Generative Modeling of Weights: Generalization or Memorization?

Appendix

Table of Contents

A. Experimental Setup for Each Method

In this paper, we analyze four representative generative modeling methods for neural network weights, under their primary experimental setups. Here, we expand on Section 2.1 to provide more details about the training, inference, and analysis of the four methods.

Hyper-Representations provides, for each of MNIST, SVHN, CIFAR-10, and STL-10, a dataset of thousands of classification model checkpoints, together with a pre-trained generative model trained on that checkpoint dataset. We evaluate the pre-trained Hyper-Representations autoencoder checkpoint for the SVHN classification model weights. Unless explicitly stated otherwise, the training weights referenced throughout the paper refer to the reconstructions of the original training weights produced by the Hyper-Representations autoencoder. As noted in Figure 3, the reconstructed weights have lower accuracies than the original training weights.

Hyper-Representations save one checkpoint at each of epochs 21 to 25 in every classification model training run when creating its datasets of checkpoints. However, when calculating the L_2 distance from training and generated checkpoints to their nearest training checkpoints, we only use training checkpoints from the 25th epoch. This is to ensure that the distances between training checkpoints correctly represent distances between models independently trained from scratch.

G.pt provides datasets of 2.1M to 11.3M checkpoints and pre-trained generative models of weights for MNIST and CIFAR-10 classification models, as well as Cartpole reinforcement learning models. We evaluate the pre-trained generative model for MNIST classification model weights.

Although G.pt's training procedure uses 200 checkpoints from each MNIST training run, to reduce computational cost, we use only the final checkpoint from each run as the training weights throughout our paper. This would not underestimate memorization, since the one-step generation of G.pt explicitly prompts the generative model to produce MNIST classification model weights with zero test loss, making the generated weights more similar to final checkpoints than to earlier ones.

HyperDiffusion trains an unconditional diffusion model on MLPs that each represent a neural occupancy field of a unique 3D shape. These MLPs are trained to map 3D coordinates to occupancy values. Meshes can be extracted from the MLPs using Marching Cubes [37].

HyperDiffusion is applied to MLPs trained on the car, chair, and airplane categories of ShapeNet [6], as well as to MLPs representing 4D animation sequences. However, the dataset of MLP checkpoints and the corresponding generative model of MLP weights were only released for airplane shapes. Thus, our analysis is focused only on this experimental setting.

P-diff is applied to multiple image classification datasets and model architectures. However, the analysis in the original paper focuses on the setting of generating the last two batch normalization layers for a CIFAR-100 classification model of ResNet-18 architecture. Accordingly, we evaluate p-diff under this setting. Since no weight datasets or pre-trained generative models are released, we follow the official codebase to collect the training weights and train the generative model ourselves.

B. Results on Non-Parametric Test for Data-Copying Detection

Meehan et al. [42] proposed a non-parametric test for detecting memorization in generative models. The test requires the training set, a holdout set P_n of size n, and a set of generated samples Q_m of size m. The $n+m$ samples in $P_n \cup Q_m$ are sorted by their distance to the nearest training data point, with rank 1 assigned to the closest sample and rank $n+m$ to the farthest. Let R_{Q_m} denote the sum of the ranks of all generated samples Q_m. The U statistic [41] is

$$U_{Q_m} = R_{Q_m} - \frac{m(m+1)}{2}, \tag{1}$$

which is then normalized to obtain the z-scored statistic Z_U:

$$Z_U = \frac{U_{Q_m} - \mu_U}{\sigma_U}, \tag{2}$$

where $\mu_U = \frac{mn}{2}$ and $\sigma_U = \sqrt{\frac{mn(m+n+1)}{12}}$. Intuitively, $Z_U \ll 0$ indicates memorization (data-copying), while $Z_U \gg 0$ suggests that the generative model underfits the training data.

We apply the Z_U metric to the original generative models of weights from the four studied methods, as well as to the models trained in our ablation experiments. This provides additional quantitative evidence regarding whether the models memorize or generalize. Since each method provides a different number of holdout samples, the Z_U values are not directly comparable across methods.

B.1. Data-Copying Test for Original Models

In Section 3.1, our visualization showed that, except for p-diff, whose training checkpoints are saved consecutively in the same run and are of very low diversity, the generated checkpoints of all other methods are much closer to the training checkpoints than the training checkpoints are to one another.

Here, we measure the Z_U score based on the same L_2 distance metric between checkpoints for Hyper-Representations, G.pt, and HyperDiffusion, and find that the Z_U (-13.6, -8.5, and -30.8, respectively) scores are significantly smaller than 0, indicating severe memorization. This confirms the results in Section 3.1.

B.2. Data-Copying Test for Models from Ablation Experiments

In Section 4.1, we identified limited dataset size and model overparameterization as potential reasons why memorization occurs. Specifically, we showed that scaling up the training data reduces memorization in G.pt, and HyperDiffusion is capable of memorizing random weights. Here, we further confirm these trends using the Z_U score.

Scaling data for G.pt. We measure the Z_U score before and after increasing the training dataset size of G.pt from 2.1M to 20.4M samples. The Z_U score rises from -8.5 to 3.5, indicating that scaling data effectively mitigates memorization in G.pt.

training dataset size	2.1M	20.4M
mean dist b/w train & nearest train	3.64	2.70
mean dist b/w gen & nearest train	2.25	2.78
Z_U score [42]	-8.5	3.5

Table 2. **Scaling up training data can reduce memorization in G.pt**. The increase to positive Z_U score after scaling up data confirms the reduced memorization observed in Figure 9.

Training HyperDiffusion on random weights. In Section 4.1, we trained HyperDiffusion on the original dataset of checkpoints with one or all layers randomly reinitialized. We report the Z_U score of the original and newly-trained HyperDiffusion models in Table 3.

We find that all models have a Z_U score of -30.8, the lowest possible score when $n = 606$ and $m = 666$. This is because, in every case, each generated checkpoint is closer to a training checkpoint than any training checkpoint is to another training checkpoint. This further confirms that HyperDiffusion closely memorizes the training checkpoints without capturing meaningful patterns.

