PROJECT HOT TOPICS IN COMPUTER VISION - NERF WEIGHT SPACE LEARNING

Zilong Liu
Field of Study: Geoinfomation Science
Technische Universität Berlin
Enrollment Number: 501297
zilong.liu@campus.tu-berlin.de

Roman Oelfken
Field of Study: Information Systems
Technische Universität Berlin
Enrollment Number: 519746
roman@oelfken.net

Berk Can Özmen
Field of Study: Computer Science
Technische Universität Berlin
Enrollment Number: 396694
ozmenberkcan@gmail.com

Can-Philipp Tura
Field of Study: Computer Science
Technische Universität Berlin
Enrollment Number: 379626
c.tura@outlook.de

July 20, 2025

ABSTRACT

We extend the SANE (Schürholt et al. [2024]) algorithm for weight-space learning to the domain of Instant Neural Graphics Primitives (Instant-NGP) (Müller et al. [2022]), enabling compact, structured representations of 3D scenes directly in parameter space. Our approach relies on a dataset of neural fields representing diverse objects under controlled rotations, where the sampling of rotated views encourages the separation of object-specific and rotation-specific factors. We propose a transformer-based autoencoder architecture that learns latent representations by tokenizing Instant-NGP's multi-resolution hash grids and MLP weights. Our tokenization strategy is tailored to the structural and spatial properties of Instant-NGP, incorporating specialized positional encodings that preserve both 3D coherence and architectural organization. The model is trained with a multi-objective loss combining reconstruction, contrastive learning, and rotation prediction, with the aim of learning disentangled and reusable latent embeddings. Our results demonstrate success at scene reconstruction and rotation-invariant representations, while generalization across rotational views remains challenging and appears to require fundamentally different data or modeling strategies.

1 Introduction

Neural Radiance Fields (NeRFs) have recently emerged as a powerful technique for representing complex three-dimensional scenes through neural networks. Unlike traditional explicit 3D representations, such as voxel grids or meshes, NeRFs implicitly encode scene geometry and appearence such as angle-dependant reflections in the weights of a multilayer perceptron (MLP). Mildenhall et al. [2021] By learning to map spatial coordinates and viewing directions to color and density values, NeRFs achieve highly realistic renderings and have significantly advanced applications in view synthesis, 3D reconstruction, and augmented reality. Mildenhall et al. [2021]

However, manipulating and analyzing the learned representations encoded within NeRF weights remains challenging. Li et al. [2018] Particularly, tasks such as identifying and controlling geometric transformations, such as rotation, within the weight space have not yet been fully explored. This paper addresses this gap by applying Structured Affine Neural Embeddings (SANE), a novel technique for weight space learning, to explicitly identify and interpret rotational transformations embedded within NeRF models. Schürholt et al. [2024]

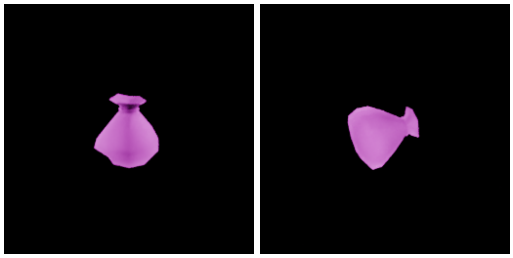


Figure 1: The same NeRF-rendered object under two distinct rotations. (images provided by our project supervisor) The left image shows the canonical orientation (base), while the right image is rotated by approximately (90°, 0°, 90°).

Our objective is to enable explicit extraction and manipulation of rotational parameters from NeRF weight data, as illustrated in Figure 1. The depicted rotations highlight the core challenge of identifying such transformations solely from the learned network parameters. By focusing on these examples, we aim to provide deeper insights into how geometric transformations are represented internally within the model. Understanding this structure not only enhances interpretability and scene manipulation, but also offers valuable insights for downstream tasks such as alignment, transfer learning, and model adaptation.

We demonstrate that by leveraging the structured embedding capabilities of SANE, rotational attributes in weight space can be reliably identified, interpreted, and manipulated, providing a foundation for future research on interpretable neural scene representations and enhanced controllability of NeRF-generated content.Schürholt et al. [2024]

While we build upon the SANE framework and draw conceptual inspiration from the Instant-NGP architecture, significant modifications are required to adapt both the tokenization of model parameters and the positional encoding mechanisms to the structure of NeRFs. Müller et al. [2022] The original SANE formulation was not designed with implicit 3D representations or volumetric rendering in mind, which necessitates careful redesign of the input embeddings and structural priors used during training. In particular, we propose tailored weight-token mappings and introduce rotation-aware encodings that better reflect the geometric nature of NeRF-based scene representations.Schürholt et al. [2024]

2 Related Work

A precise understanding of prior work is essential to contextualize our contributions. As already mentioned, we focus on two key methods: Instant Neural Graphics Primitives (Instant-NGP) and Structured Affine Neural Embeddings (SANE). Instant-NGP advances Neural Radiance Fields (NeRFs) by introducing an efficient, hash-based spatial encoding, while SANE provides a framework for learning structured representations in neural weight space using a transformer-based autoencoder. Both approaches are central to the methodology of this study.

2.1 From Classical NeRFs to Instant-NGP

Classical NeRFs, as introduced by Mildenhall et al. Mildenhall et al. [2021], represent scenes using a deep MLP that maps a 3D coordinate $\mathbf{x} \in \mathbb{R}^3$ and view direction $\mathbf{d} \in \mathbb{S}^2$ to radiance and density values. To capture high-frequency details, NeRF employs sinusoidal positional encoding of the inputs. While capable of producing photorealistic novel views, training classical NeRFs is computationally expensive and often takes hours or days.

Instant-NGP Müller et al. [2022] addresses these challenges by replacing Fourier-based positional encoding with a multi-resolution hash grid. In this design, voxel vertices are mapped via a hash function to indices in compact feature tables. Retrieved features are linearly interpolated and concatenated for multiple resolution levels before being processed by a shallow MLP with only a few layers. This method, illustrated in Figure 2, captures both low- and high-frequency scene components while enabling real-time training and inference.

