	
10^{-15}	\times	0	0.0142	-	92.14	82.36	90.88	89.67	85.50	90.38	83.93	85.62	-	-
10^{-15}	\times	-	0.0142	87.31	92.96 ± 0.74	81.94 ± 1.54	90.48 ± 2.01	89.10 ± 0.99	84.83 ± 1.73	90.48 ± 0.09	83.35 ± 0.83	85.34 ± 0.39	-	-
10^{-14}	\checkmark	300	0.0191	-	-	-	-	-	-	-	82.78	85.11	-	
10^{-14}	\checkmark	150	0.0191	-	93.31	79.61	90.71	88.47	83.07	90.36	-	-	-	
10^{-14}	\checkmark	100	0.0191	-	92.87	81.73	91.22	87.78	86.36	90.58	-	-	-	
10^{-14}	\checkmark	0	0.0191	-	92.59	82.28	90.36	87.95	85.60	90.24	83.66	85.42	-	-
10^{-14}	\checkmark	-	0.0191	87.11	92.92 ± 0.37	81.21 ± 1.41	90.76 ± 0.43	88.07 ± 0.36	85.01 ± 1.73	90.39 ± 0.17	83.22 ± 0.62	85.26 ± 0.21	-	-
10^{-14}	\times	300	0.0141	-	-	-	-	-	-	-	82.55	85.12	-	
10^{-14}	\times	150	0.0141	-	93.15	80.31	90.02	88.47	83.69	90.22	-	-	-	
10^{-14}	\times	100	0.0141	-	93.15	82.60	90.71	89.16	85.91	90.52	-	-	-	
10^{-14}	\times	0	0.0141	-	92.30	82.68	91.05	87.61	84.39	90.21	84.06	85.60	-	-
10^{-14}	\times	-	0.0141	87.17	92.87 ± 0.49	81.86 ± 1.34	90.59 ± 0.53	88.41 ± 0.78	84.66 ± 1.14	90.32 ± 0.18	83.30 ± 1.07	85.36 ± 0.34	-	-
10^{-13}	\checkmark	300	0.0193	-	-	-	-	-	-	-	82.68	85.11	-	
10^{-13}	\checkmark	150	0.0193	-	92.95	80.08	89.67	87.09	83.21	90.43	-	-	-	
10^{-13}	\checkmark	100	0.0193	-	92.74	82.36	91.22	87.78	86.09	90.53	-	-	-	
10^{-13}	\checkmark	0	0.0193	-	92.018	81.65	91.05	87.09	85.36	90.08	83.88	85.50	-	-
10^{-13}	\checkmark	-	0.0193	86.96	92.57 ± 0.49	81.36 ± 1.17	90.65 ± 0.85	87.32 ± 0.40	84.88 ± 1.50	90.35 ± 0.23	83.28 ± 0.85	85.30 ± 0.28	-	-
10^{-13}	\times	300	0.0141	-	-	-	-	-	-	-	82.85	85.07	-	
10^{-13}	\times	150	0.0141	-	93.23	79.84	90.19	89.16	83.03	90.15	-	-	-	
10^{-13}	\times	100	0.0141	-	93.07	82.68	90.02	88.12	85.95	90.47	-	-	-	
10^{-13}	\times	0	0.0141	-	92.50	82.20	90.36	89.50	84.80	90.30	83.79	85.61	-	-
10^{-13}	\times	-	0.0141	87.15	92.94 ± 0.38	81.57 ± 1.52	90.19 ± 0.17	88.93 ± 0.72	84.59 ± 1.47	90.31 ± 0.16	83.32 ± 0.66	85.34 ± 0.39	-	-
10^{-12}	\checkmark	300	0.0218	-	93.19	79.61	89.33	87.95	82.17	89.71	-	82.76	85.26	-
10^{-12}	\checkmark	150	0.0218	-	-	-	-	-	-	-	-	-	-	-

Continued on next page

Table 11 – continued from previous page

10^{-12}	✓	100	0.0218	-	93.15	81.89	91.05	88.47	85.91	89.78	-	-
10^{-12}	✓	0	0.0218	-	91.69	81.89	89.85	88.64	74.98	89.34	83.97	85.76
10^{-12}	✓	-	0.0218	86.47	92.68 ± 0.85	81.13 ± 1.32	90.07 ± 0.88	88.35 ± 0.36	81.02 ± 5.55	89.61 ± 0.24	83.37 ± 0.86	85.51 ± 0.35
10^{-12}	✗	300	0.0157	-	-	-	-	-	-	-	82.64	84.99
10^{-12}	✗	150	0.0157	-	92.99	80.00	89.16	88.47	83.14	88.60	-	-
10^{-12}	✗	100	0.0157	-	93.15	82.36	91.74	88.81	84.84	90.07	-	-
10^{-12}	✗	0	0.0157	-	91.65	81.50	90.02	88.47	74.57	89.74	83.70	85.69
10^{-12}	✗	-	0.0157	86.45	92.60 ± 0.82	81.29 ± 1.20	90.30 ± 1.31	88.58 ± 0.20	80.85 ± 5.50	89.47 ± 0.77	83.17 ± 0.75	85.34 ± 0.49
$10^{-11.75}$	✓	300	0.024	-	-	-	-	-	-	-	82.69	85.23
$10^{-11.75}$	✓	150	0.024	-	92.95	78.90	88.64	87.78	82.41	87.16	-	-
$10^{-11.75}$	✓	100	0.024	-	92.46	81.02	92.25	87.09	85.25	87.96	-	-
$10^{-11.75}$	✓	0	0.024	-	89.06	82.05	90.02	88.64	74.36	87.84	83.82	85.58
$10^{-11.75}$	✓	-	0.024	85.91	91.49 ± 2.12	80.66 ± 1.61	90.30 ± 1.82	87.84 ± 0.78	80.67 ± 5.65	87.66 ± 0.43	83.25 ± 0.80	85.40 ± 0.25
$10^{-11.75}$	✗	300	0.0171	-	-	-	-	-	-	-	82.39	84.97
$10^{-11.75}$	✗	150	0.0171	-	92.71	79.92	89.85	88.81	82.03	88.45	-	-
$10^{-11.75}$	✗	100	0.0171	-	92.99	82.68	91.22	88.12	85.53	88.25	-	-
$10^{-11.75}$	✗	0	0.0171	-	88.94	81.26	90.19	88.64	77.65	88.91	83.95	85.78
$10^{-11.75}$	✗	-	0.0171	86.33	91.55 ± 2.26	81.29 ± 1.38	90.42 ± 0.72	88.53 ± 0.36	81.74 ± 3.95	88.54 ± 0.34	83.17 ± 1.11	85.37 ± 0.57
$10^{-11.5}$	✓	300	0.029	-	-	-	-	-	-	-	82.08	84.43
$10^{-11.5}$	✓	150	0.029	-	90.76	79.29	88.30	86.06	82.30	88.48	-	-
$10^{-11.5}$	✓	100	0.029	-	92.71	79.92	90.53	87.78	84.49	87.99	-	-
$10^{-11.5}$	✓	0	0.029	-	90.40	77.87	89.50	87.61	79.98	88.13	83.80	85.66
$10^{-11.50}$	✓	-	0.029	85.67	91.29 ± 1.24	79.03 ± 1.05	89.44 ± 1.12	87.15 ± 0.95	82.26 ± 2.26	88.20 ± 0.25	82.94 ± 1.22	85.05 ± 0.87
$10^{-11.5}$	✗	300	0.0198	-	-	-	-	-	-	-	82.00	84.66
$10^{-11.5}$	✗	150	0.0198	-	91.69	75.98	89.67	88.64	80.22	88.71	-	-
$10^{-11.5}$	✗	100	0.0198	-	92.02	81.50	91.91	89.33	78.14	88.46	-	-
$10^{-11.5}$	✗	0	0.0198	-	89.67	71.42	87.44	86.75	78.90	89.20	83.60	85.63
$10^{-11.50}$	✗	-	0.0198	85.15	91.13 ± 1.27	76.30 ± 5.05	89.67 ± 2.24	88.24 ± 1.34	79.09 ± 1.05	88.79 ± 0.38	82.80 ± 1.13	85.15 ± 0.68
$10^{-11.25}$	✓	300	0.039	-	-	-	-	-	-	-	82.02	84.75
$10^{-11.25}$	✓	150	0.039	-	89.34	78.11	89.16	87.44	81.40	88.36	-	-
$10^{-11.25}$	✓	100	0.039	-	88.29	81.50	88.47	87.78	79.91	88.75	-	-
$10^{-11.25}$	✓	0	0.039	-	90.60	74.02	86.23	87.44	79.91	88.41	82.29	85.00
$10^{-11.25}$	✓	-	0.039	84.84	89.41 ± 1.16	77.87 ± 3.75	87.95 ± 1.53	87.55 ± 0.20	80.41 ± 0.86	88.51 ± 0.21	82.16 ± 0.19	84.87 ± 0.18
$10^{-11.25}$	✗	300	0.0264	-	-	-	-	-	-	-	81.87	84.65
$10^{-11.25}$	✗	150	0.0264	-	89.63	78.11	89.85	88.12	78.24	88.55	-	-
$10^{-11.25}$	✗	100	0.0264	-	89.51	78.35	90.71	86.06	78.38	88.68	-	-
$10^{-11.25}$	✗	0	0.0264	-	90.64	74.49	84.51	85.37	80.64	88.72	82.85	85.11
$10^{-11.25}$	✗	-	0.0264	84.59	89.92 ± 0.62	76.98 ± 2.16	88.35 ± 3.36	86.52 ± 1.43	79.09 ± 1.34	88.65 ± 0.09	82.36 ± 0.69	84.88 ± 0.33
10^{-11}	✓	300	0.0619	-	-	-	-	-	-	-	81.99	84.70
10^{-11}	✓	150	0.0619	-	91.57	77.40	87.95	86.92	77.72	88.61	-	-
10^{-11}	✓	100	0.0619	-	90.11	73.46	88.12	87.95	76.86	88.27	-	-
10^{-11}	✓	0	0.0619	-	91.29	76.77	84.85	87.95	79.18	88.90	82.28	84.79
10^{-11}	✓	-	0.0619	84.36	90.99 ± 0.77	75.88 ± 2.11	86.98 ± 1.84	87.61 ± 0.60	77.92 ± 1.17	88.59 ± 0.31	82.14 ± 0.21	84.75 ± 0.06
10^{-11}	✗	300	0.0419	-	-	-	-	-	-	-	81.63	84.47
10^{-11}	✗	150	0.0419	-	90.60	76.77	87.26	83.65	80.74	88.84	-	-
10^{-11}	✗	100	0.0419	-	90.76	77.40	83.99	86.57	80.01	88.91	-	-
10^{-11}	✗	0	0.0419	-	91.49	76.54	86.40	86.57	80.15	88.90	81.84	84.70
10^{-11}	✗	-	0.0419	84.36	90.95 ± 0.47	76.90 ± 0.45	85.89 ± 1.70	85.60 ± 1.69	80.30 ± 0.39	88.88 ± 0.04	81.74 ± 0.15	84.58 ± 0.16
$10^{-10.75}$	✓	300	0.0955	-	-	-	-	-	-	-	81.65	84.53
$10^{-10.75}$	✓	150	0.0955	-	91.00	75.04	85.54	86.23	78.31	89.02	-	-
$10^{-10.75}$	✓	100	0.0955	-	91.13	78.82	88.64	86.92	77.41	89.09	-	-
$10^{-10.75}$	✓	0	0.0955	-	91.65	77.80	85.89	87.61	80.01	89.03	81.67	84.89
$10^{-10.75}$	✓	-	0.0955	84.51	91.26 ± 0.34	77.22 ± 1.95	86.69 ± 1.70	86.92 ± 0.69	78.58 ± 1.32	89.04 ± 0.04	81.66 ± 0.02	84.71 ± 0.25
$10^{-10.75}$	✗	300	0.0672	-	-	-	-	-	-	-	81.73	84.77
$10^{-10.75}$	✗	150	0.0672	-	91.17	76.69	85.71	86.57	81.02	89.68	-	-
$10^{-10.75}$	✗	100	0.0672	-	92.06	78.74	86.75	88.12	81.26	90.17	-	-
$10^{-10.75}$	✗	0	0.0672	-	91.29	77.64	87.95	86.92	80.08	89.07	81.97	84.95
$10^{-10.75}$	✗	-	0.0672	85.04	91.50 ± 0.48	77.69 ± 1.02	86.80 ± 1.12	87.21 ± 0.81	80.79 ± 0.62	89.64 ± 0.55	81.85 ± 0.17	84.86 ± 0.13
$10^{-10.5}$	✓	300	0.1013	-	-	-	-	-	-	-	81.73	84.52
$10^{-10.5}$	✓	150	0.1013	-	91.57	77.09	84.51	84.34	79.01	88.95	-	-
$10^{-10.5}$	✓	100	0.1013	-	91.17	78.11	84.68	85.37	78.73	89.35	-	-
$10^{-10.5}$	✓	0	0.1013	-	90.92	77.80	85.54	84.51	70.37	89.26	82.13	84.87
$10^{-10.50}$	✓	-	0.1013	83.80	91.22 ± 0.33	77.66 ± 0.52	84.91 ± 0.55	84.74 ± 0.55	76.04 ± 4.91	89.19 ± 0.21	81.93 ± 0.29	84.70 ± 0.25
$10^{-10.5}$	✗	300	0.0753	-	-	-	-	-	-	-	82.27	85.00
$10^{-10.5}$	✗	150	0.0753	-	91.21	76.61	87.95	86.75	80.33	89.88	-	-
$10^{-10.5}$	✗	100	0.0753	-	91.53	77.56	87.09	87.61	80.88	89.87	-	-
$10^{-10.5}$	✗	0	0.0753	-	90.72	77.09	86.40	87.09	78.45	89.16	81.88	84.79
$10^{-10.50}$	✗	-	0.0753	84.88	91.15 ± 0.41	77.09 ± 0.47	87.15 ± 0.78	87.15 ± 0.43	79.89 ± 1.27	89.64 ± 0.41	82.08 ± 0.27	84.89 ± 0.15
$10^{-10.25}$	✓	300	0.0996	-	-	-	-	-	-	-	81.80	84.53
$10^{-10.25}$	✓	150	0.0996	-	91.00	76.30	83.48	84.68	79.35	89.31	-	-
$10^{-10.25}$	✓	100	0.0996	-	90.96	77.80	84.51	84.85	79.15	89.29	-	-