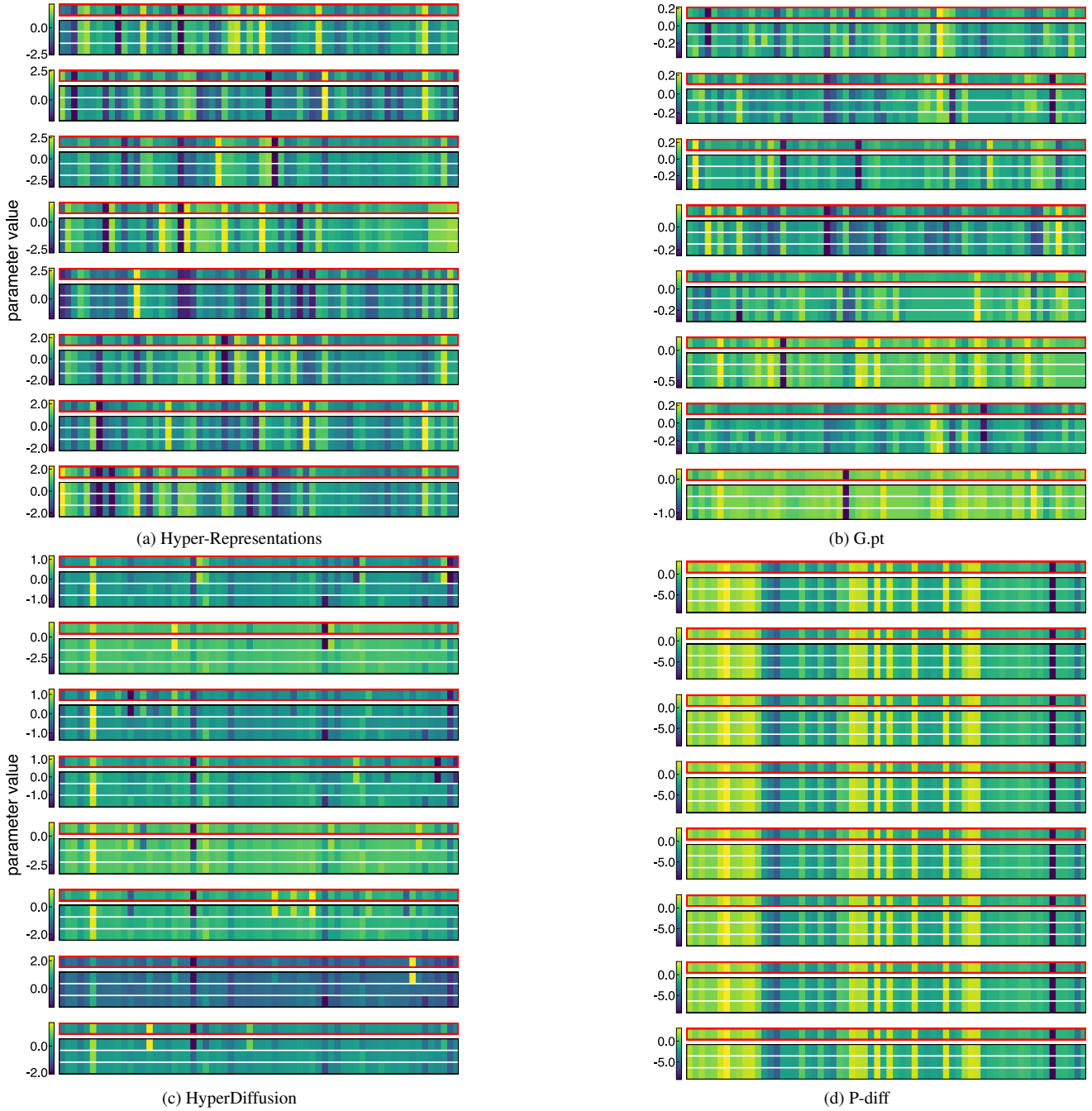
layers reinitialized in trained MLPs	none	1st	2nd	3rd	4th	all
mean dist b/w train & nearest train	109.5	114.0	90.3	106.0	126.4	16.0
mean dist b/w gen & nearest train	7.0	9.0	1.5	2.6	3.6	0.8
Z_U score [42]	-30.8	-30.8	-30.8	-30.8	-30.8	-30.8

Table 3. **HyperDiffusion memorizes training weights without learning their patterns**. The consistently low Z_U score confirms that severe memorization occurs in all cases, as observed in Table 1.

C. Additional Visualization of Model Weights and Behaviors

C.1. Additional Random Examples of Weight Heatmaps

In Section 3.1, we showed the values of random parameters in generated checkpoints and their nearest training checkpoints for all four methods, to demonstrate the memorization in weight space. Due to space constraints, we presented only three random examples per method. In Figure 13, we provide heatmap visualizations of eight additional random generated checkpoints and their nearest training checkpoints for each method. Consistently, we observe that for almost every generated checkpoint, there exists at least one training checkpoint with highly similar weights.