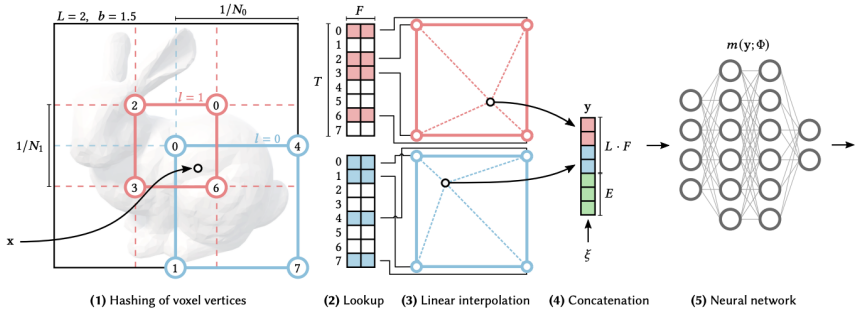


Figure 2: Figure by Müller et al. [2022]. Overview of Instant-NGP's multiresolution hash encoding. Input coordinates $\mathbf{x}$ are mapped to voxel vertices at different resolutions using a hash function (1), the corresponding feature vectors are retrieved (2), interpolated (3), and concatenated (4). A compact MLP (5) then processes this representation. The hash function efficiently compresses spatial information while preserving scene detail.

The hash function maps spatial voxel coordinates to entries in a compact feature table, enabling a sparse yet expressive representation of the scene. This design avoids the memory overhead of dense voxel grids while maintaining high-frequency details through multiple resolution levels. The hash mapping is central to Instant-NGP's efficiency and will play a key role in our analysis of the weight space.

2.2 SANE

SANE, introduced by Schürholt et al. Schürholt et al. [2024], shifts the focus from input-output mappings to the weight space of neural networks. SANE tokenizes a model's parameters into sequences that are processed by a transformer-based autoencoder. The encoder g_θ compresses these tokens into a latent hyper-representation z, while the decoder h_ψ reconstructs or samples new weight configurations. This framework captures structural dependencies across layers and enables both discriminative tasks and generative tasks.

2.3 Relevance to Our Study

We combine the computational efficiency of Instant-NGP with SANE's weight-space embeddings to analyze geometric transformations from NeRF models. To achieve this, we adapt SANE's tokenization and positional encoding to the specific structure of Instant-NGP, introducing rotation-aware embeddings that directly capture transformations within the weight parameters.

3 Dataset

Our dataset consists of 342 distinct 3D objects, each represented by up to 6 different Instant-NGP neural networks trained on the same object under different rotational viewpoints. We treat each object as a single data point (dataset size = 342), where the multiple rotational views serve as natural augmentations of the same underlying 3D structure.

Each Instant-NGP model contains two distinct components: 16 hash table layers with approximately 6.5 million parameters, and 6 MLP layers with around 20,000 parameters. This architectural division creates a fundamental challenge for weight space analysis, as these components encode spatial information through entirely different mechanisms.

4 Methods

4.1 Problem Formulation

Instant-NGPs encode a neural radiance field as a function $f : \mathbb{R}^n \to \mathbb{R}^m$ that maps 3D coordinates and direction to color and density values. This mapping is decomposed into two stages: a hash encoding function $h : \mathbb{R}^3 \to \mathbb{R}^{32}$ that maps 3D points to high-dimensional features, followed by an MLP $m : \mathbb{R}^{32} \to \mathbb{R}^4$ that produces the final output, such that $f = m \circ h$.

The hash table and MLP components encode complementary but fundamentally different types of information. The hash table employs 16 parallel tables operating at different spatial resolutions, where semantic information about any given 3D point is distributed across multiple low-dimensional hash entries. In contrast, the MLP processes information sequentially through high-dimensional layers, with each layer building upon the previous one's representations.

To apply transformer-based methods to Instant-NGP weights, we must address the fundamental challenge of converting these heterogeneous parameter sets into a uniform token sequence $\mathbf{X} \in \mathbb{R}^{C \times D}$, where C is the sequence length and $D = 256$ is the token dimension.

4.2 Weight Space Representation

4.2.1 Hash Table Representation

Our hash representation strategy leverages the spatial structure inherent in multi-resolution hash tables. For any 3D point, the hash encoding retrieves 8 surrounding vertices from each of the 16 resolution levels, where each vertex stores a 2D feature vector. We concatenate these features to create a single token: $8 \times 16 \times 2 = 256$ dimensions per token.

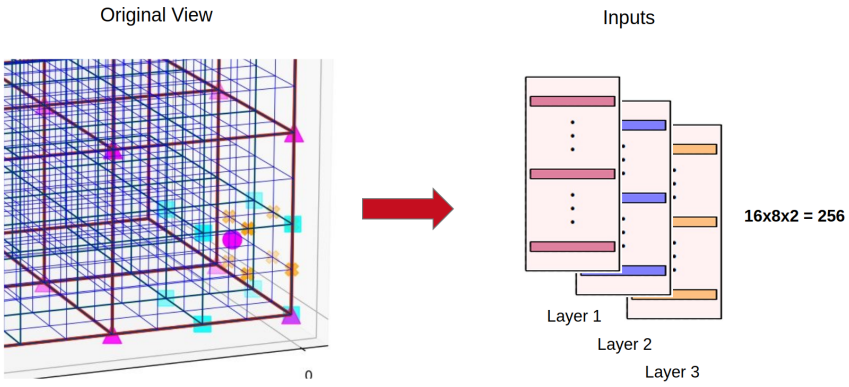


Figure 3: Representation of hash layers. Given a point in 3D space (purple circle), we identify the surrounding vertices at each resolution level. For each layer (layer 1: purple triangles, layer 2: blue squares, layer 3: orange crosses), we concatenate the 2D hash vectors of all 8 vertices using the hash encoding scheme from Müller et al. [2022].

To construct the hash portion of our input, we uniformly sample 380 points throughout the 3D space and convert each to a 256-dimensional token, yielding $\mathbf{H} \in \mathbb{R}^{380 \times 256}$. This sampling strategy ensures that our token representation captures the spatial distribution of information encoded in the hash tables.