Continued on next page

Table 11 – continued from previous page

$10^{-10.25}$	✓	0	0.0996	-	91.13	77.40	83.82	84.51	79.04	89.08	82.36	84.85
$10^{-10.25}$	✓	-	0.0996	84.00	91.03 \pm 0.08	77.17 \pm 0.78	83.94 \pm 0.53	84.68 \pm 0.17	79.18 \pm 0.16	89.22 \pm 0.13	82.08 \pm 0.39	84.69 \pm 0.23
$10^{-10.25}$	✗	300	0.0754	-	-	-	-	-	-	-	82.37	85.01
$10^{-10.25}$	✗	150	0.0754	-	91.90	76.93	85.37	86.75	80.57	89.89	-	-
$10^{-10.25}$	✗	100	0.0754	-	91.53	77.40	85.20	85.54	80.88	89.98	-	-
$10^{-10.25}$	✗	0	0.0754	-	91.37	76.46	86.06	86.06	80.95	89.79	82.49	84.91
$10^{-10.25}$	✗	-	0.0754	84.78	91.60 \pm 0.27	76.93 \pm 0.47	85.54 \pm 0.46	86.12 \pm 0.60	80.80 \pm 0.20	89.89 \pm 0.10	82.43 \pm 0.08	84.96 \pm 0.08
10^{-10}	✓	300	0.0981	-	-	-	-	-	-	-	82.04	84.68
10^{-10}	✓	150	0.0981	-	91.41	76.69	83.65	85.37	79.49	89.41	-	-
10^{-10}	✓	100	0.0981	-	91.33	76.46	83.99	84.85	79.25	89.58	-	-
10^{-10}	✓	0	0.0981	-	90.56	75.75	83.48	84.85	78.73	88.97	82.22	84.67
10^{-10}	✓	-	0.0981	83.93	91.10 \pm 0.47	76.30 \pm 0.49	83.71 \pm 0.26	85.03 \pm 0.30	79.16 \pm 0.39	89.32 \pm 0.31	82.13 \pm 0.13	84.67 \pm 0.01
10^{-10}	✗	300	0.0763	-	-	-	-	-	-	-	82.05	84.70
10^{-10}	✗	150	0.0763	-	91.41	77.64	86.75	85.03	80.43	89.68	-	-
10^{-10}	✗	100	0.0763	-	91.13	77.87	85.20	86.75	80.71	89.95	-	-
10^{-10}	✗	0	0.0763	-	91.45	76.77	85.03	86.57	78.66	89.46	82.18	84.67
10^{-10}	✗	-	0.0763	84.62	91.33 \pm 0.18	77.43 \pm 0.58	85.66 \pm 0.95	86.12 \pm 0.95	79.93 \pm 1.11	89.70 \pm 0.25	82.12 \pm 0.09	84.69 \pm 0.02
10^{-9}	✓	300	0.0984	-	-	-	-	-	-	-	82.20	84.56
10^{-9}	✓	150	0.0984	-	91.21	77.09	85.54	85.54	79.98	89.26	-	-
10^{-9}	✓	100	0.0984	-	91.17	77.32	85.37	84.85	78.80	89.43	-	-
10^{-9}	✓	0	0.0984	-	91.29	77.32	85.54	84.17	79.49	89.00	82.02	84.71
10^{-9}	✓	-	0.0984	84.27	91.22 \pm 0.06	77.24 \pm 0.14	85.48 \pm 0.10	84.85 \pm 0.69	79.42 \pm 0.59	89.23 \pm 0.22	82.11 \pm 0.12	84.64 \pm 0.10
10^{-9}	✗	300	0.0774	-	-	-	-	-	-	-	82.23	84.75
10^{-9}	✗	150	0.0774	-	91.29	76.85	85.20	86.57	80.81	89.96	-	-
10^{-9}	✗	100	0.0774	-	91.09	77.32	86.06	86.06	80.53	89.79	-	-
10^{-9}	✗	0	0.0774	-	90.60	77.80	85.71	86.40	80.50	89.64	82.02	84.82
10^{-9}	✗	-	0.0774	84.71	90.99 \pm 0.35	77.32 \pm 0.47	85.66 \pm 0.43	86.35 \pm 0.26	80.62 \pm 0.17	89.80 \pm 0.16	82.13 \pm 0.15	84.78 \pm 0.05
10^{-8}	✓	300	0.0987	-	-	-	-	-	-	-	81.73	84.62
10^{-8}	✓	150	0.0987	-	91.41	78.43	84.34	85.37	79.94	89.53	-	-
10^{-8}	✓	100	0.0987	-	91.13	77.01	83.48	85.54	80.19	89.65	-	-
10^{-8}	✓	0	0.0987	-	91.00	77.64	83.48	85.37	79.04	89.26	81.71	84.81
10^{-8}	✓	-	0.0987	84.21	91.18 \pm 0.21	77.69 \pm 0.71	83.76 \pm 0.50	85.43 \pm 0.10	79.72 \pm 0.60	89.48 \pm 0.20	81.72 \pm 0.01	84.72 \pm 0.13
10^{-8}	✗	300	0.0760	-	-	-	-	-	-	-	82.35	84.79
10^{-8}	✗	150	0.0760	-	91.09	76.22	86.75	85.37	80.46	89.72	-	-
10^{-8}	✗	100	0.0760	-	91.61	77.32	86.06	86.92	80.29	89.83	-	-
10^{-8}	✗	0	0.0760	-	91.45	76.46	86.23	89.16	80.60	89.58	82.26	84.94
10^{-8}	✗	-	0.0760	84.86	91.38 \pm 0.27	76.67 \pm 0.58	86.35 \pm 0.36	87.15 \pm 1.90	80.45 \pm 0.16	89.71 \pm 0.12	82.30 \pm 0.06	84.87 \pm 0.11
10^{-7}	✓	300	0.0999	-	-	-	-	-	-	-	81.95	84.72
10^{-7}	✓	150	0.0999	-	91.33	76.61	85.03	84.34	79.15	89.26	-	-
10^{-7}	✓	100	0.0999	-	91.29	76.77	84.51	84.51	80.15	89.44	-	-
10^{-7}	✓	0	0.0999	-	90.88	76.93	83.99	84.51	78.70	89.00	81.91	84.91
10^{-7}	✓	-	0.0999	84.03	91.17 \pm 0.25	76.77 \pm 0.16	84.51 \pm 0.52	84.45 \pm 0.10	79.33 \pm 0.75	89.23 \pm 0.22	81.93 \pm 0.03	84.81 \pm 0.13
10^{-7}	✗	300	0.0749	-	-	-	-	-	-	-	82.05	84.89
10^{-7}	✗	150	0.0749	-	91.45	77.17	85.89	85.89	80.40	89.97	-	-
10^{-7}	✗	100	0.0749	-	91.61	77.48	85.89	86.57	80.71	90.00	-	-
10^{-7}	✗	0	0.0749	-	91.53	76.54	85.54	85.71	80.81	89.72	81.88	84.89
10^{-7}	✗	-	0.0749	84.73	91.53 \pm 0.08	77.06 \pm 0.48	85.77 \pm 0.20	86.06 \pm 0.46	80.64 \pm 0.22	89.90 \pm 0.15	81.96 \pm 0.12	84.89 \pm 0.00
10^{-6}	✓	300	0.0997	-	-	-	-	-	-	-	81.67	84.71
10^{-6}	✓	150	0.0997	-	91.37	76.54	83.99	87.61	78.94	89.68	-	-
10^{-6}	✓	100	0.0997	-	90.84	77.01	84.34	85.71	78.87	89.55	-	-
10^{-6}	✓	0	0.0997	-	90.92	76.14	83.65	85.54	78.94	89.20	82.37	84.93
10^{-6}	✓	-	0.0997	84.14	91.05 \pm 0.28	76.56 \pm 0.43	83.99 \pm 0.34	86.29 \pm 1.15	78.92 \pm 0.04	89.48 \pm 0.25	82.02 \pm 0.49	84.82 \pm 0.15
10^{-6}	✗	300	0.0765	-	-	-	-	-	-	-	82.28	84.80
10^{-6}	✗	150	0.0765	-	91.21	77.24	85.89	85.20	80.67	89.95	-	-
10^{-6}	✗	100	0.0765	-	91.61	77.01	85.37	86.06	81.05	90.03	-	-
10^{-6}	✗	0	0.0765	-	90.72	76.77	85.54	86.06	80.43	89.51	82.05	84.88
10^{-6}	✗	-	0.0765	84.64	91.18 \pm 0.45	77.01 \pm 0.24	85.60 \pm 0.26	85.77 \pm 0.50	80.72 \pm 0.31	89.83 \pm 0.28	82.16 \pm 0.17	84.84 \pm 0.05
10^{-5}	✓	300	0.1000	-	-	-	-	-	-	-	81.62	84.64
10^{-5}	✓	150	0.1000	-	91.37	77.32	83.65	85.89	79.39	89.67	-	-
10^{-5}	✓	100	0.1000	-	90.96	76.77	83.65	83.82	79.01	89.68	-	-
10^{-5}	✓	0	0.1000	-	91.37	76.14	83.99	85.29	78.30	89.13	82.02	84.88
10^{-5}	✓	-	0.1000	83.99	91.23 \pm 0.23	76.75 \pm 0.59	83.76 \pm 0.20	85.20 \pm 1.19	78.93 \pm 0.51	89.49 \pm 0.31	81.82 \pm 0.28	84.76 \pm 0.17
10^{-5}	✗	300	0.0756	-	-	-	-	-	-	-	82.25	84.86
10^{-5}	✗	150	0.0756	-	91.90	77.17	86.92	86.57	80.81	89.92	-	-
10^{-5}	✗	100	0.0756	-	91.73	77.24	86.57	86.50	81.16	89.91	-	-
10^{-5}	✗	0	0.0756	-	91.69	77.24	85.54	86.75	80.71	89.66	81.96	84.85
10^{-5}	✗	-	0.0756	84.96	91.77 \pm 0.11	77.22 \pm 0.05	86.35 \pm 0.72	86.69 \pm 0.10	80.89 \pm 0.24	89.83 \pm 0.15	82.10 \pm 0.20	84.85 \pm 0.01
10^{-4}	✓	300	0.1000	-	-	-	-	-	-	-	82.03	84.63
10^{-4}	✓	150	0.1000	-	91.21	78.03	82.44	85.89	78.70	89.57	-	-
10^{-4}	✓	100	0.1000	-	90.84	76.69	84.51	85.89	78.73	89.50	-	-
10^{-4}	✓	0	0.1000	-	91.05	77.87	83.48	85.54	78.49	89.20	81.95	85.03