(a) Hyper-Representations

(b) G.pt

(c) HyperDiffusion

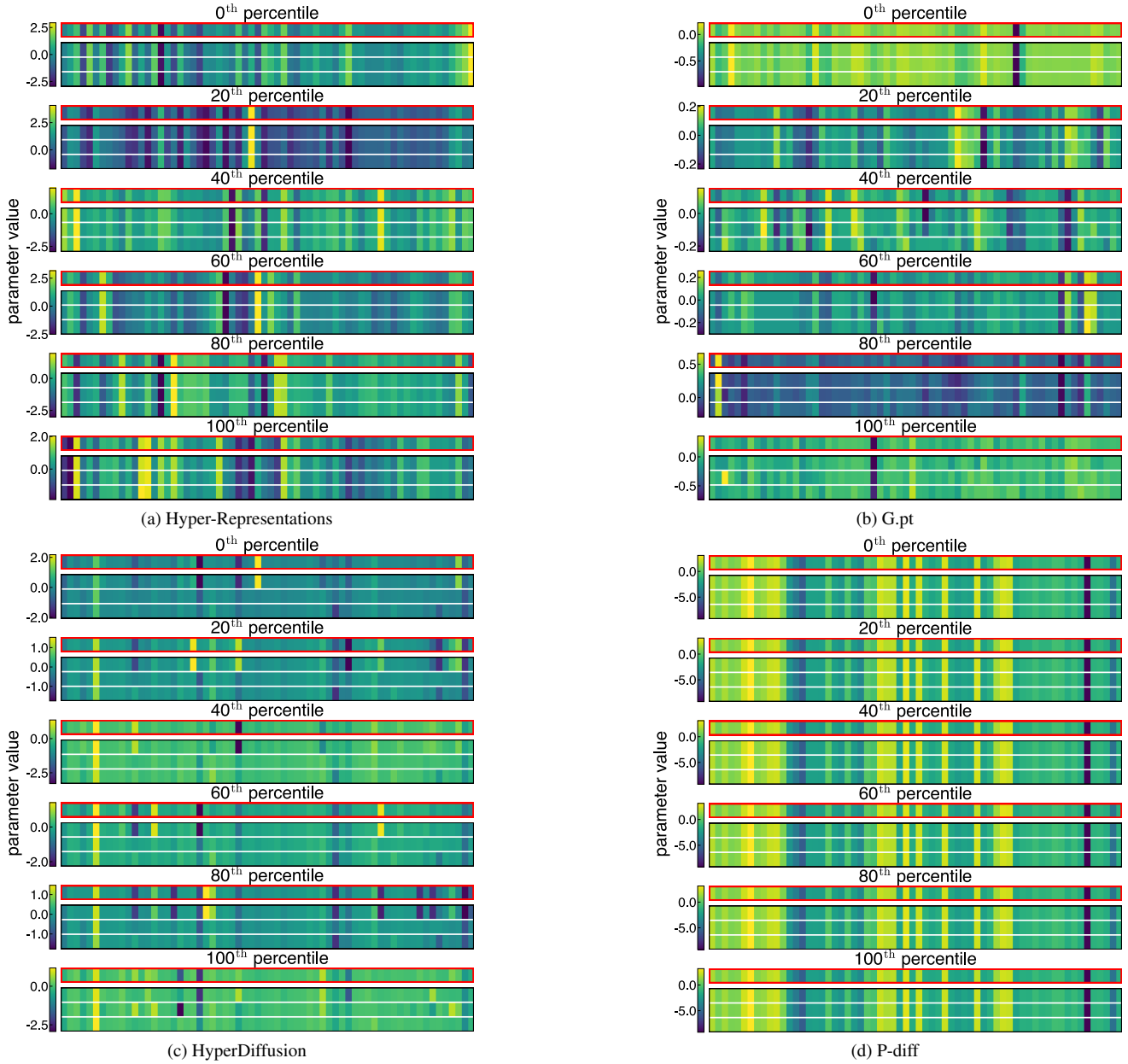
(d) P-diff

Figure 13. **Additional random examples of weight heatmap** for generated models and their nearest training models.

C.2. Heatmaps by Percentile of Distance to Nearest Training Checkpoint

In Section 3.1 and Appendix C.1, we showed weight heatmaps of random generated checkpoints and their nearest training checkpoints. Here, we further rank the generated checkpoints by their L_2 distance to the nearest training checkpoint and present weight heatmaps at different percentiles in Figure 14. A lower percentile corresponds to a smaller distance to the nearest training checkpoint.

Consistent with our earlier findings, the generated weights from Hyper-Representations are nearly identical to their nearest training weights across all percentiles. Similarly, for G.pt and HyperDiffusion, all generated checkpoints are highly similar to their nearest training checkpoints, except at the 100th percentile, which show moderate differences. For p-diff, across all percentiles, all training and generated checkpoints are nearly identical. As noted in Section 3, this is likely because its training checkpoints are saved from consecutive steps within the same training run, resulting in low diversity.



(a) Hyper-Representations

(b) G.pt

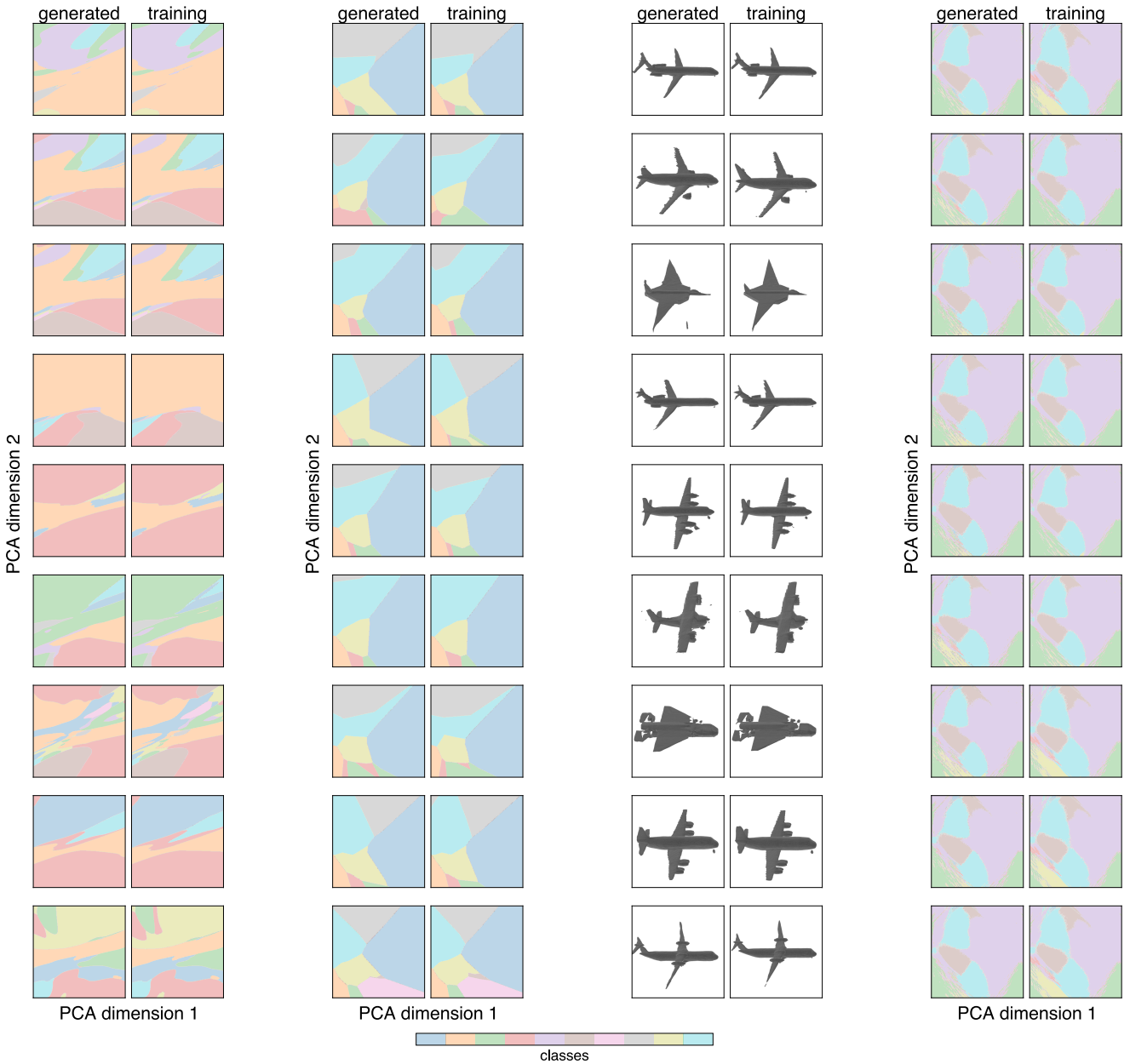
(c) HyperDiffusion

(d) P-diff

Figure 14. **Heatmaps of generated checkpoints at different percentiles of distance to the nearest training checkpoint**. Results are consistent with those observed for random generated weights: all generated checkpoints closely resemble their nearest training checkpoints, except for those at the 100th percentile in G.pt and HyperDiffusion, which show moderate differences.

C.3. Additional Random Examples of Model Outputs

In Section 3.2, we visualized the decision boundaries or reconstructed 3D shapes of generated models and their nearest training models, to demonstrate their high similarity in model behaviors. Due to space constraints, we presented only three random examples per method. In Figure 15, we provide visualizations of model outputs for nine additional random generated checkpoints per method and the nearest training checkpoint to each of them. These results further confirm that the generated checkpoints closely resemble their nearest training checkpoints not only in weight space but also in model behavior.