Our sampling strategy has an important consequence for the frequency characteristics of the learned representations. Since our task involves predicting rotation information, preserving low-frequency spatial information is crucial. However, our sampling strategy does not treat each hash layer equally in terms of coverage. The earlier layers contain smaller hash tables - for instance, the first layer has only 4096 indices. When we sample 380 points and retrieve 8 surrounding vertices each (380 × 8 = 3040 indices), we effectively access nearly the entire first layer for every input. This means that low-frequency features from smaller hash tables are significantly overrepresented compared to high-frequency features from larger tables.

4.2.2 MLP Representation

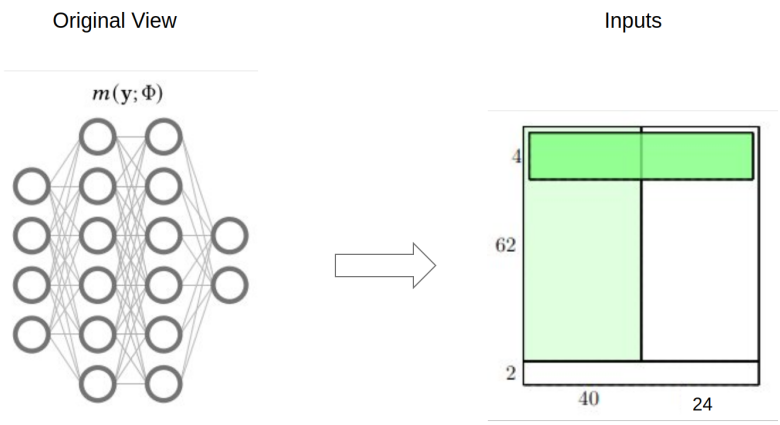


Figure 4: Representation of MLP layers. Each MLP layer (shown as 62×40, light green) is first padded to 64×64 (or to closest bigger integer that is divisible by 4), then represented by grouping consecutive 4 neurons (dark green) into 256-dimensional tokens.

The MLP component requires a different representation approach due to its sequential, fully-connected structure. A MLP can be thought as a concatenation of linear transformations and non linear activation functions as $a(t_i) : \mathbb{R}^n \to \mathbb{R}^m$. In this sense, every neuron in a layer shares mutual information, and every layer shares mutual information with the consecutive or other layers. In the original SANE algorithm Schürholt et al. [2024], we can see that, this problem can be overcome, however since MLP layers are significantly smaller than hash tables (20K vs 6.5M parameters), we include the entire MLP network to preserve its representational capacity.

Another difference between the MLP and hash structure is that MLP layers are effected by the permutation problem as discussed in Schürholt et al. [2024]. This means that there exist other functionally equivalent MLP networks than we are trying to learn on. While hash layers can suffer from a similar problem, where many different versions of encoding the 3D space are possible, it usually requires a change in the subsequent MLP network or non linear transformations in the representation space.

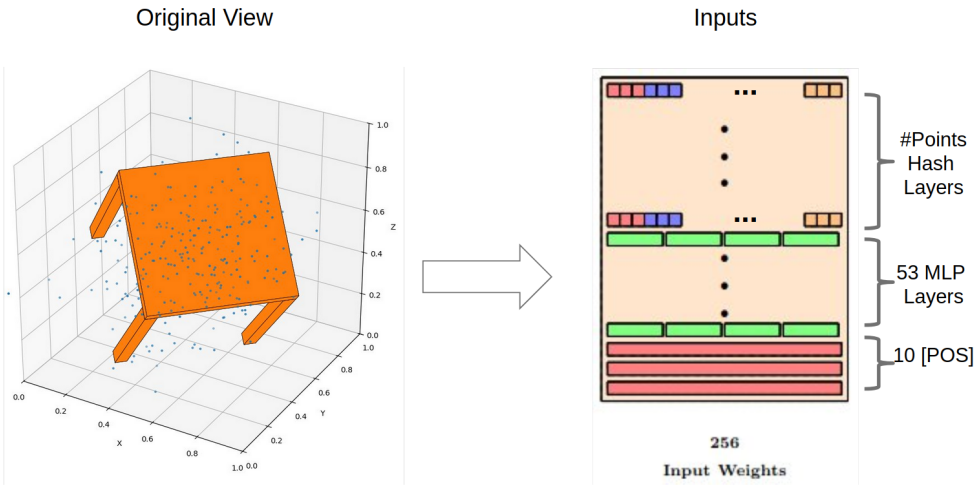


Figure 5: Complete input construction pipeline. From the 3D space (left), we sample 380 points uniformly in $[0, 1]^3$, each converted to a 256D token by concatenating hash features across resolution levels (colored squares represent vertices from different layers). MLP layers (green) and POS tokens (red) are then concatenated to form the final input sequence.

Thus to tackle this, we use random permutations (consistent across layers) of our MLP network. Then we pad each layer to a maximum width of 64 neurons (the largest width in our dataset) and group consecutive sets of 4 neurons into single tokens, yielding $64 \times 4 = 256$ dimensions per token. This process generates $\mathbf{M} \in \mathbb{R}^{53\times256}$ for the complete MLP component.

4.2.3 Positional Tokens for Rotation Prediction

For our downstream rotation prediction task, we append 10 [POS] tokens initialized as zeros: $\mathbf{P} = \mathbf{0} \in \mathbb{R}^{10\times256}$. These tokens serve as dedicated outputs for the rotation prediction head, allowing the model to specialize certain latent dimensions for rotational reasoning.

The final input sequence is constructed by concatenating all components: $\mathbf{X} = [\mathbf{H}; \mathbf{M}; \mathbf{P}] \in \mathbb{R}^{443\times256}$.

4.3 Tokenization and Positional Encoding

Since we have 2 different ways of representing two different semantic information, for both tokenization and positional encoding, we treat hash inputs and MLP inputs separately. For the tokenization process, we use two different learned linear projections to create input tokens. Then our input to encoder becomes:

$$\begin{aligned} \mathbf{T}_h &= Linear(\mathbf{H}) \in \mathbb{R}^{256,1024} \\ \mathbf{T}_m &= Linear(\mathbf{M}) \in \mathbb{R}^{256,1024} \\ \mathbf{T} &= [\mathbf{T}_h, \mathbf{T}_m, \mathbf{P}] \in \mathbb{R}^{443\times1024} \end{aligned} \tag{1}$$

4.3.1 Hash Token Positioning

Hash tokens require two types of positional information. First, we encode the 3D spatial coordinates of each sampled point using a learned linear projection: $\mathbf{PE}_{\text{x,y,z}} = \text{Linear}(\mathbf{xyz}) \in \mathbb{R}^{380x256}$.