Continued on next page

Table 11 – continued from previous page

10^{-4}	✓	-	0.1000	84.09	91.03 ± 0.18	77.53 ± 0.73	83.48 ± 1.03	85.77 ± 0.20	78.64 ± 0.13	89.42 ± 0.20	81.99 ± 0.06	84.83 ± 0.28
10^{-4}	✗	300	0.0757	-	-	-	-	-	-	-	82.29	84.91
10^{-4}	✗	150	0.0757	-	91.45	78.35	86.23	86.75	80.26	89.81	-	-
10^{-4}	✗	100	0.0757	-	91.37	77.64	86.75	85.37	81.33	90.06	-	-
10^{-4}	✗	0	0.0757	-	91.73	77.40	87.44	85.20	80.64	89.81	82.49	85.05
10^{-4}	✗	-	0.0757	84.99	91.52 ± 0.19	77.80 ± 0.49	86.80 ± 0.60	85.77 ± 0.85	80.74 ± 0.55	89.89 ± 0.15	82.39 ± 0.14	84.98 ± 0.10
10^{-3}	✓	300	0.1000	-	-	-	-	-	-	-	81.81	84.69
10^{-3}	✓	150	0.1000	-	91.25	77.01	85.37	84.51	79.46	89.40	-	-
10^{-3}	✓	100	0.1000	-	91.41	76.85	84.34	86.06	79.35	89.60	-	-
10^{-3}	✓	0	0.1000	-	91.05	77.87	83.48	85.54	79.01	89.00	82.04	85.06
10^{-3}	✓	-	0.1000	84.20	91.23 ± 0.18	77.24 ± 0.55	84.39 ± 0.95	85.37 ± 0.79	79.27 ± 0.24	89.33 ± 0.30	81.93 ± 0.16	84.87 ± 0.26
10^{-3}	✗	300	0.0761	-	-	-	-	-	-	-	82.42	84.89
10^{-3}	✗	150	0.0761	-	92.02	77.87	86.23	87.09	80.81	90.03	-	-
10^{-3}	✗	100	0.0761	-	91.33	78.19	85.20	87.26	81.92	90.00	-	-
10^{-3}	✗	0	0.0761	-	91.61	76.61	86.06	85.54	80.01	89.79	81.90	84.87
10^{-3}	✗	-	0.0761	84.95	91.65 ± 0.35	77.56 ± 0.83	85.83 ± 0.55	86.63 ± 0.95	80.92 ± 0.96	89.94 ± 0.13	82.16 ± 0.37	84.88 ± 0.01
10^{-2}	✓	300	0.1000	-	-	-	-	-	-	-	82.14	84.69
10^{-2}	✓	150	0.1000	-	91.00	77.24	85.20	85.37	78.80	89.51	-	-
10^{-2}	✓	100	0.1000	-	91.09	77.40	83.99	86.23	79.77	89.59	-	-
10^{-2}	✓	0	0.1000	-	90.96	76.46	83.48	84.51	79.46	89.03	82.20	84.92
10^{-2}	✓	-	0.1000	84.17	91.02 ± 0.06	77.03 ± 0.51	84.22 ± 0.88	85.37 ± 0.86	79.34 ± 0.50	89.38 ± 0.30	82.17 ± 0.05	84.81 ± 0.16
10^{-2}	✗	300	0.0759	-	-	-	-	-	-	-	82.31	85.04
10^{-2}	✗	150	0.0759	-	91.77	77.17	85.03	85.37	80.43	89.94	-	-
10^{-2}	✗	100	0.0759	-	91.65	77.95	85.71	85.89	82.10	90.28	-	-
10^{-2}	✗	0	0.0759	-	91.25	76.93	85.37	86.75	80.50	89.69	81.92	84.97
10^{-2}	✗	-	0.0759	84.80	91.56 ± 0.28	77.35 ± 0.54	85.37 ± 0.34	86.00 ± 0.70	81.01 ± 0.94	89.97 ± 0.30	82.11 ± 0.27	85.00 ± 0.05
10^{-1}	✓	300	0.1000	-	-	-	-	-	-	-	81.95	84.78
10^{-1}	✓	150	0.1000	-	91.37	77.01	85.03	84.85	78.87	89.58	-	-
10^{-1}	✓	100	0.1000	-	91.45	77.56	86.06	84.85	79.98	89.62	-	-
10^{-1}	✓	0	0.1000	-	90.84	76.61	85.37	85.71	79.39	89.36	82.04	85.04
10^{-1}	✓	-	0.1000	84.34	91.22 ± 0.33	77.06 ± 0.47	85.48 ± 0.53	85.14 ± 0.50	79.41 ± 0.56	89.52 ± 0.14	82.00 ± 0.06	84.91 ± 0.18
10^{-1}	✗	300	0.0753	-	-	-	-	-	-	-	82.28	85.02
10^{-1}	✗	150	0.0753	-	91.41	78.11	86.23	86.92	80.99	89.76	-	-
10^{-1}	✗	100	0.0753	-	91.37	77.24	85.20	87.09	81.33	90.13	-	-
10^{-1}	✗	0	0.0753	-	91.65	77.24	86.57	86.92	80.67	89.89	82.13	84.80
10^{-1}	✗	-	0.0753	85.00	91.48 ± 0.15	77.53 ± 0.50	86.00 ± 0.72	86.98 ± 0.10	81.00 ± 0.33	89.93 ± 0.19	82.20 ± 0.10	84.91 ± 0.16
1	✓	300	0.0997	-	-	-	-	-	-	-	81.79	84.76
1	✓	150	0.0997	-	92.10	77.48	85.03	85.71	79.04	89.05	-	-
1	✓	100	0.0997	-	91.17	77.09	83.48	84.68	79.98	89.67	-	-
1	✓	0	0.0997	-	91.57	77.01	83.13	86.23	78.94	88.81	81.95	84.85
1	✓	-	0.0997	84.17	91.61 ± 0.47	77.19 ± 0.25	83.88 ± 1.01	85.54 ± 0.79	79.32 ± 0.57	89.17 ± 0.44	81.87 ± 0.11	84.80 ± 0.06
1	✗	300	0.0765	-	-	-	-	-	-	-	82.19	84.89
1	✗	150	0.0765	-	91.65	78.11	86.57	86.57	80.33	90.00	-	-
1	✗	100	0.0765	-	91.57	77.87	85.71	86.23	81.40	89.89	-	-
1	✗	0	0.0765	-	91.29	77.80	85.54	86.40	80.85	89.42	81.95	84.86
1	✗	-	0.0765	84.92	91.50 ± 0.19	77.93 ± 0.16	85.94 ± 0.55	86.40 ± 0.17	80.86 ± 0.54	89.77 ± 0.31	82.07 ± 0.17	84.88 ± 0.02
2	✓	300	0.0992	-	-	-	-	-	-	-	82.07	84.78
2	✓	150	0.0992	-	91.17	76.69	84.51	83.13	79.11	89.41	-	-
2	✓	100	0.0992	-	90.96	77.09	83.13	84.51	79.46	89.51	-	-
2	✓	0	0.0992	-	91.17	77.17	83.48	83.48	78.49	89.07	81.93	84.65
2	✓	-	0.0992	83.82	91.10 ± 0.12	76.98 ± 0.25	83.71 ± 0.72	83.71 ± 0.72	79.02 ± 0.49	89.33 ± 0.23	82.00 ± 0.11	84.71 ± 0.09
2	✗	300	0.0770	-	-	-	-	-	-	-	82.17	84.74
2	✗	150	0.0770	-	92.22	77.01	84.51	86.57	80.19	89.82	-	-
2	✗	100	0.0770	-	91.49	77.01	86.23	85.89	80.74	89.93	-	-
2	✗	0	0.0770	-	91.77	77.64	85.20	86.40	81.05	89.78	82.05	84.70
2	✗	-	0.0770	84.75	91.83 ± 0.37	77.22 ± 0.36	85.31 ± 0.87	86.29 ± 0.36	80.66 ± 0.44	89.84 ± 0.08	82.11 ± 0.08	84.72 ± 0.03

Table 11. **Detailed finetuning results of Foundry-Gate.** All presented results are finetunings from a frozen or unfrozen backbone during distillation using 16 supertokens. During distillation and finetunings, no weight decay is applied on the gate module. For the classification task on ModelNet40 (MN40), OmniObject3D (OO3D), the three splits of ScanObjectNN (SONN-{PB-T50-RS, OBJ-BG, OBJ-ONLY}) and ShapeNet55 (SN55), the used metric is the top-1 accuracy on the validation set. For part segmentation with ShapeNetPart (SNP), we show for both category and instance mIoUs. Mean \pm Std Dev over 10 runs for the baseline, 3 for the classification task, and 2 for the part segmentation task.