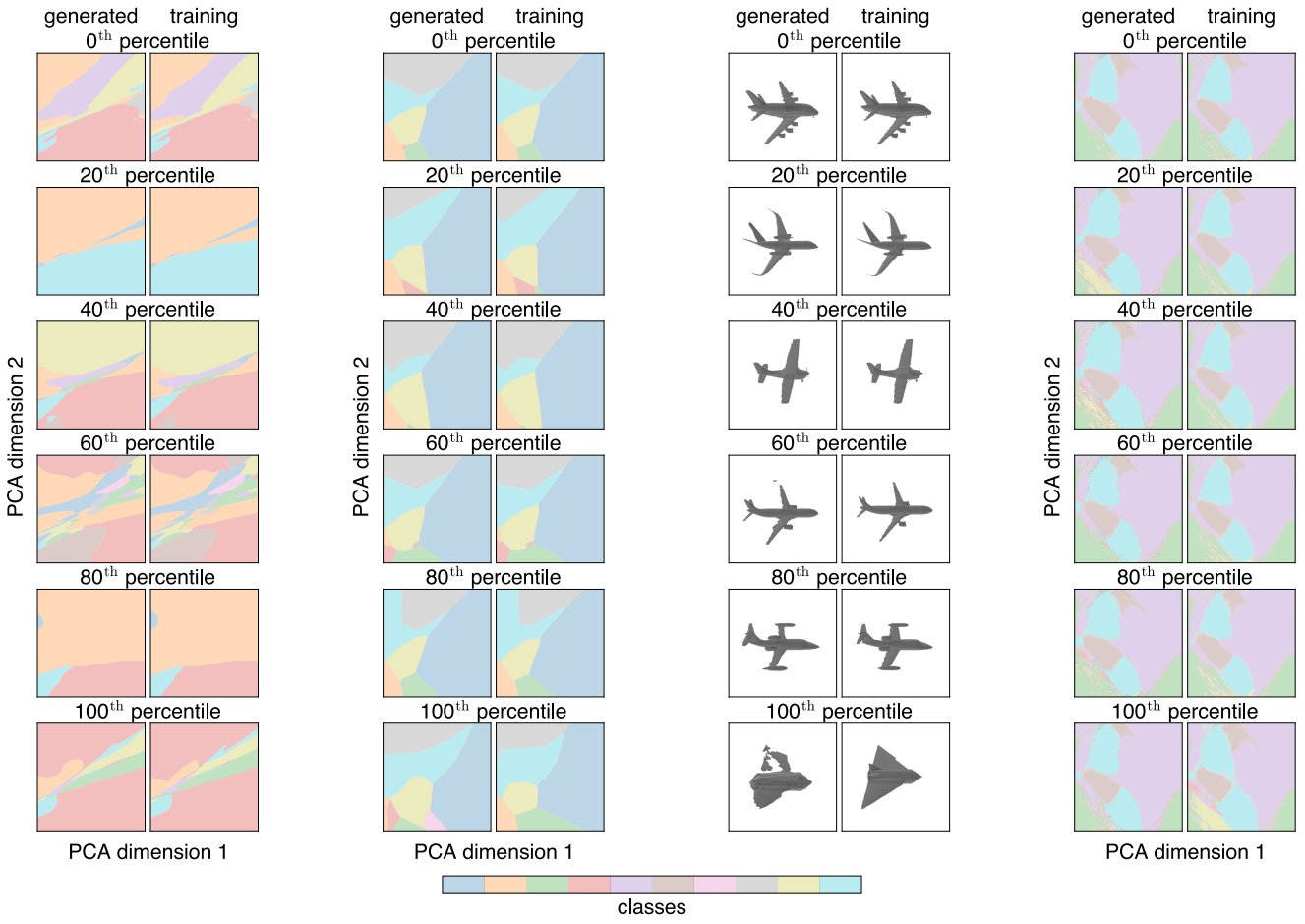
(a) Hyper-Representations (b) G.pt (c) HyperDiffusion (d) P-diff

Figure 15. **Additional random examples of decision boundaries and reconstructed meshes** of generated models and their nearest training models.

C.4. Model Outputs by Percentile of Distance to Nearest Training Checkpoint

In Section 3.2 and Appendix C.3, we showed model outputs from random generated checkpoints and their nearest training checkpoints. Here, we further rank the generated checkpoints by their L_2 distance to the nearest training checkpoint and present model outputs at different percentiles in Figure 16. A lower percentile corresponds to a smaller distance to the nearest training checkpoint.

Consistent with our earlier findings, the behaviors of generated models from Hyper-Representations are nearly identical to their nearest training weights across all percentiles. Similarly, for G.pt and HyperDiffusion, all generated checkpoints produce outputs highly similar to their nearest training checkpoints, except those at the 100th percentile, which show moderate differences. We note that the HyperDiffusion-generated checkpoint at the 100th percentile is of low quality (as seen in the degraded shape it reconstructs to) and thus cannot be matched to any training checkpoint. For p-diff, across all percentiles, all training and generated checkpoints' outputs are highly similar.



(a) Hyper-Representations (b) G.pt (c) HyperDiffusion (d) P-diff

Figure 16. **Decision boundaries and reconstructed meshes of generated checkpoints at different percentiles of distance to the nearest training checkpoint**. Results are consistent with those observed for random generated weights: all generated checkpoints closely resemble their nearest training checkpoints in model outputs, except at the 100th percentile in G.pt and HyperDiffusion, where the lower quality of the generated checkpoints may account for the observed differences.

D. Additional Information on Memorization in Model Behaviors

D.1. Metric for Checkpoint Similarity Based on Incorrect Predictions

In Section 3.2, we used the metric from Wang et al. [68] to quantify the similarity between two classification model checkpoints. The metric measures similarity based on the Intersection over Union (IoU) of the sets of incorrect predictions made by the two model checkpoints. Formally, it is defined as follows:

$$\begin{aligned} I_1 &= \{\, i \in \{1, \ldots, N\} \mid M_1(X_i) \neq y_i \}, \\ I_2 &= \{\, i \in \{1, \ldots, N\} \mid M_2(X_i) \neq y_i \}, \\ \mathrm{IoU}(M_1, M_2) &= \frac{|I_1 \cap I_2|}{|I_1 \cup I_2|}, \end{aligned} \tag{3}$$

where $\{(X_i, y_i)\}_{i=1}^{N}$ represents the test set on which the model checkpoints are evaluated. The sets I_1 and I_2 contain the indices of test samples for which model checkpoints M_1 and M_2 make incorrect predictions, respectively.

D.2. Additional Information on the Noise-Addition Baseline

In Section 3.2, we introduced a noise-addition baseline to compare with the generated models in terms of performance and novelty. For Hyper-Representations, whose KDE sampling method is based on the top 30% highest-accuracy training checkpoints, we apply noise to reconstructions of a random subset of these highest-accuracy checkpoints to ensure a fair comparison. For all other methods, we apply noise to checkpoints uniformly sampled from all training checkpoints.