Second, following Schürholt et al. [2024], we encode the structural position within the hash table using a 3-dimensional index $\mathbf{PE}_{\text{n,l,k}} = [n, l, k]$, where $n \in [1, N]$ indicates the global sequence position, $l \in [1, L]$ represents the layer index, and $k \in [1, K(l)]$ denotes the position within that layer.

And finally we simply sum input tokens with all the position tokens: $\mathbf{PE}_{\text{hash}} = \mathbf{PE}_{\text{x,y,z}} + \mathbf{PE}_n + \mathbf{PE}_l + \mathbf{PE}_k$

4.3.2 MLP and POS Token Positioning

MLP tokens receive unique learned embeddings based on their sequential position (indices 0-52), while [POS] tokens are assigned distinct embeddings (indices 53-62). This approach reflects the fixed, sequential nature of MLP processing compared to the spatial sampling of hash tables.

And finally, the positional encoding is constructed by concatenating all components $\mathbf{PE} = [\mathbf{PE}_{hash}; \mathbf{PE}_{mlp}; \mathbf{PE}_{POS}] \in \mathbb{R}^{443x256}$

4.4 Data Augmentation

We apply three augmentation strategies during training. First, multiplicative Gaussian noise ($\sigma = 0.1$)is added to all input tokens to improve robustness. Second, we randomly permute the order of MLP layers, leveraging the insight from Schürholt et al. [2024] that neural network layers can often be reordered without affecting the encoded function. Lastly, we add a small noise ($\sigma = 0.1°$) in euler degrees to the true rotation of the views.

4.5 Architecture and Training Objectives

Our model employs a transformer-based autoencoder architecture with separate projection heads for different tasks. The encoder maps input tokens to a latent representation $\mathbf{Z} \in \mathbb{R}^{443\times128}$, which is then processed by task-specific heads.

4.5.1 Reconstruction Loss

The autoencoder reconstructs the original input tokens through an encoder-decoder framework:

$$\begin{aligned} \mathbf{Z} &= \text{Encoder}_\theta(\mathbf{T}, \mathbf{PE}) \\ \hat{\mathbf{T}} &= \text{Decoder}_\theta(\mathbf{Z}, \mathbf{PE}) \end{aligned} \tag{2}$$

The original autoencoder objective goes beyond optimizing for a downstream task — it forces the model to learn a compact and informative representation of the dataset itself. By requiring the network to reconstruct its inputs from a compressed encoding, the model is encouraged to capture the underlying structure and dynamics of the data. Similar to Schürholt et al. [2024], we use a mask $\mathbf{M}$ with element-wise multiplication ($\odot$), where padded positions of $\mathbf{T}$ and $[POS]$ tokens are not considered in reconstruction.

$$\mathcal{L}_{\text{recon}} = \left\| \mathbf{M} \odot \left(\hat{\mathbf{T}} - \mathbf{T} \right) \right\|_2^2 \tag{3}$$

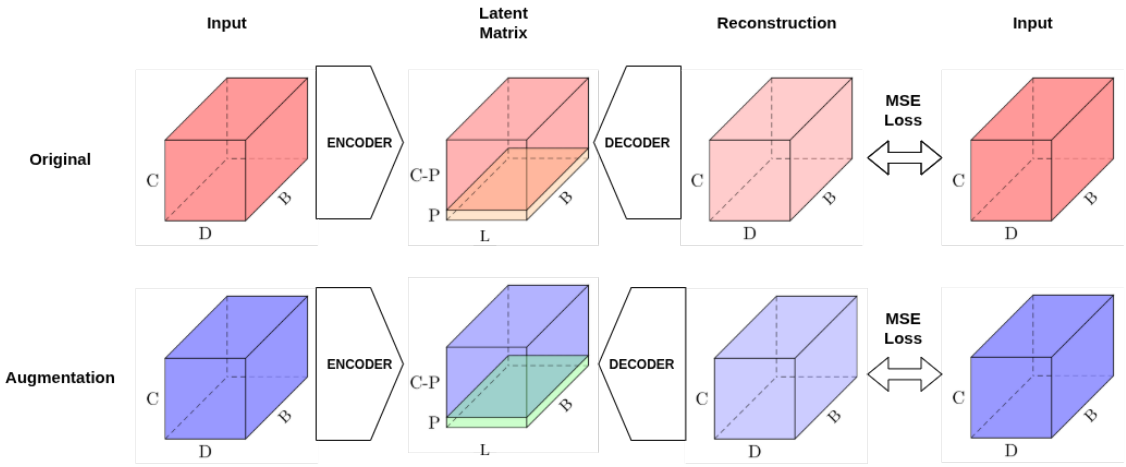


Figure 6: Reconstruction pathway through the autoencoder architecture. Each input sequence $\in \mathbb{R}^{B,C,D}$ is passed through an autoencoder to create a latent representation $\mathbf{z} \in \mathbb{R}^{B,C,L}$. Note that the latent matrix is also further divided into two components along the context (C) dimension, where orange and green parts represents the part of the latent matrix that corresponds to the [POS] tokens. Then the latent matrix $\mathbf{z}$ is passed through the decoder to create reconstructed inputs $\hat{\mathbf{X}}$. Afterwards each reconstruction is only compared to it's own view, so no information between the original and augmented view is shared at this stage.

4.5.2 Contrastive Learning Loss

The contrastive learning objective encourages the model to learn rotation-invariant representations of 3D objects. Rather than reconstructing the input, the model is trained to recognize that two differently rotated views of the same object should produce similar latent embeddings. By pulling together the representations of augmented and original views in latent space—while simultaneously pushing apart representations from different objects—the model learns to abstract away irrelevant spatial transformations and focus on intrinsic object identity. This promotes the emergence of a content-centric representation that captures the consistent underlying structure across rotational viewpoints.