Table 12 shows the Spearman correlation values for each dataset between final metrics and gate regularization as a single global value. We observe an inverse correlation between performances and gate regularization. The more we regularize for compressing tokens, the less downstream performance is obtained.

Figure 6 presents the content distillation loss of distilled models for each gate regularization value with specification when we freeze/unfreeze the model during distillation and when we add weight decay on gate. For $\lambda_{\text{gate}} \in [10^{-12}, 10^{-10}]$, we observe an increase in the token compression ratio, leading to 100% of the tokens selected by the gate being merged. We

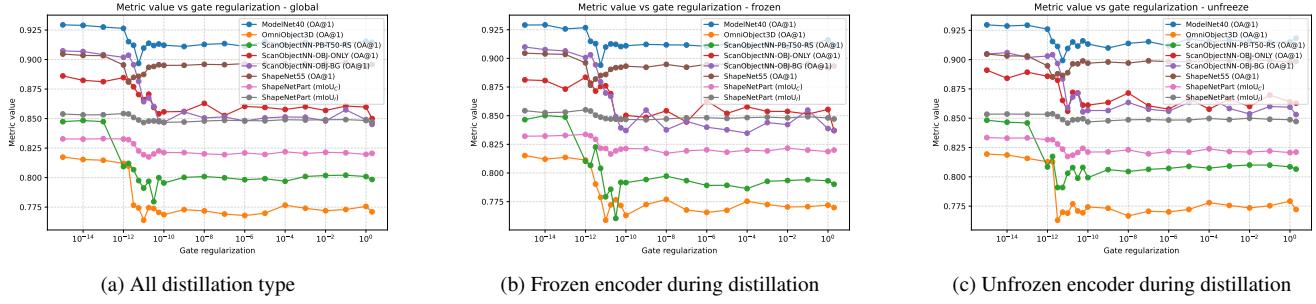


Figure 5. **Fine-tuning results vs λ_{gate} .** No weight decay used on the gate module during distillation are shown. All metrics have been computed on the validation set for each dataset.

Frozen student	MN40 OA@1	OO3D OA@1	SONN-OBJ-BG OA@1	SONN-OBJ-ONLY OA@1	SONN-PB-T50-RS OA@1	SN55 OA@1	SNP mIoU _C	SNP mIoU _I
✓	-0.4542	-0.6583	-0.5198	-0.8024	-0.6953	-0.0257	-0.6630	-0.4397
✗	-0.1339	-0.2451	-0.0934	-0.7556	-0.6225	0.1948	-0.5702	-0.5099
All	-0.1897	-0.5278	-0.3330	-0.8103	-0.7359	0.1472	-0.5741	-0.4091

Table 12. **Correlation between downstream performances and gate regularization of Foundry-Gate.** All presented results are finetunings from a frozen or unfrozen backbone during distillation using 16 supertokens. We compute the Spearman correlation on ModelNet40 (MN40), OmniObject3D (OO3D), the three splits of ScanObjectNN (SONN-{PB-T50-RS, OBJ-BG, OBJ-ONLY}), ShapeNet55 (SN55) and ShapeNetPart (SNP) between the mean of the considered metrics for each dataset on the validation set and λ_{gate} .

also observe an increase in the distillation loss. By combining these two observations, we can state that the distillation loss has a correlation with the token compression ratio. Additionally, with the help of Tab. 12, by transitivity, the distillation loss is also correlated with downstream task performances.

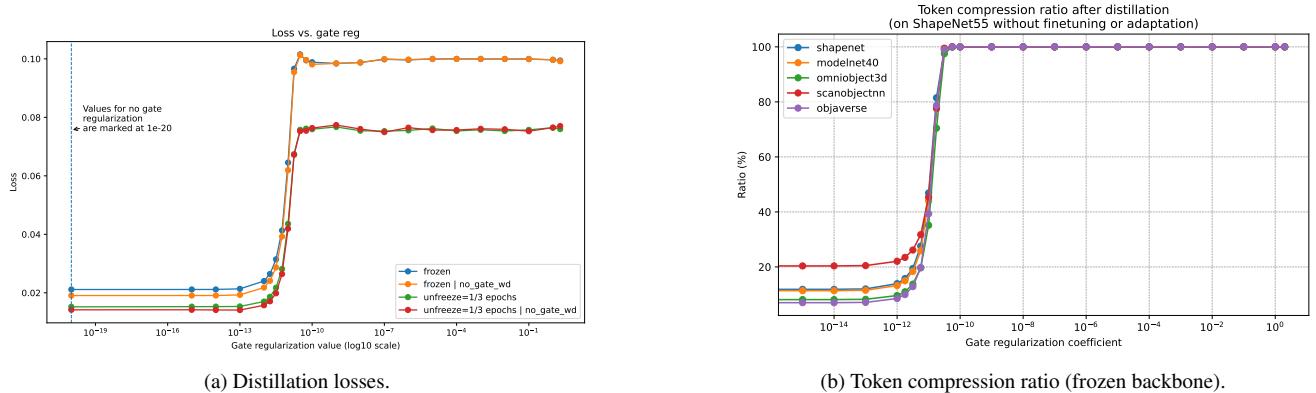


Figure 6. **Effect of gate regularization λ_{gate} on distillation stability.** Distillation loss and token compression ratio curves for frozen and unfrozen backbones across a wide range of λ_{gate} . This figure shows the relationship between the distillation loss, token compression rate and gate regularization. (Left) Several distillation configurations are compared, frozen/unfrozen, add/remove weight decay on the gate module. We provide also token fusion ratios for the validation set on Objaverse [8] for additional results on an unannotated dataset with more diverse objects. (Right) For $\lambda_{\text{gate}} \in [10^{-12}, 10^{-10}]$, we observe the token compression ratio is increasing to obtain 100% of tokens selected by the gate for being merged.

Budget-Aware Inference with Selective Compression. We budgeted the number of tokens that can be processed by the core encoder by, (i) fixing a maximum number of tokens that the encoder can take, (ii) changing the fusion ratio. These two techniques are used at inference-time and using the output probabilities π_i of the gate. Figure 7a illustrates the first technique. By limiting the number of tokens, we see different behavior. For the ShapeNet55 dataset, it is at its maximum when all tokens

are fused. In contrast, for all others the maximum are when we don't fuse between 50/62.5 to 75/87.5% of the input tokens (for 64/128 input tokens). Figure 7b shows the results for the second technique at inference-time. We observe that the best ratio value is for the value used for distilling and fine-tuning the model. We observe a large downfall for ShapeNet55, larger than for any other dataset. We think that it comes from the fact that it was used during pre-training, distillation and fine-tuning and by fixing a r , it does not succeed in inferring the inputs we give. As a hypothesis, we believe that during distillation and fine-tuning, in order to make the ratio more easily changeable at inference time, choosing a random fusion ratio could help models generalize to any value. Table 13 contains all plotted values.

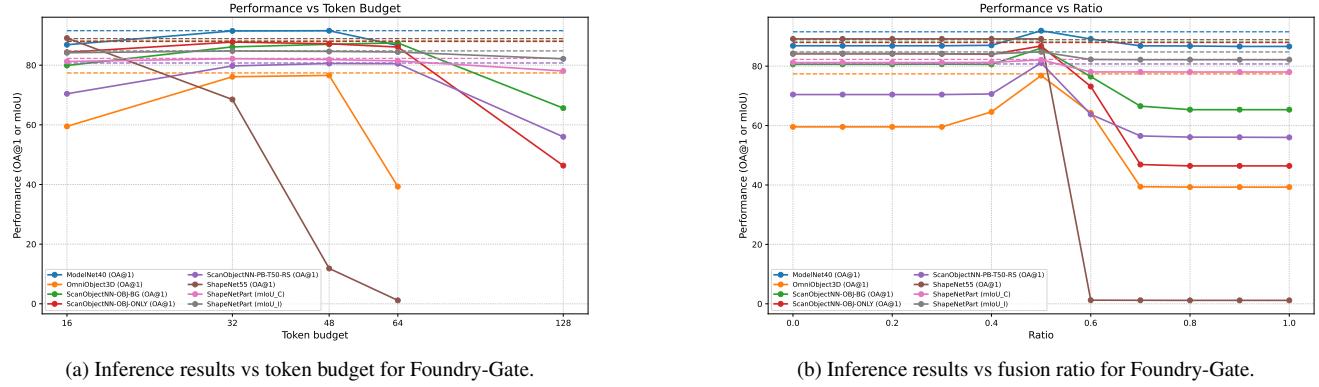


Figure 7. **Inference results for Foundry-Gate.** We use the best backbone on each dataset with $\lambda_{\text{gate}} = 10^{-11}$ and without weight decay. r represents the fusion ratio threshold. All metrics have been computed on the validation set for each dataset.