D.3. Alternative Similarity Metric: Overlap in Classification Predictions

For classification models, the percentage of test set predictions they agree on provides an intuitive measure of their similarity in behavior. Table 4 shows the prediction overlap between classification model weights generated by Hyper-Representations, G.pt, and P-diff and their nearest training checkpoints under L_2 distance, along with prediction overlap between training models and their nearest neighbors (excluding self-comparisons) for comparison. As in Section 3, for Hyper-Representations, we use reconstructed training weights rather than the original ones.

method	Hyper-Representations	G.pt	P-diff
mean accuracy of training models	51.3	94.5	76.9
pred overlap b/w training & nearest training	75.6	97.9	91.4
pred overlap b/w generated & nearest training	98.5	98.2	93.5

Table 4. **Classification predictions highly overlap between generated and training models**. This shows that the generated models highly resemble the behaviors of training models.

Across all methods, generated models show higher prediction overlap with their nearest training models than training models do. This high overlap suggests that the generated models closely resemble the training models in behavior. However, we note that prediction overlap can be strongly influenced by accuracy: two models with accuracy x will have a minimum overlap of $\max(2x - 1,\ 0)$.

E. Additional Information on the Novelty of P-diff's Generated Models

All training and generated checkpoints of p-diff exhibit highly similar weight values (Figures 2 and 13) and decision boundaries (Figures 5 and 15). Yet, unlike other methods, p-diff achieves a better accuracy-novelty trade-off than noise addition (Figure 6), and its generated models are often farther from the nearest training model than training models are from one another (Figure 4).

This may be explained by p-diff's training checkpoints being saved from consecutive steps in the same training run, which results in significantly lower diversity in training models, compared to other methods that sample checkpoints across different runs. Consequently, p-diff may be interpolating within a narrow region of the weight space, which still *appears* novel relative to its low-diversity training distribution.

To investigate this, we analyze the weight distribution of p-diff's training and generated checkpoints, in comparison with models trained from scratch using different random seeds.

E.1. Comparison with Models Trained from Scratch

We train 300 models from scratch using different random seeds, with the same architecture (ResNet-18) and downstream task (CIFAR-100) as p-diff. The training recipe, shown in Table 5, is tuned so that the final accuracies of these models (75.4% $\pm$ 0.3%) approximately match those of p-diff's training checkpoints (76.8% $\pm$ 0.2%).

config	value
optimizer	AdamW [38]
learning rate	5e-4
weight decay	5e-4
optimizer momentum	β_1, β_2=0.9, 0.999
batch size	128
learning rate schedule	cosine decay
training epochs	300
augmentation	`RandomResizedCrop` [65] & `RandAug` (9, 0.5) [11]

Table 5. **Training recipe** for CIFAR-100 classification models trained from scratch.

In Figure 17, we visualize the weights of 20 randomly selected checkpoints from each group: p-diff training checkpoints, p-diff generated checkpoints, and models trained from scratch. We sample the same 64 parameter indices across all models. Both the training and generated checkpoints from p-diff exhibit minimal variation, while the from-scratch models display substantially greater diversity in parameter values.

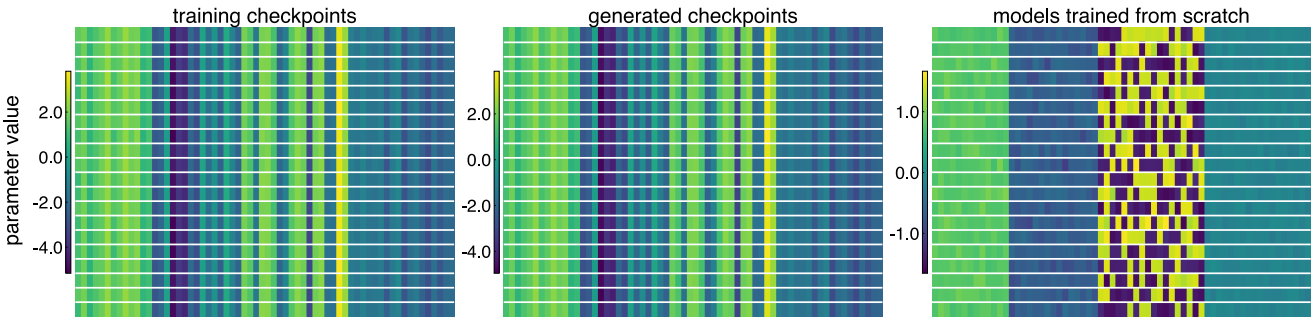


Figure 17. **P-diff's training and generated checkpoints show limited diversity compared to models trained from scratch**. Each row (separated by white lines) is a model checkpoint; each column is a randomly selected parameter index. The same indices are used across all three subplots.

We further quantify the weight space diversity of p-diff's training and generated checkpoints, compared to models trained from scratch. As shown in Table 6, both training and generated checkpoints of p-diff occupy a narrow region of the weight space, with low pairwise and nearest-neighbor distances. In contrast, models trained independently from scratch exhibit much higher weight variation.

case	mean L_2 distance
b/w all pairs of training checkpoints	6.9
b/w all pairs of training and generated checkpoints	6.3
b/w all pairs of from-scratch models	46.1
from training checkpoints to nearest training checkpoints	0.3
from generated checkpoints to nearest training checkpoints	5.4
from from-scratch models to nearest from-scratch models	43.1

Table 6. **Distances among p-diff's training and generated checkpoints are much smaller than the distances among from-scratch models**. This shows that p-diff's training and generated checkpoints occupy a narrow range in weight space compared to models trained from scratch.

These results confirm that p-diff's training and generated checkpoints occupy a highly constrained region in weight space, substantially narrower than the region spanned by independently trained models. Thus, although p-diff appears to interpolate between training checkpoints (Figures 7 and 8), the training checkpoints themselves lack diversity. As a result, the interpolation occurs within a narrow subspace and does not reflect meaningful generalization beyond the training data.

E.2. Additional Information on P-diff Weight Value Distribution

In Section 3.2, we showed that the parameter values in checkpoints generated by p-diff tend to center around the average of the parameter values in training checkpoints, using smoothed weight distribution curves. Figure 18 presents the same plot without smoothing, confirming that the moderate smoothing does not affect the observed trends.

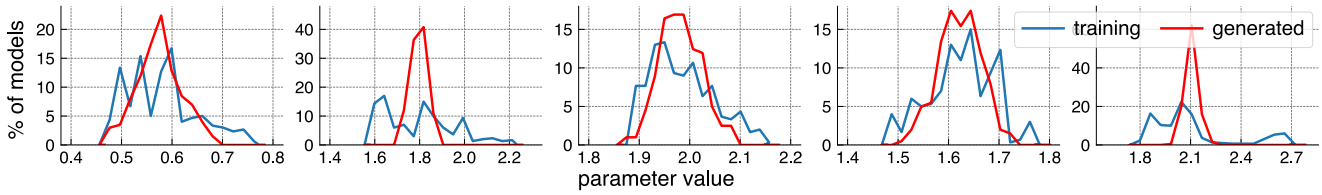


Figure 18. **Distributions of 5 randomly selected parameters** from the weight matrix of the first layer in the training and generated checkpoints of p-diff. This figure corresponds to Figure 7 but without smoothing applied to the distribution curves.