To learn quasi rotation-invariant representations, we apply contrastive learning using the NT-Xent loss with $\tau = 0.1$. The hash and MLP portions of the latent representation (excluding [POS] tokens) are projected through a dedicated head $\mathbf{p}_\theta$:

$$\begin{aligned} \mathcal{L}_{\text{NTX}} &= \text{NT-Xent}(\mathbf{p}_\theta(\mathbf{z}^{\text{orig}}_{\text{content}}), \mathbf{p}_\theta(\mathbf{z}^{\text{aug}}_{\text{content}})) \\ \text{NT-Xent} &= -\log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \end{aligned} \tag{4}$$

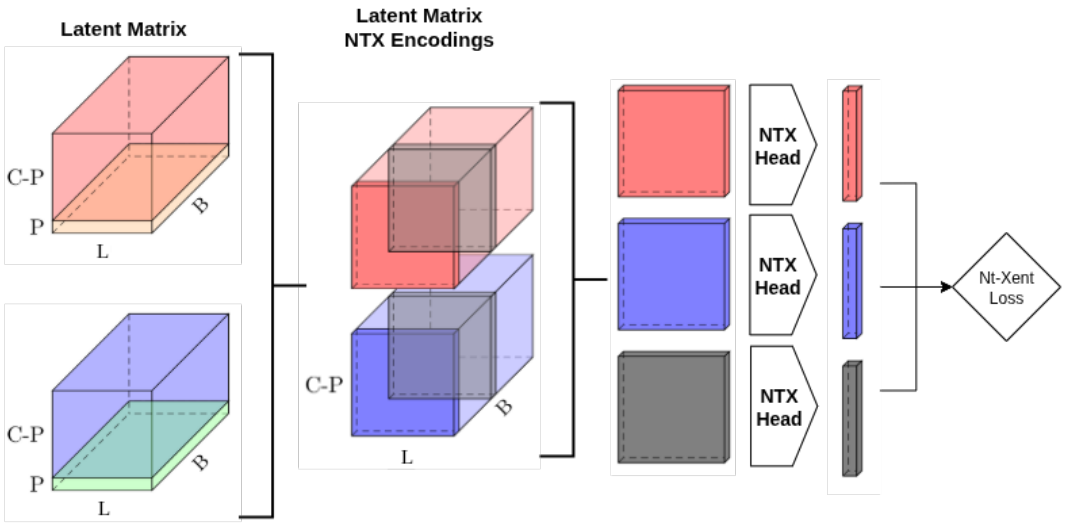


Figure 7: Contrastive learning pathway using NT-Xent loss on content representations. First the [POS] part of the latent matrix is removed. Then each object's latent representation is individually passed through a NTX head (see section 5) before finally passed to NT-Xent loss.

4.5.3 Rotation Prediction Loss

The [POS] token representations from both original and augmented views are concatenated and passed through a rotation head $\mathbf{r}_\theta$ to predict the relative rotation as a quaternion:

$$\begin{aligned} \mathbf{q}_{\text{pred}} &= \mathbf{r}_\theta([\mathbf{z}_{\text{pos}}^{\text{orig}}; \mathbf{z}_{\text{pos}}^{\text{aug}}]) \\ \mathbf{q}_{\text{true}} &= \mathbf{q}_{\text{aug}} \times \mathbf{q}_{\text{orig}}^{-1} \\ \mathcal{L}_{\text{rot}} &= \left(2 \cdot \arccos(|\mathbf{q}_{\text{true}} \cdot \mathbf{q}_{\text{pred}}|)\right)^2 \end{aligned} \tag{5}$$

where $\mathbf{q}_{\text{true}}$ represents the ground truth relative rotation quaternion. The loss measures the squared geodesic distance between predicted and true quaternions on the 3-sphere, providing a geometrically meaningful rotation error metric.

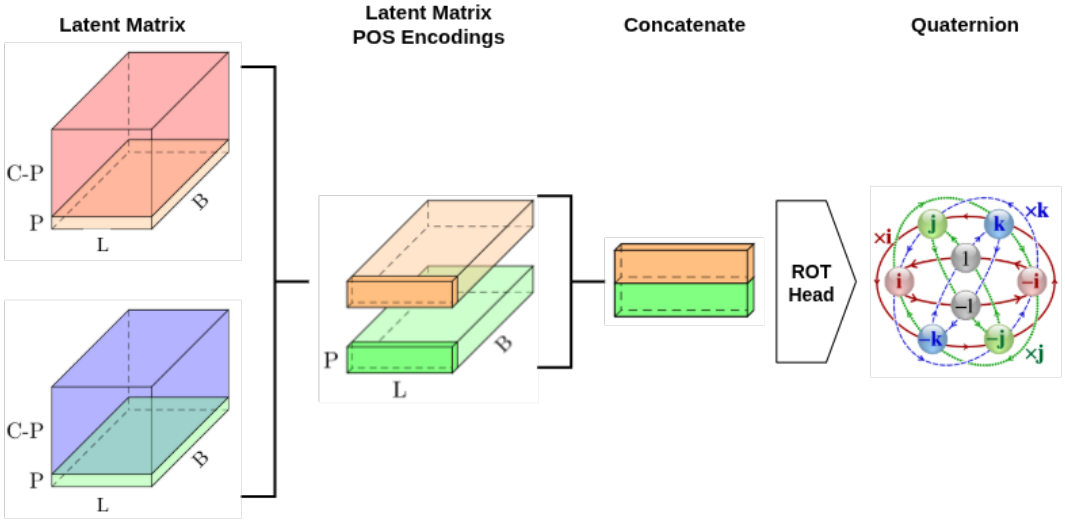


Figure 8: Rotation prediction pathway using [POS] token representations to predict relative quaternions. First the [POS] part of the original and augmented latent matrices are concatenated and fed through the rotation head, that predicts a 4-dimensional vector.