Token budget	Foundry-Gate r	Avg	MN40	OO3D	SONN-OBJ-BG	SONN-OBJ-ONLY	SONN-PB-T50-RS	SN55	SNP	
									mIoU _C	mIoU _I
Foundry-Gate with $\lambda_{\text{gate}} = 10^{-11}$ baseline (Ours)										
-	-	84.36	90.99 ± 0.77	75.88 ± 2.11	86.98 ± 1.84	87.61 ± 0.60	77.92 ± 1.17	88.59 ± 0.31	82.14 ± 0.21	84.75 ± 0.06
Foundry-Gate (Ours) - Inference token-budgeted										
16	-	79.44	86.86 ± 0.22	59.49 ± 0.59	79.91 ± 0.59	84.34 ± 0.82	70.42 ± 0.43	89.09 ± 0.12	81.25 ± 0.13	84.17 ± 0.06
32	-	82.09	91.49 ± 0.37	76.13 ± 0.69	86.14 ± 0.57	87.76 ± 0.56	79.75 ± 0.39	68.50 ± 0.19	82.17 ± 0.11	84.77 ± 0.05
48	-	75.16	91.56 ± 0.24	76.58 ± 0.38	87.01 ± 0.74	87.18 ± 0.65	80.57 ± 0.32	11.86 ± 0.13	81.88 ± 0.13	84.62 ± 0.05
64	-	68.37	86.63 ± 0.35	39.30 ± 0.26	87.47 ± 0.61	86.09 ± 0.44	80.50 ± 0.33	01.15 ± 0.00	81.44 ± 0.14	84.41 ± 0.05
80	-	-	-	-	-	-	-	-	80.89 ± 0.13	83.91 ± 0.06
96	-	-	-	-	-	-	-	-	80.07 ± 0.17	83.33 ± 0.05
112	-	-	-	-	-	-	-	-	79.17 ± 0.17	82.67 ± 0.06
128	-	-	-	-	65.61 ± 0.46	46.33 ± 0.53	55.99 ± 0.42	-	78.06 ± 0.11	82.14 ± 0.05
Foundry (Ours) - change fusion ratio r										
-	0.0	79.54	86.87 ± 0.38	59.56 ± 0.72	80.62 ± 0.60	84.25 ± 0.78	70.44 ± 0.41	89.18 ± 0.13	81.21 ± 0.14	84.15 ± 0.05
-	0.1	79.54	86.87 ± 0.38	59.56 ± 0.72	80.62 ± 0.60	84.25 ± 0.78	70.44 ± 0.41	89.18 ± 0.13	81.21 ± 0.14	84.15 ± 0.05
-	0.2	79.54	86.87 ± 0.38	59.56 ± 0.72	80.62 ± 0.60	84.25 ± 0.78	70.44 ± 0.41	89.18 ± 0.13	81.21 ± 0.14	84.15 ± 0.05
-	0.3	79.53	86.87 ± 0.38	59.56 ± 0.74	80.67 ± 0.67	84.18 ± 0.77	70.44 ± 0.41	89.18 ± 0.13	81.21 ± 0.14	84.15 ± 0.05
-	0.4	80.18	87.01 ± 0.34	64.61 ± 0.59	80.60 ± 0.70	83.99 ± 0.65	70.63 ± 0.34	89.18 ± 0.13	81.27 ± 0.13	84.15 ± 0.05
-	0.5	84.86	91.94 ± 0.24	76.83 ± 0.59	86.08 ± 0.94	86.78 ± 0.51	81.10 ± 0.31	89.18 ± 0.13	82.12 ± 0.13	84.81 ± 0.05
-	0.6	66.04	89.15 ± 0.27	64.27 ± 0.71	76.49 ± 0.53	73.17 ± 1.14	63.77 ± 0.35	01.21 ± 0.05	78.05 ± 0.09	82.24 ± 0.06
-	0.7	57.20	86.84 ± 0.21	39.42 ± 0.44	66.54 ± 0.64	46.88 ± 0.38	56.51 ± 0.30	01.18 ± 0.01	78.05 ± 0.09	82.17 ± 0.06
-	0.8	56.91	86.79 ± 0.24	39.28 ± 0.43	65.35 ± 0.61	46.42 ± 0.40	56.10 ± 0.37	01.15 ± 0.00	78.03 ± 0.09	82.16 ± 0.06
-	0.9	56.89	86.64 ± 0.22	39.28 ± 0.43	65.35 ± 0.61	46.42 ± 0.40	56.07 ± 0.37	01.15 ± 0.00	78.03 ± 0.09	82.16 ± 0.06
-	1.0	56.88	86.64 ± 0.22	39.28 ± 0.43	65.35 ± 0.61	46.42 ± 0.40	56.01 ± 0.37	01.15 ± 0.00	78.03 ± 0.09	82.16 ± 0.06

Table 13. **Changing ratio at inference-time and token-budgeted inference for Foundry-Gate.** We use the best backbone on each dataset with $\lambda_{\text{gate}} = 10^{-11}$ and without weight decay. The token-budget represents the maximum number of tokens we accept as input in the core encoder. r represents the fusion ratio threshold.

Empirical computational cost. We provide GPU and CPU costs in Tab. 14. The used GPU is a NVIDIA RTX A3000 6GB Laptop GPU and the CPU is Intel Core i9-11950H @ 2.60GHz. Reported values take into account tokenization and encoding steps.

Method	# supertokens	Token budget	Params (M)	MACs (G)	FLOPs (G)	GPU Mem (MiB)	Avg runtime (s)	Throughput (obj/s)
baseline	-	-	21.84	238.624	478.128	1143.0	0.09/0.93	747.11/68.71
ToMe [4] ($r_{tome} = 16$)	-	-	21.84	115.412	231.319	1143.0	0.06/0.63	1091.42/100.96
ToMe [4] ($r_{tome} = 8$)	-	-	21.84	97.8966	196.233	1143.0	0.05/0.60	1174.07/107.56
ToMe [4] ($r_{tome} = 4$)	-	-	21.84	90.0449	180.505	1143.0	0.05/0.58	1213.10/109.90
ToMe [4] ($r_{tome} = 2$)	-	-	21.84	86.572	173.548	1143.0	0.05/0.58	1219.81/110.11
ToMe [4] ($r_{tome} = 1$)	-	-	21.84	85.0621	170.523	1143.0	0.05/0.58	1217.40/109.93
Foundry (Ours)								
Foundry	16	-	22.73	88.9903	178.394	1135.5	0.06/0.61	1160.97/104.32
Foundry (ViT-T)	16	-	6.33	71.6618	143.698	1073.0	0.05/0.58	1335.26/110.52
Foundry	8	-	22.73	78.042	156.464	1135.5	0.05/0.58	1271.58/110.28
Foundry	4	-	22.73	72.5678	145.498	1135.5	0.05/0.59	1309.76/109.08
Foundry	2	-	22.73	69.8307	140.016	1135.5	0.05/0.56	1349.58/113.30
Foundry	1	-	22.73	68.4622	137.274	1135.5	0.05/0.56	1365.92/115.18
Foundry-Gate	16	-	22.78	157.593	315.812	1135.7	0.11/0.97	591.60/65.88
Foundry (Ours) - Inference token-budgeted								
Foundry-Gate	16	16	22.78	87.8332/89.1921	176.076/178.798	1135.7	0.07/0.64	864.02/100.57
Foundry-Gate	16	32	22.78	107.651/107.651	215.774/215.774	1135.7	0.08/0.72	756.92/88.91
Foundry-Gate	16	48	22.78	126.073/127.432	252.674/255.396	1135.7	0.09/0.82	679.59/78.27
Foundry (Ours) - change fusion ratio r								
Foundry-Gate (r=0.1)	16	-	22.78	87.8332/168.012	176.076/336.684	1135.7	0.07/0.66	881.45/97.50
Foundry-Gate (r=0.2)	16	-	22.78	89.1921/163.293	178.798/327.23	1135.7	0.07/0.66	875.99/97.52
Foundry-Gate (r=0.3)	16	-	22.78	159.254/159.141	319.139/318.912	1135.7	0.09/0.72	751.83/88.35
Foundry-Gate (r=0.4)	16	-	22.78	161.028/161.783	322.694/324.205	1135.7	0.11/0.96	588.67/66.36
Foundry-Gate (r=0.5)	16	-	22.78	157.593/156.196	315.812/313.014	1135.7	0.11/0.96	593.70/66.77
Foundry-Gate (r=0.6)	16	-	22.78	152.006/152.006	304.621/304.621	1135.7	0.10/0.94	623.29/67.88
Foundry-Gate (r=0.7)	16	-	22.78	152.006/152.006	304.621/304.621	1135.7	0.10/0.96	623.66/66.88
Foundry-Gate (r=0.8)	16	-	22.78	152.006/153.403	304.621/307.419	1135.7	0.10/0.96	620.73/66.51
Foundry-Gate (r=0.9)	16	-	22.78	152.006/152.006	304.621/304.621	1135.7	0.10/0.95	623.58/67.11

Table 14. **Empirical computational cost at inference-time.** We use random input processed only by the pre-trained considered method. The input contains 64 point clouds of 2^{10} 3D points. We group them into 64 tokens of 32 points each. Runtime and throughput are averaged over 40 runs after 10 warmup runs on an NVIDIA RTX A3000 6GB Laptop GPU/Intel Core i9-11950H @ 2.60GHz (tokenization and encoding are included). All methods take N tokens and output N tokens (regardless of the specific task). ToMe [4] halves the number of tokens at each layer up to the specified r_{tome} . We use unfrozen backbones for Foundry. For Foundry-Gate, we use the model with $\lambda_{gate} = 10^{-11}$ without weight decay. MACs and FLOPs are for the forward pass only and were computed with the `calflops` library. The Token-budget represents the maximum number of tokens we accept as input in the core encoder.

D.3. Visualizations

Embedding visualization. In Fig. 8, we observe that at the encoder output clear clusters appear, in opposition to outputs of the backbone with supertokens where the clusters are more mixed, which explains the degradation in fine-tuning results. This is due to our chosen architecture where in CAU, to recover tokens we upsample from encoder outputs (processed supertokens) and CAM from DSO and we add the output of the tokenizer to add some original information as done in 3DLST [22]. Nonetheless, it proves that our method succeed to extract basis vectors of the representation space. Between embeddings of different compression levels, we can't say that one is better than the other. With ViT-T, we see more close clusters and the separation is less obvious compared to its left neighbor. We think that comes from both compressing the input and the model embedding size. The capacity of encoding data is less important but helps to speed up inferences.

PCA embedding visualization. In Fig. 9, we observe that the supertokens are more inclined to analyze edges rather than semantics by analyzing PCA embeddings per object. We think that the DSO/CAU blocks try to focus the encoder more on discriminative parts for simplifying analysis. This is why for complex objects, the distilled encoder succeeds in classifying them. Additionally, with Fig. 10, we also share some examples where the semantics seem equal or even better. We also see that the edges are more present compared to the baseline. In Fig. 11, some semantics are degraded by being less precise like flower, plane or glass, or completely lost with the bottle or car when the number of supertokens is too low. This demonstrates the need to use enough supertokens not only to improve performances, but for keeping semantic information in the latent space.