We further extend this analysis by visualizing the parameter value distributions of 50 randomly selected entries from the first-layer weight matrix in both training and generated checkpoints. As shown in Figure 19, the concentration of generated weights around the mean of training weights persists across this larger set of parameters.

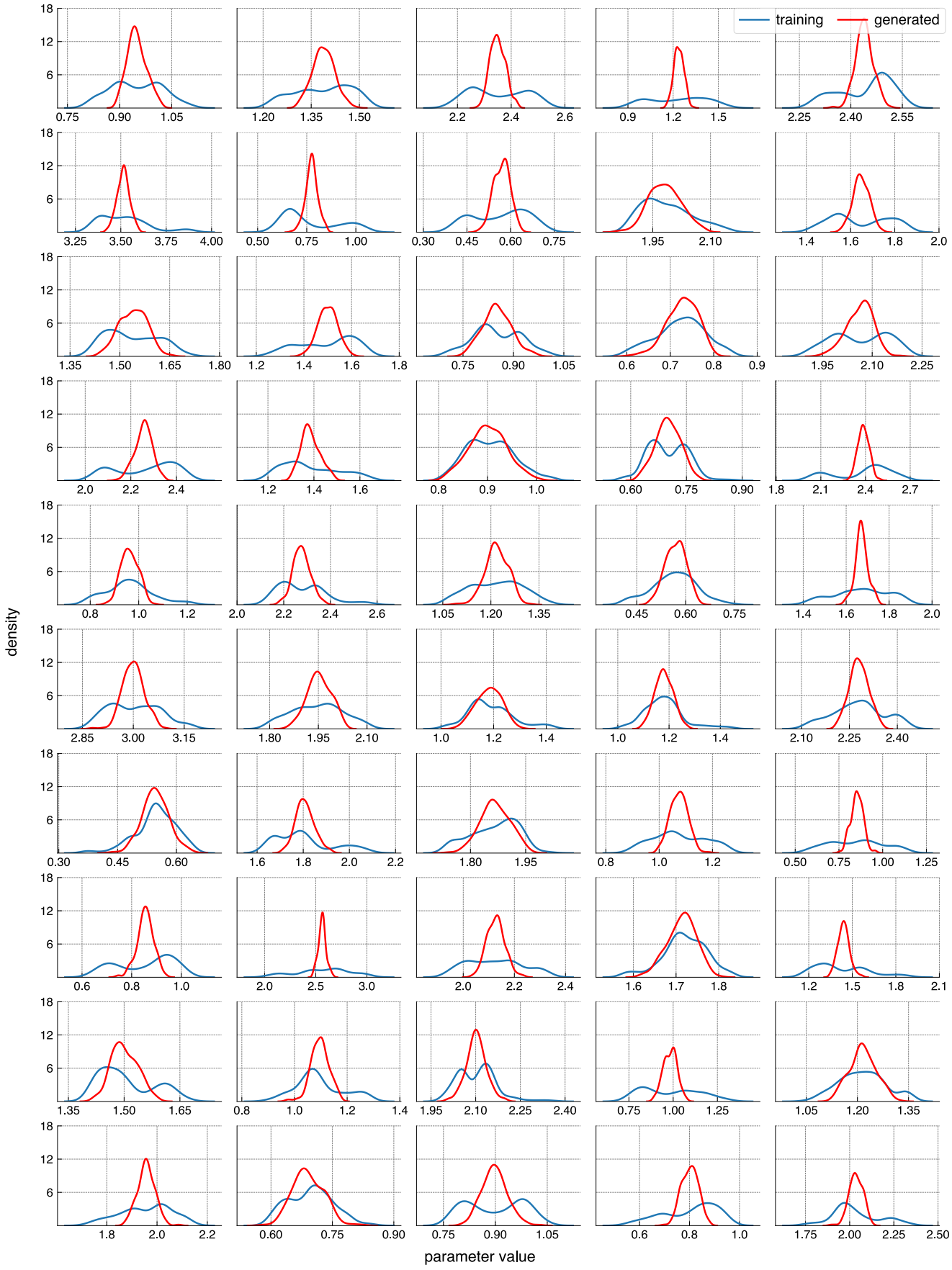


Figure 19. **Distributions of 50 randomly selected parameters** from the weight matrix of the first layer in the training and generated checkpoints of p-diff. This figure extends the analysis of Figure 7 to a broader set of parameters, further confirming the observed trend of generated weight values concentrating around the average of the training values.

F. Additional Information on the Impact of Training Configurations on Memorization

In Section 4.1, we demonstrated that limited data and overparameterized models likely contribute to memorization. Here, we extend this analysis by testing additional settings and modeling factors beyond data and model size. The Z_U score used in this section is introduced in Appendix B.

F.1. Data Scaling

In Section 4.1, we showed that scaling up training data for G.pt can effectively mitigate memorization. Here, we further explore scaling data for Hyper-Representations.

training dataset size	2896	24136
mean dist b/w train & nearest train	49.97	48.17
mean dist b/w gen & nearest train	8.11	9.24
Z_U score [42]	-13.6	-13.6

Table 7. **Scaling up training data does not reduce memorization in Hyper-Representations**. After scaling up, the distances between training and generated checkpoints remain much smaller than the distances among training checkpoints, and Z_U remains unchanged.

Concretely, we increased the number of training checkpoints from 2896 to 24136. However, as shown in Table 7, generated checkpoints remain far closer to training checkpoints than training checkpoints are to each other. The unchanged Z_U score after scaling further confirms that scaling data does not mitigate memorization in Hyper-Representations.

While this result does not rule out the possibility that scaling to much larger datasets might eventually reduce memorization, it suggests that there may be more fundamental modeling issues at play (*e.g.*, the lack of designs explicitly integrating the properties of weight data, as discussed in Section 4.2), beyond insufficient data.

F.2. Model Capacity

In Section 4.1, we showed that HyperDiffusion can fully memorize its training checkpoints, even when one or all layers are reinitialized with random weights, suggesting that it simply memorizes weights without capturing meaningful patterns. Here, we further demonstrate this by training HyperDiffusion on MLPs trained with different training lengths.

By default, HyperDiffusion trains the first MLP model from random initialization, and subsequent MLP models are initialized from the trained weights of the first model. To enable a fairer comparison across training lengths, we instead train all MLP models from scratch.

training length	0 epoch	200 epochs	400 epochs	full
mean dist b/w train & nearest train	16.0	115.7	129.4	126.8
mean dist b/w gen & nearest train	0.8	2.8	3.7	5.7
Z_U score [42]	-30.8	-30.8	-30.8	-30.8

Table 8. **HyperDiffusion memorizes MLP weights regardless of their training length**. This suggests that memorization occurs regardless of whether meaningful patterns exist in the weights.

Following HyperDiffusion's codebase, a full MLP training run ends when the training loss fails to improve for 50 consecutive epochs. Across 500 independent runs, the average training length is 667.7 epochs. We collect new datasets of checkpoints by training all MLPs for 200 and for 400 epochs, and then train a HyperDiffusion on each dataset. The evaluation results are shown in Table 8.