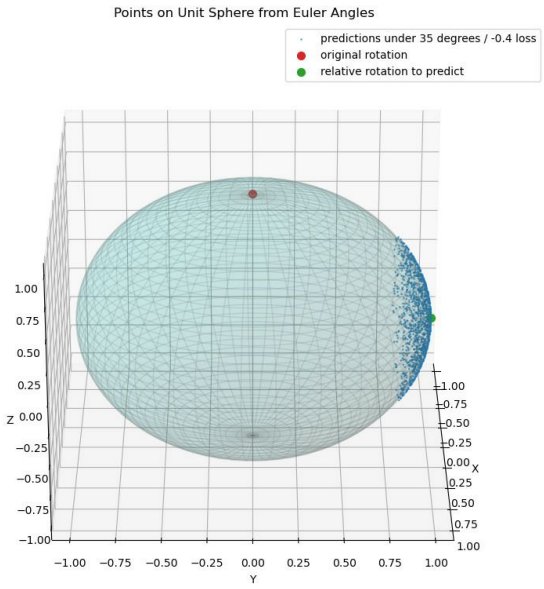


Figure 9: Visualization of quaternion loss on the 3-sphere: Given original (red) and augmented (green) rotations, blue points show random samples where the angular distance between $\mathbf{q}_{\text{pred}}$ and $\mathbf{q}_{\text{true}}$ is less than 30 degrees. By decreasing this angular degree, we are basically forcing the blue points in a smaller and smaller circle around the augmented (green) point

4.5.4 Combined Objective

Moreover, we also add a L2 regularization on the latent vector as a component of our loss function (see section 5). The final training objective combines all losses with learned regularization on the latent representations:

$$\mathcal{L} = \gamma_1 \mathcal{L}_{\text{recon}} + \gamma_2 \mathcal{L}_{\text{NTX}} + \gamma_3 \mathcal{L}_{\text{rot}} + \gamma_4 \|\mathbf{Z}\|_2^2 \tag{6}$$

where $\gamma = [0.4, 0.05, 0.53, 0.02]$ are the loss weights determined empirically to balance the different objectives.

5 Results and Discussion

5.1 Experimental Setup

We train our model using the Adam optimizer with a learning rate of 10^{-4} and a batch size of 32. The transformer architecture consists of 4 attention heads and 4 layers, with an embedding dimension of 1024 and a latent dimension of 128. Training is conducted for 500 epochs, with aggressive regularization including 0.4 dropout rates and 0.01 weight decay to combat overfitting on our limited dataset.

The loss weights are set to $\gamma = [0.4, 0.05, 0.53, 0.02]$ for reconstruction, NT-Xent, rotation, and L2 regularization losses respectively.

5.2 Results

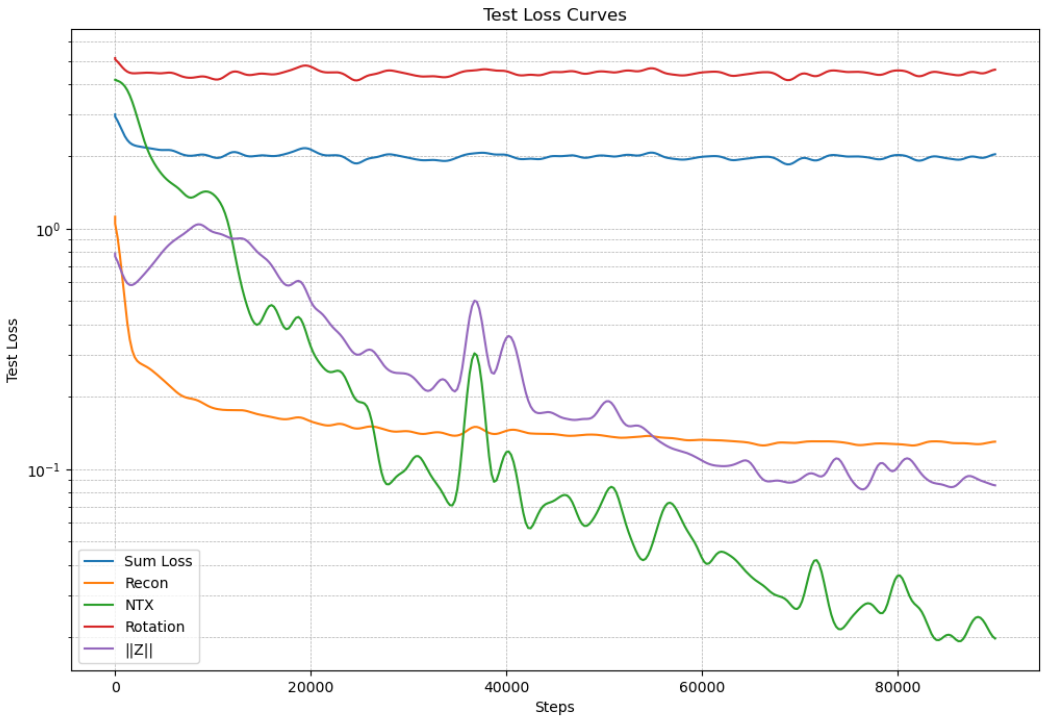


Figure 10: Loss curves on test dataset. See Appendix 6 for hyperparameter configuration. All reconstruction, NTX and Z norm metrics converge, while the weighted sum loss stays flat due to the persistent rotational loss.

5.2.1 Reconstruction Performance

We achieve around 0.1 reconstruction loss on both test and train sets, which matches similar levels reported in Schürholt et al. [2024]. However, since the magnitude of loss depends heavily on the data distribution, a direct comparison is not entirely meaningful. The consistent reconstruction across train and test sets indicates that our tokenization strategy successfully preserves the essential information from both hash table and MLP components without overfitting.

5.2.2 Contrastive Learning Success

We achieve < 0.1 NTX Loss, demonstrating that the network can successfully push representations of different rotations of the same object close together while keeping different objects further apart. This is a crucial result that validates our core hypothesis about learning rotation-invariant object representations.

For the NTX head architecture, we experimented with two approaches: directly flattening the latent matrix $(C, D) \to (C \times D)$ and feeding it through a 2-layer MLP, versus first applying attention pooling $(C, D) \to (D)$ followed by the MLP. Both methods achieved similar levels of success, suggesting that the contrastive learning objective is robust to these architectural choices.

5.2.3 Sampling Strategy Impact

The sampling strategy for hash tokens significantly affects NTX learnability. We tested two approaches: sampling the exact same 380 points for both original and augmented views, versus super-sampling a larger set and sub-sampling individually (so views share approximately 1/4 of their points on average). Same-point sampling makes the NTX task much easier, which intuitively makes sense - since we encode "per token" representations rather than whole network representations. Sampling different points means the latent representations don't semantically match between views.