CAM visualization. Figure 12 shows the chosen supertokens for three random examples. Firstly, we observe that the number of chosen supertokens is not constant and it depends on the considered example. Some need more supertokens for representing the object and some only one. Also, we remark that the fewer supertokens we defined at the beginning, the more supertokens are used. For the first column, we observe that for 16 supertokens, one is needed but for 8, we use all. We think

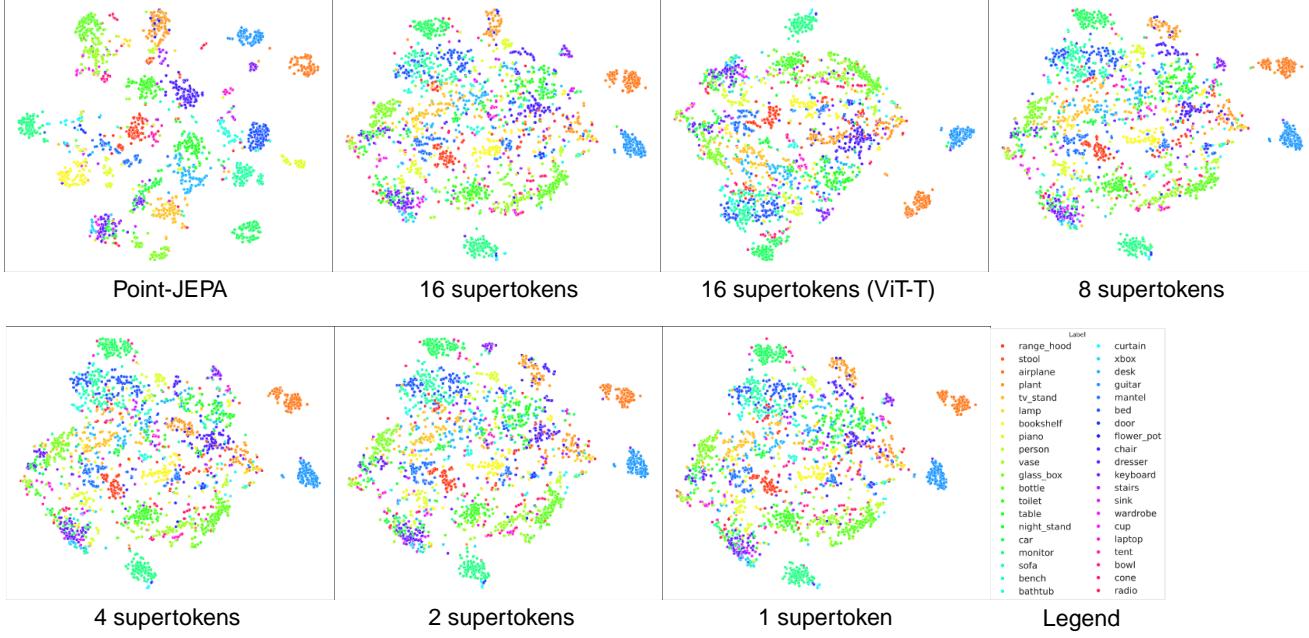


Figure 8. **Embedding visualization for Foundry.** t-SNE [37] projections of the output representation of frozen backbone at distillation ending on ModelNet40 [41] validation set. Point-JEPA outputs are from encoder and CAU for others. If not specified, the Transformer architecture used for student during distillation is ViT-S. We recommend zooming in to see details.

that when we let the model with larger number of supertokens, they are more specialized than smaller, which is expected behavior. Finally, reducing the embedding size 384 of ViT-S to 192 of ViT-T degrades the quality of supertokens. This result makes the link with the observation on the global embedding in Fig. 8.

Supertokens visualization. Figure 13 shows selected tokens on few objects. In this figure, we observe that different supertokens are used for each category of objects, even though some are shared between several classes. When divergent forms appear within the same category, other supertokens tend to be used. This is consistent with the fact that merging is performed after tokenization, without access to the context provided by neighboring tokens, which limits the consideration of more global semantic information. Thus, merging relies mainly on low-level cues—such as local patterns or geometric structures (e.g., the orientation or curvature of a shape). This explains why some supertokens are specialized for very local features, while others, which are more general, appear in several classes. In other words, the distribution of supertokens reflects both the internal geometric diversity within categories and the degree of invariance captured by the merging process.

Figures 14 and 15 respectively displays the embedding space of tokenized inputs and the supertokens set before and after being projected by \mathbf{W}_Q and \mathbf{W}_K . In Fig. 14, we observe that the supertokens all appear in the same location in the token and supertoken space before projection. This result is expected, as the similarity used to construct the CAM is based on a linear projection applied identically to tokens and supertokens via the two weight matrices. Figure 15 is therefore more informative: it represents the space in which the fusion is actually performed. In this space, supertokens are generally close to the tokens assigned to them. However, some are not. We believe that this discrepancy stems from the effects of t-SNE projection: the dimensions relevant for distinguishing certain tokens may not be correctly reproduced in 2D, which can mask the actual relationships that exist in high-dimensional space.

E. Hyper-parameters

This section highlights the used hyper-parameters for distillation and fine-tunings. Table 15 shows used values for important hyper-parameters during distillation. For fine-tunings, we use traditional hyper-parameters and for those specific for Foundry and Foundry-Gate, we vary the value of the epoch when we unfreezing the encoder for classification (0, 100 and 150) and part segmentation (0 and 300) tasks and we fine-tuned only the distilled checkpoints which no weight decay is added for gate

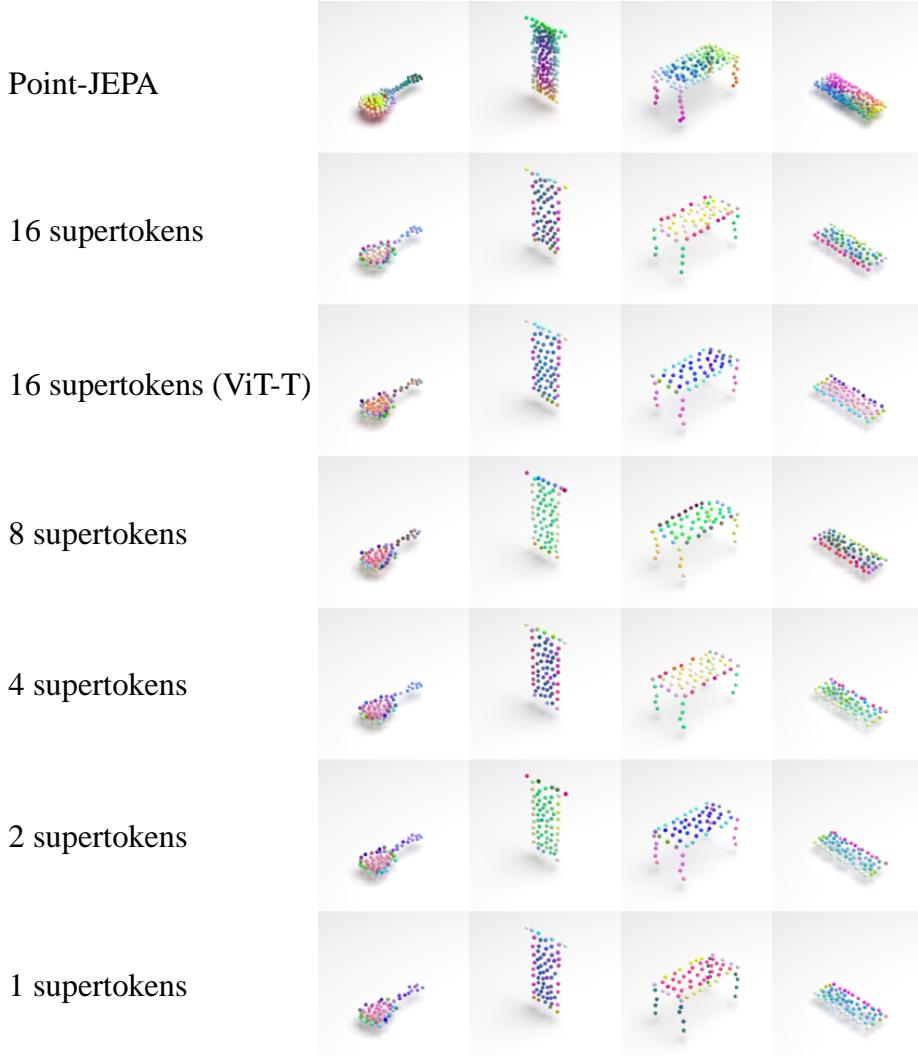


Figure 9. PCA embedding visualization for Foundry which changes feature quality by focusing on object edges for few examples. PCA projections of the output representation of frozen backbone at distillation ending on ModelNet40 [41] validation set in RGB space. If not specified, the Transformer architecture used for student during distillation is ViT-S. We recommend zooming in to see details. The associated labels for each object are ordered as follow: guitar, curtain, table and keyboard.

module. For OmniObject3D, we use ModelNet40 configuration.

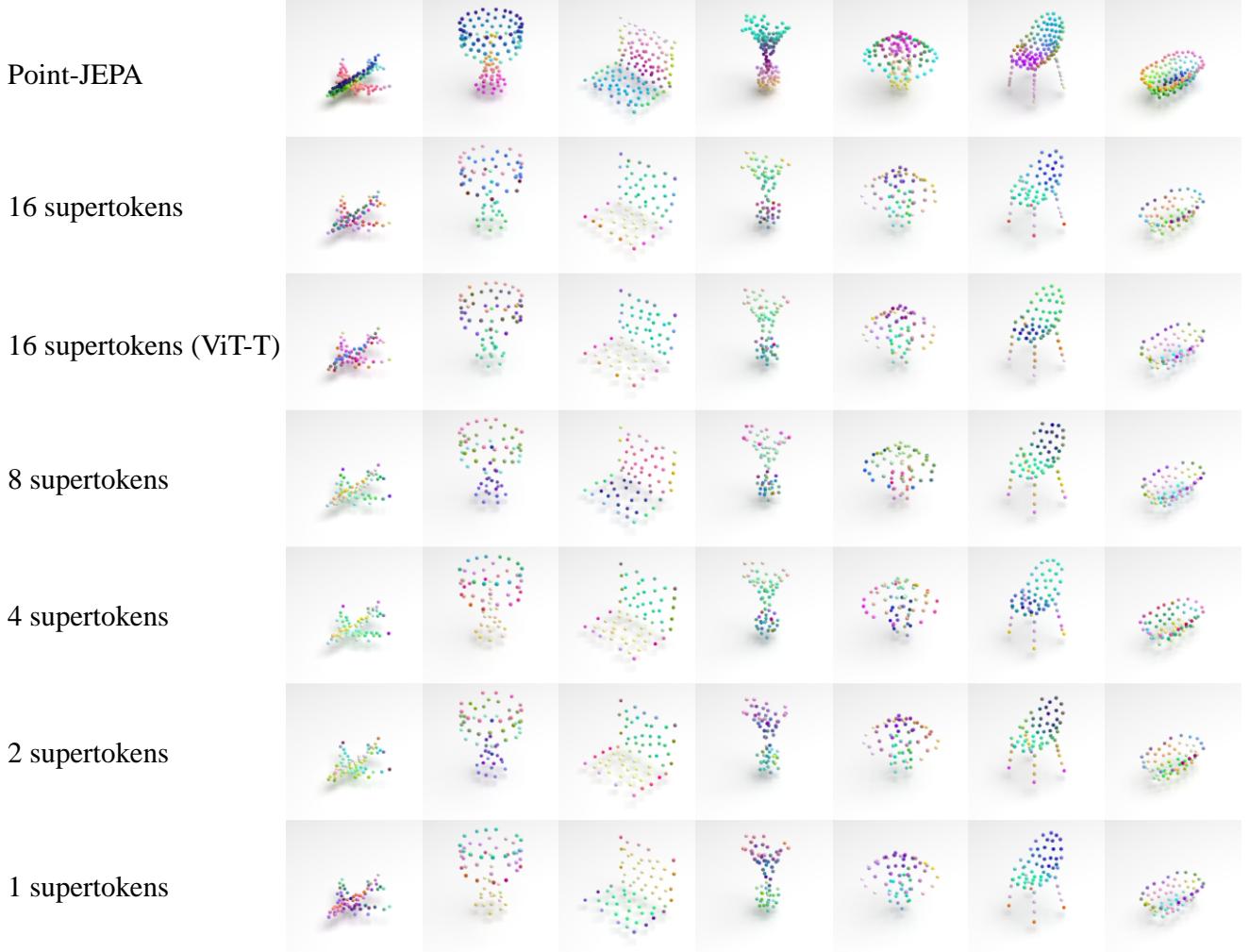


Figure 10. PCA embedding visualization for Foundry for keeping feature quality on some selected examples. PCA projections of the output representation of frozen backbone at distillation ending on ModelNet40 [41] validation set in RGB space. If not specified, the Transformer architecture used for student during distillation is ViT-S. We recommend zooming in to see details. The associated labels for each object are ordered as follow: airplane, lamp, laptop, plant, cone, chair and bathtub.