F.3. Other Modeling Factors

Modeling choices such as training duration, model architecture, and regularization strategy have been shown to significantly impact memorization in image diffusion models [18, 26, 62, 73]. In Section 4.1, we also showed that the large size of the generative models of weights likely contributes to memorization. Here, we investigate whether adjusting these factors suffices to mitigate the memorization in generative models of weights.

Quantitative metrics. In Section 3, we apply various metrics and baselines to demonstrate the memorization in weight space and model behaviors. Here, we measure the mean L_2 distance between generated models and their nearest training models, as a simple proxy to quantify the novelty of generated weights. However, a high L_2 distance to training weights may also arise from low-quality weight generations. Thus, we also evaluate model performance: accuracy for classification

	Hyper-Representations			G.pt			HyperDiffusion		
	#params	acc.↑	L_2 ↑	#params	acc.↑	L_2 ↑	#params	MMD↓	L_2 ↑
baseline									
training	–	65.2	50.0	–	94.4	3.64	–	0.026	109.5
default gen	223M	57.5	8.11	378M	94.0	2.25	1.4B	0.036	7.03
training epochs									
33.3%	223M	33.8	7.86	378M	94.0	2.40	1.4B	0.035	7.40
50.0%	223M	47.1	7.97	378M	94.0	2.25	1.4B	0.034	4.74
model size									
+dim & head	359M	50.1	8.06	579M	93.6	2.12	2.1B	0.034	2.69
+layer	362M	55.9	8.09	605M	93.9	2.08	2.0B	0.036	3.32
–dim & head	118M	44.1	7.93	220M	93.6	2.51	0.8B	0.039	22.47
–layer	154M	42.2	7.93	208M	86.6	3.70	1.0B	0.033	3.17
regularization									
+10% dropout	223M	53.9	7.76	378M	93.7	2.27	1.4B	0.035	5.10
+20% dropout	223M	44.7	7.26	378M	92.5	3.13	1.4B	0.034	6.08
+Gaussian noise	223M	57.8	8.14	378M	92.9	2.37	1.4B	0.033	3.24

Table 9. **Modeling changes do not effectively mitigate memorization**: modifications known to reduce memorization in image diffusion fail to meaningfully improve the novelty of generated weights (measured via L_2 to nearest training model) without degrading performance. The resulting changes in L_2 for generated weights are often much smaller than the gap in L_2 between the two baselines.

models and Minimum Matching Distance (under Chamfer Distance) for neural field models. These quantitative evaluations of generative models under varying modeling factors are shown in Table 9. We report the metrics for training weights and generated weights under default settings as baselines.

Training epochs. Reducing training epochs tends to lessen memorization in generative models [18, 62, 73]. We shorten training to 1/2 and 1/3 of the original length. Nonetheless, this has minimal impact on the L_2 distances and the quality of generated weights for G.pt and HyperDiffusion, while significantly degrading the accuracy of models produced from Hyper-Representations.

Model size. The size of a generative model can influence its sample quality and generalization [13, 44]. We vary the model size by increasing or decreasing its depth (*i.e.*, number of layers) and width (*i.e.*, model dimensions). However, across the three methods, changing the model size does not meaningfully increase the L_2 distances without compromising the generated models' performance.

Regularization. Regularization techniques have long been leveraged to prevent models from overfitting to the training set [50, 64, 66]. Here, we apply dropout [64] and inject random Gaussian noise into the training weights. Yet, these only result in minor changes to sample quality and L_2 distances.

Discussion. Modeling factor adjustments common in image diffusion cannot alleviate the memorization issue: none substantially improved the novelty of the generated weights without degrading performance. Notably, the changes in L_2 distance resulting from these variations were much smaller than the original gap between L_2 measured on training weights and on generated weights.

G. Additional Information on Permutation Augmentation for HyperDiffusion

In Section 4.2, we investigated whether adding permutation augmentations to the training data of HyperDiffusion reduces memorization. Specifically, we added 1, 3, and 7 random weight permutations during training, effectively enlarging the dataset by factors of $\times 2$, $\times 4$, and $\times 8$, respectively.

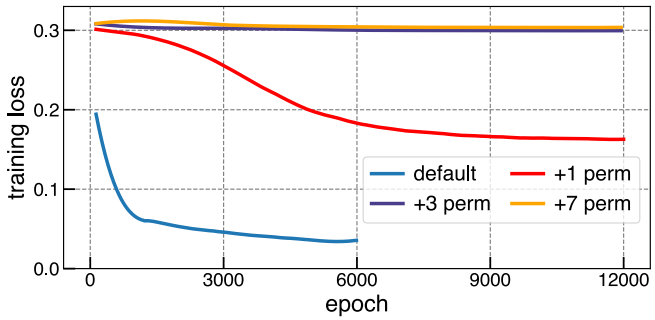


Figure 20. **HyperDiffusion fails to converge when three or more permutations are added**.

Figure 20 shows the corresponding training loss curves. We observe that when three or more permutations are applied, the model completely fails to converge. This aligns with the 3D shape visualizations in Figure 11, where the generated shapes do not represent any meaningful object.

H. System-Level Comparison between Image and Weight Generation Models

In our study, we show that current generative models for weights primarily memorize training data, drawing a system-level comparison with the generalization of image generation models. Here, we provide a example to illustrate the generalization behaviors of a common image generation model trained on the same amount of data as weight generation models. Concretely, we compare HyperDiffusion, an unconditional diffusion model for weight generation, with a standard Denoising Diffusion Probabilistic Model (DDPM) [23] trained on Flowers [45].

We randomly select 2749 images from the 20 largest classes in Flowers, to match the dataset size used for HyperDiffusion. We then train an unconditional DDPM on this dataset, as well as on two smaller subsets of 100 and 500 images, applying horizontal flipping as data augmentation. All models are trained at a resolution of 64×64 with a consistent training setup: 43K iterations, batch size 64, 500 warm-up steps, and a learning rate of 1e-4.

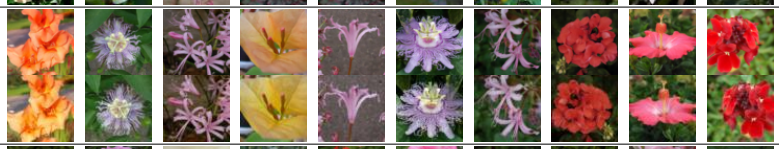
# training imgs	type	randomly sampled images
100	generated	
	training	
500	generated	
	training	
2749	generated	
	training	

Table 10. **Image diffusion models improve generalization and reduce memorization with more training data**. Each pair of consecutive rows shows randomly selected generated images alongside their most similar training images. When trained on 100 or 500 images, the model often replicates training samples or their horizontal flips—a data augmentation used during training. However, with 2749 training samples, the model generates novel images, demonstrating improved generalization.