However, we couldn't definitively confirm whether the network simply learns to overfit on positional encodings, since both original and augmented views receive identical positional encoding when using the same sampling points. This remains an open question requiring further investigation.

5.2.4 Latent Space Analysis

We track both $E[\mathbf{z}^2]$ and $E[(\mathbf{z}_{\text{ntx}}^{\text{orig}} - \mathbf{z}_{\text{ntx}}^{\text{aug}})^2]$ to understand the latent space dynamics. The distance between original and augmented representations decreases as the regularization reduces $E[\mathbf{z}^2]$. Interestingly, while at initialization $E[\mathbf{z}^2] > E[(\mathbf{z}_{\text{ntx}}^{\text{orig}} - \mathbf{z}_{\text{ntx}}^{\text{aug}})^2]$, this relationship reverses over time and both values converge to similar magnitudes.

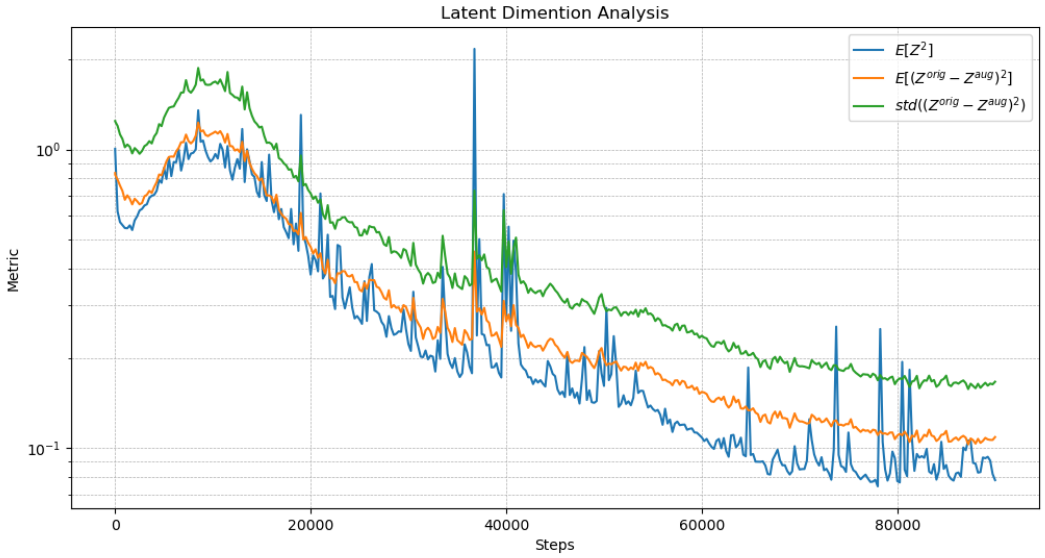
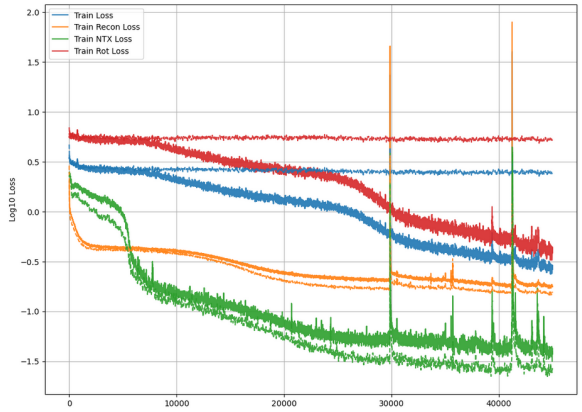


Figure 11: Latent dimension analysis showing metrics for $E[\mathbf{z}^2]$, $E[(\mathbf{z}_{\text{ntx}}^{\text{orig}} - \mathbf{z}_{\text{ntx}}^{\text{aug}})^2]$, and std$[(\mathbf{z}_{\text{ntx}}^{\text{orig}} - \mathbf{z}_{\text{ntx}}^{\text{aug}})^2]$.

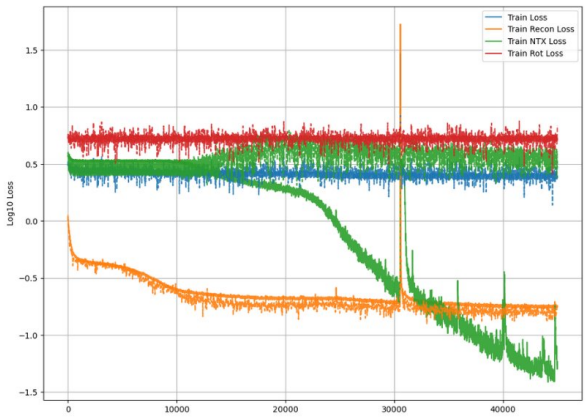
Since both NTX heads are non-linear mappings, there may not be a direct relationship between the latent representations and the distance metrics we observe. This warrants further investigation to understand the true dynamics of the learned embeddings.

5.2.5 Rotation Prediction Challenges

Despite successful reconstruction and contrastive learning, we have not been able to predict rotations on the test set to this point. This represents the most significant challenge in our multi-task learning framework.



(a) Test and train loss curves where the rotation task overfits severely.



(b) Test and train loss curves where NTX task overfits and rotation task underfits.

Figure 12: Visualization of loss curves from different training runs. Solid lines represent training loss while dashed lines of the same color represent test loss.

5.3 Discussion

5.3.1 Overfitting Challenges

Transformers are notoriously data-hungry models that easily overfit on low-capacity datasets. One of our biggest challenges was persistent overfitting, despite aggressive regularization including 0.3 dropout rates and 0.01 weight decay. Our dataset consists of only 342 objects with 6 discrete rotations each, without the means to generate new objects or continuous rotations. In this sparse sampling regime, rotation prediction becomes more like classification than regression, making generalization particularly difficult.

To combat overfitting, beyond high regularization (see Appendix 6), we heavily augmented our dataset by adding multiplicative noise and permuting MLP layers. The permutation problem, which initially seemed like a challenge, turned out to be an effective way of synthetically expanding the dataset.