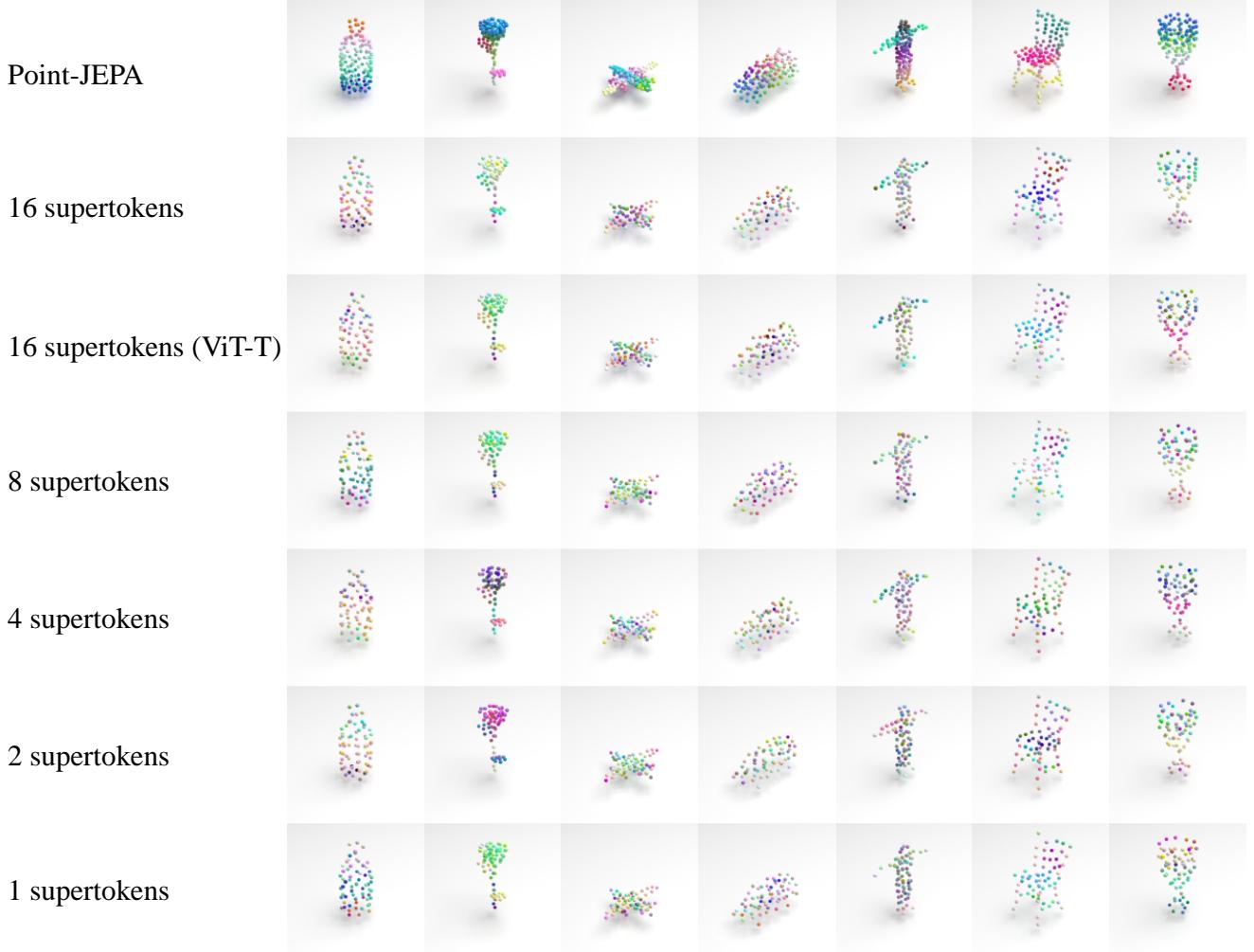


Figure 11. PCA embedding visualization for Foundry for lower feature quality examples. PCA projections of the output representation of frozen backbone at distillation ending on ModelNet40 [41] validation set in RGB space. If not specified, the Transformer architecture used for student during distillation is ViT-S. We recommend zooming in to see details. The associated labels for each object are ordered as follow: bottle, plant, airplane, car, person, chair and cup.

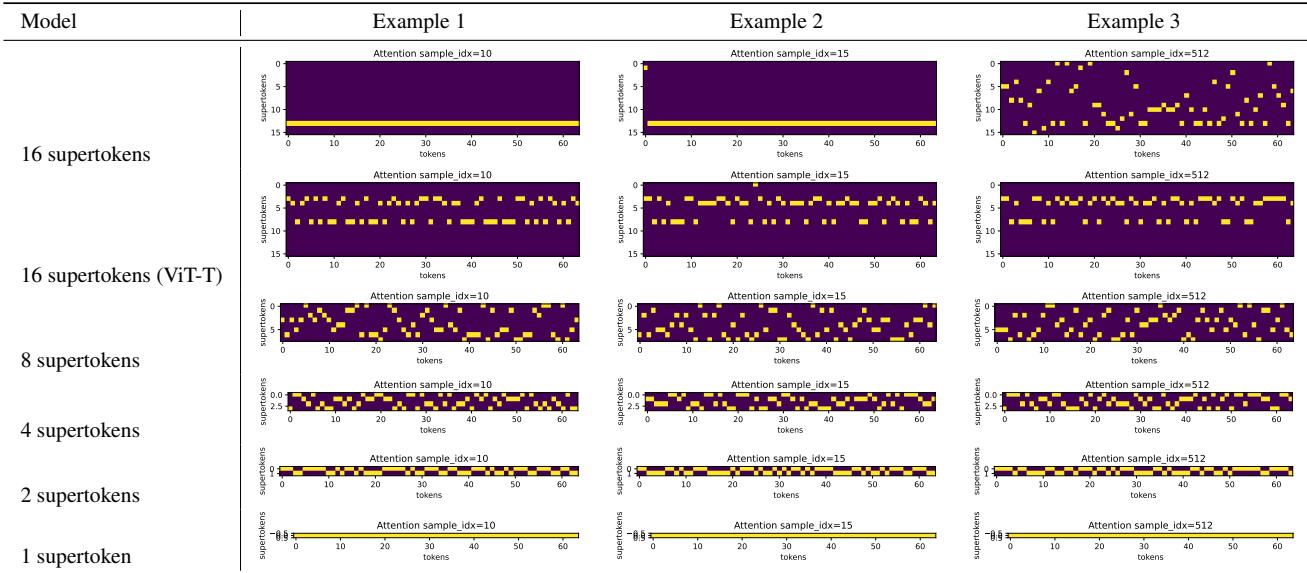


Figure 12. **Cross-Attention Map (CAM) comparison.** Unless specified, ViT-S backbone is used for student branch during distillation. The first example represents a person, second a bottle and third a toilet object. The examples come from validation set of ModelNet40 [41]. The used backbones come from the end of distillation stage.

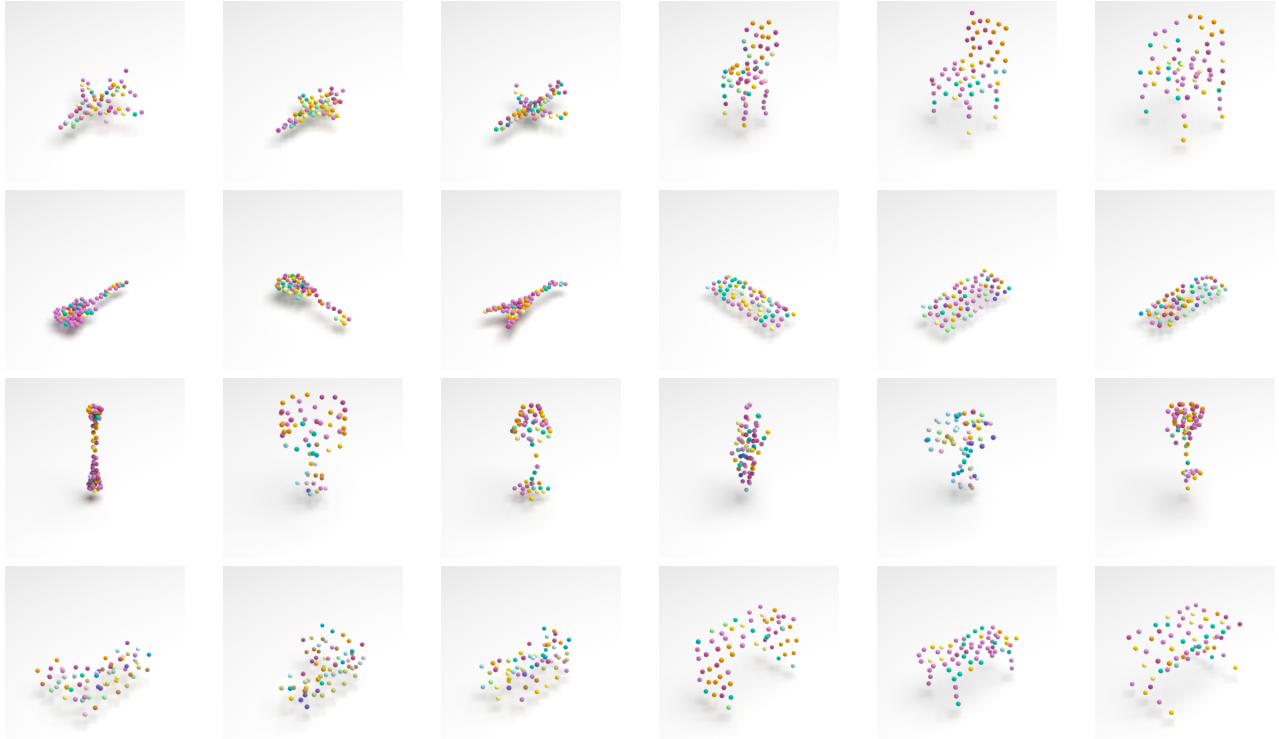


Figure 13. **Qualitative supertokens selection view in 3D space.** We select examples from following categories: airplane, chair, guitar, keyboard, lamp, plant, sofa and, table. Colors are unique and are associated to supertokens throughout all selected examples. The examples come from validation set of ModelNet40 [41]. Classifier is the 16 supertokens best checkpoint.