5.3.2 L2 Regularization as Disentanglement

Interestingly, regularizing the latent matrix proved to be another effective approach for combating overfitting. The intuition comes from our goal of learning disentangled representations Wang et al. [2024]. Ideally, we want rotation-specific information encoded in the [POS] token portion and object-specific information in the remaining latent matrix. Beta-VAE approaches achieve this through KL divergence loss and reparameterization tricks, effectively regularizing the latent space in a statistically principled way. L2 regularization of the latent matrix has a similar effect - it can be thought of as a deterministic version of KL divergence that encourages sparsity and structure in the learned representations.

5.4 Future Work

Our current architecture inputs a tensor of shape $(C - P) \times 256$ and produces latent representations of shape $(C, 128)$. The number of [POS] tokens effectively provides extra space for encoding information in the latent matrix. It might be possible to reduce the latent dimension while increasing the number of [POS] tokens to more meaningfully decouple rotational and object-specific components.

The tokenization and positional encoding strategy is crucial, but we didn't have sufficient time to experiment with alternative approaches. For example, in our current implementation, positional encodings are simply summed, so we don't explicitly differentiate between different types of information by projecting them to subsets of the embedding dimension. The original SANE implementation Schürholt et al. [2024] used embeddings to lower dimensions for positional encodings and concatenated them, explicitly signaling to the network that these information types are distinct.

Additionally, exploring continuous rotation generation rather than discrete sampling, investigating alternative hash sampling strategies that better preserve high-frequency information, and developing more sophisticated architectural components for rotation prediction remain important directions for future work.

References

Konstantin Schürholt, Michael W. Mahoney, and Damian Borth. Towards scalable and versatile weight space learning, 2024. URL https://arxiv.org/abs/2406.09997.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4), July 2022. ISSN 0730-0301. doi:10.1145/3528223.3530127. URL https://doi.org/10.1145/3528223.3530127.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: representing scenes as neural radiance fields for view synthesis. *Commun. ACM*, 65(1):99–106, December 2021. ISSN 0001-0782. doi:10.1145/3503250. URL https://doi.org/10.1145/3503250.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets, 2018. URL https://arxiv.org/abs/1712.09913.

Xin Wang, Hong Chen, Si'ao Tang, Zihao Wu, and Wenwu Zhu. Disentangled representation learning, 2024. URL https://arxiv.org/abs/2211.11695.

6 Appendix

Table 1: Hyperparameter Configuration

Parameter Category		
Parameter	Description	Value
Architecture Parameters		
Input Dimension	Token feature dimension	256
Embedding Dimension	Transformer embedding size	1024
Latent Dimension	Compressed representation size	128
Number of Heads	Multi-head attention heads	4
Number of Layers	Transformer encoder/decoder layers	4
Projection Dimension	NTX head output dimension	128
Dropout	Regularization dropout rate	0.3
Bias	Linear layer bias terms	False
Sequence Configuration		
Hash Token Size	Sampled 3D points for hash tables	380
MLP Token Size	Tokenized MLP parameters	53
POS Token Size	Rotation-specific tokens	10
Total Sequence Length	Complete input sequence	443
Training Parameters		
Batch Size	Training batch size	32
Learning Rate	Initial learning rate	1×10^{-4}
Total Epochs	Training duration	20
Weight Decay	L2 regularization coefficient	1×10^{-3}
Gradient Clipping	Maximum gradient norm	5.0
Scheduler	Learning rate schedule	OneCycleLR
Loss Configuration		
γ_1 (Reconstruction)	Reconstruction loss weight	0.4
γ_2 (NT-Xent)	Contrastive loss weight	0.05
γ_3 (Rotation)	Rotation prediction weight	0.53
γ_4 (L2 Reg.)	Latent regularization weight	0.02
NT-Xent Temperature	Contrastive learning temperature	0.1
Augmentation Parameters		
Noise Augmentation	Multiplicative noise std	0.1
Permutation Augment	MLP layer permutation	True

6.0.1 Autoencoder Architecture

Our autoencoder employs a symmetric encoder-decoder architecture built from transformer layers. The encoder processes the tokenized input through the following pipeline:

1. **Linear Tokenization**: Projects 256-dimensional input tokens to 1024-dimensional embeddings
2. **Positional Encoding**: Adds specialized positional information for hash, MLP, and POS tokens
3. **Transformer Encoder**: 4 layers of multi-head attention with 4 heads each, using GELU activation and layer normalization
4. **Latent Projection**: Final linear layer compresses to 128-dimensional latent space

The decoder mirrors this structure in reverse, reconstructing the original 256-dimensional tokens from the compressed latent representation. Both encoder and decoder layers use identical transformer configurations with $d_{\text{model}} = 1024$, feedforward dimension of 4096, and 0.3 dropout for regularization.

6.0.2 NT-Xent Projection Head

The contrastive learning head processes the content portion of the latent representation (excluding POS tokens) through a two-stage architecture:

1. **Attention Pooling**: Aggregates the sequence of latent tokens $(C-P, 128)$ into a single fixed-size representation (128) using learned attention weights
2. **Projection MLP**: Two-layer network that maps from 128 to 64 dimensions via [128 $\rightarrow$ 64 $\rightarrow$ 32] with ReLU activation

This design allows the model to adaptively weight different tokens when creating the object representation for contrastive learning, rather than using simple averaging or concatenation. The attention pooling mechanism can learn to focus on the most discriminative tokens for object identification while ignoring rotation-specific information.

6.0.3 Rotation Prediction Head

The rotation prediction head operates exclusively on the POS token portion of the latent representations and employs a specialized architecture for quaternion prediction:

1. **Input Concatenation**: POS tokens from both original and augmented views are concatenated along the sequence dimension, creating input shape $(20, 128)$
2. **Learned Positional Encoding**: Adds unique positional embeddings for each of the 20 token positions to distinguish between original and augmented view tokens
3. **Transformer Processing**: Single transformer encoder layer with 2 attention heads processes the concatenated sequence
4. **Token Selection**: Extracts only the first token output, which serves as the aggregated rotation representation
5. **Quaternion Prediction**: Final linear layer maps from 128 to 4 dimensions, producing the predicted relative rotation quaternion

The architecture explicitly separates rotation reasoning from content representation by using dedicated POS tokens, allowing the model to specialize different parts of the latent space for different tasks. The first-token extraction mechanism forces the transformer to aggregate rotation information into a single representative token.