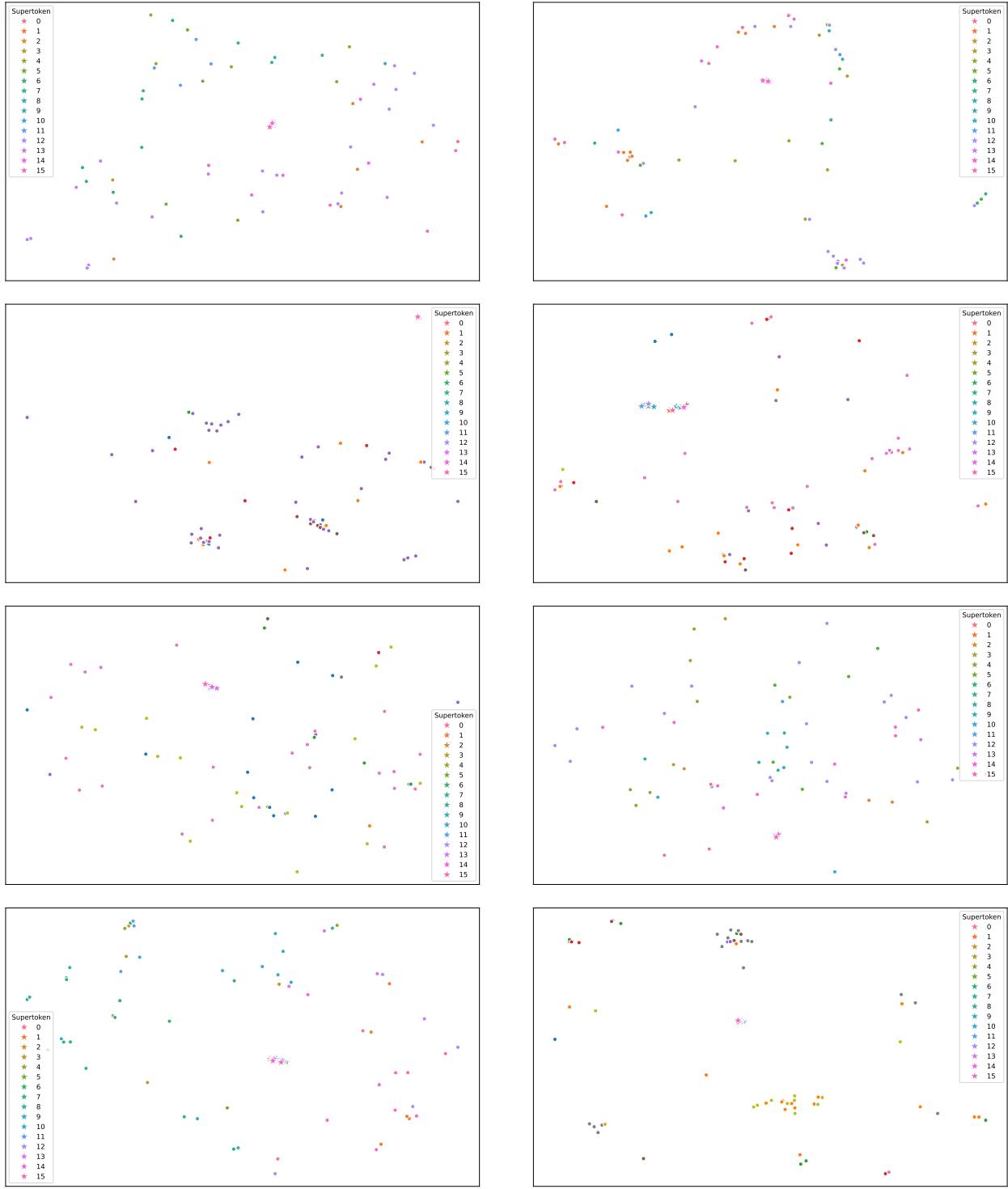


Figure 14. Qualitative supertokens selection view in token-supertoken space before projecting. We select examples from following categories: airplane, chair, guitar, keyboard, lamp, plant, sofa and, table. Colors are unique and are associated to supertokens throughout all selected examples. The examples come from validation set of ModelNet40 [41] and are the same as Fig. 13. Classifier is the 16 supertokens best checkpoint.

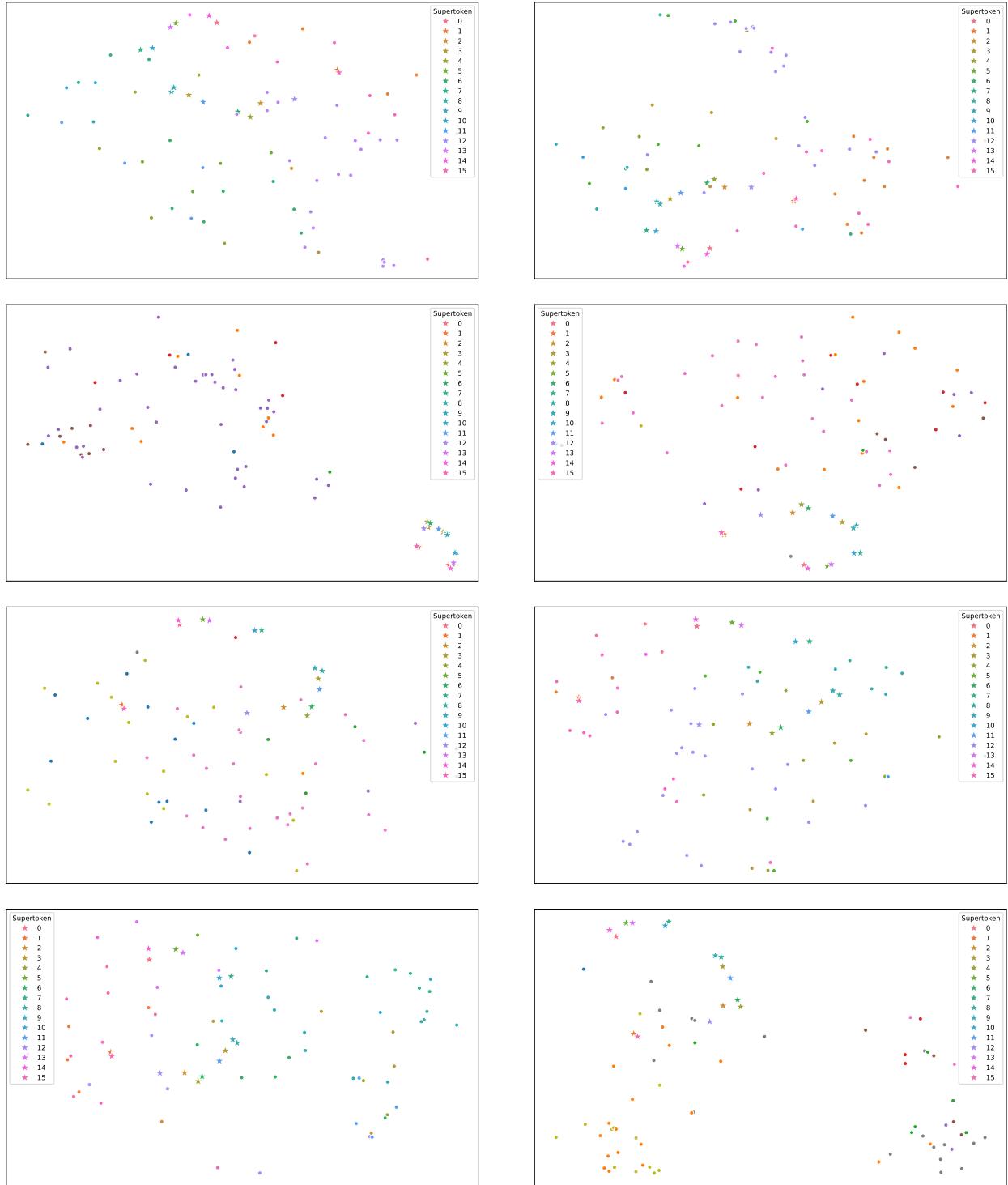


Figure 15. Qualitative supertokens selection view in token-supertoken space after projecting. We select examples from following categories: airplane, chair, guitar, keyboard, lamp, plant, sofa and, table. Colors are unique and are associated to supertokens throughout all selected examples. The examples come from validation set of ModelNet40 [41] and are the same as Fig. 13. Classifier is the 16 supertokens best checkpoint.

Hyper-parameter	Foundry	Foundry-Gate
<i>data</i>		
datasets	ShapeNet55 [6]	ShapeNet55 [6]
# points	1024	1024
<i>augmentation</i>		
scale	(0.8, 1.2)	(0.8, 1.2)
anisotropic scaling	enabled	enabled
flip	y axis	y axis
rotate Y axis	$[-\pi, \pi]$	$[-\pi, \pi]$
<i>optimization</i>		
batch size	512	512
max epochs	150	150
optimizer	AdamW [21]	AdamW [21]
weight decay	5×10^{-2}	5×10^{-2}
weight decay on gate	-	{false, true}
momentum	0.9	0.9
lr	10^{-3}	10^{-3}
start lr	10^{-6}	10^{-6}
final lr	10^{-6}	10^{-6}
lr scheduler	Cosine annealing	Cosine annealing
warmup type	linear	linear
warmup epochs	15	15
encoder unfreeze	{50, max epochs}	{50, max epochs}
<i>architecture</i>		
teacher encoder	ViT-S	ViT-S
student encoder	ViT-T/S	ViT-S
# groups/tokens	64	64
group size	32	32
max # supertokens	$\{2^i\}_{i=1}^4$	16
use supertoken P	false	false
bias on DSO qkv	false	false
arg max in DSO	true	true
λ_{gate} values	-	$\{0\} \cup \{\, 10^i \}_{i=-15}^1 \cup \{\, 10^i \mid i \in A\},$ $A = \{-10.25, -10.5, -10.75, -11.25, -11.5, -11.75\}$
<i>hardware</i>		
dtype	float32	float32
accelerator	1 A100 80G	1 A100 80G

Table 15. Distillation hyper-parameters for Foundry and Foundry-Gate.