After training the image diffusion models, we use the image copy detection method SSCD [51] to compute the similarity scores between generated and training images. Table 10 visualizes ten randomly selected generated images alongside their most similar training images. When trained on only 100 samples, the diffusion model primarily memorizes the training images, but with a larger dataset of 2749 images, it generalizes to produce novel outputs.

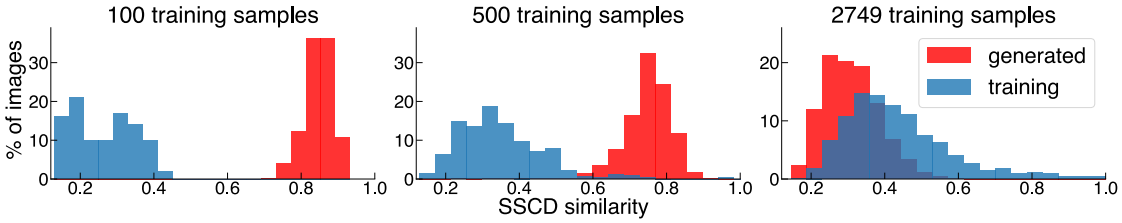


Figure 21. **Image diffusion models transition from memorization to generalization with more data**. The red histograms and blue curves show the distributions of SSCD similarity between each generated image and its most similar training image (red) and between each training image and its most similar training image (blue, excluding self-comparisons). As the training dataset grows, the red histograms shift left, indicating that the model generates increasingly distinct images rather than memorizing training samples.

Figure 21 presents the quantitative trend of similarity between generated images and their most similar training images as a function of dataset size. As a reference, we also show the similarity distribution between each training image and its most similar training image (excluding self-comparisons). We observe that with more data, the model generates images with a similarity level comparable to that between training images themselves. This *contrasts* with the trend observed for

HyperDiffusion in Section 3.1, where the model fails to generate novel weights even when trained on 2749 samples.

Discussion. Here, we provide an example illustrating that, on a system-level, a common image generation models can generalize well on the same amount of data used to train weight generation models. However, this difference can be attributed to various factors, including but not limited to the generative model size and architecture, diversity of dataset, data modality, and training recipe.

I. Impact of Downstream Dataset Diversity on Weight Memorization

For the four methods studied in this paper, their primary experimental setups use SVHN, MNIST, ShapeNet, and CIFAR-100, respectively. One may wonder whether the limited diversity of these downstream datasets leads to limited diversity in the dataset of checkpoints, and thereby indirectly contributes to the memorization in the generative models of weights. To test this, we train p-diff on a more diverse image dataset, ImageNet [12], following its official codebase.

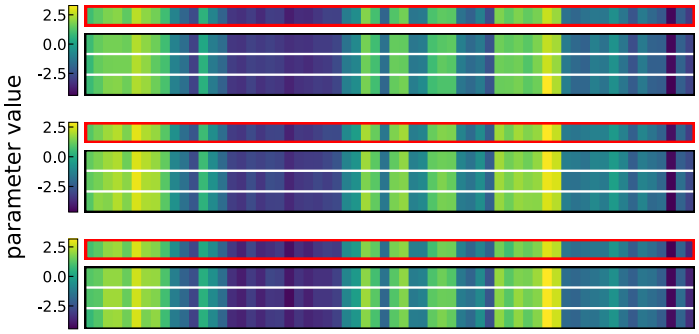


Figure 22. **P-diff's generated weights closely resemble training weights** when trained on ImageNet classification model checkpoints. The trend is consistent with the results on CIFAR-100 checkpoints (Figure 2).

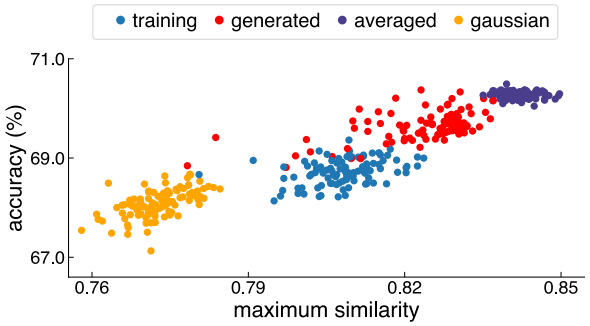


Figure 23. **P-diff does not outperform interpolation baselines** when trained on ImageNet classification model checkpoints, similar to the CIFAR-100 results (Figure 6).

Similar to the results on CIFAR-100 checkpoints (Figures 2 and 6), the generated weight values remain highly similar to training weights, and fail to outperform the interpolation baselines in the accuracy-novelty trade-off, as shown in Figures 22 and 23. In addition, 84.4% of generated weight values fall within one standard deviation around the mean of the training values, compared to 68.7% for training weights themselves. This suggests that the generated ImageNet classification model weights also tend to concentrate around the mean of the training weights.

Discussion. While the diversity of the downstream image dataset may indirectly influence the generalization of generative models of weights, our results show that increasing image diversity alone does not reduce memorization. Instead, modeling choices and the diversity of checkpoint datasets (*e.g.*, training a generative model on only 300 checkpoints saved from a single run) may be more fundamental issues.

J. Intrinsic Dimension of Image and Weight Data

J.1. Main Analysis

Unlike images, which are natural signals with high spatial redundancy, model weights have intricate dependencies between parameter groups. Weight data may exhibit higher complexity than images, potentially making it more challenging for generative models to capture their distribution, and limiting their ability to produce novel samples. To assess this complexity, we measure the *intrinsic dimensions* of weight and image data, which quantify the number of variables required to summarize high-dimensional data distributions.

Estimation method. We estimate the intrinsic dimensions using the Maximum Likelihood Estimator (MLE) [34, 40]. It can characterize data beyond simple linear structure as identified in alternative methods such as the Principal Component Analysis [47]. In essence, it estimates intrinsic dimension by modeling neighbor distributions with a Poisson process and computing the maximum likelihood intrinsic dimension from observed distances to neighbors. Formally, the estimator is formulated as

$$\bar{m}_k = \left[\frac{1}{n(k-1)} \sum_{i=1}^{n} \sum_{j=1}^{k-1} \log \frac{T_k(x_i)}{T_j(x_i)} \right]^{-1}, \tag{4}$$

where $\{x_i\}_{i=1}^n$ are the data points, $T_j(x_i)$ is the L_2 distance of x_i to its j-th nearest neighbor, and k is a hyperparameter that determines the number of nearest neighbors to consider.

MLE is shown to effectively capture the intrinsic dimensions of modern image datasets [53], but can be sensitive to the hyperparameter k (the number of nearest neighbors considered in the estimation). Thus, we report estimations for k = 3, 5, 10, 20, following Pope et al. [53].

Data. To compare the intrinsic dimensions of image and weight data, we use image datasets paired with the classification model weights trained on these datasets in Hyper-Representations.

Since the Maximum Likelihood Estimator requires the data to be independent and identically distributed (i.i.d.), we use only the weight checkpoint from the last epoch of each run to ensure that samples are i.i.d.. To align the image datasets we use with the datasets used to train the classification model checkpoints from Hyper-Representations, we resize all images to 28×28.

dataset	k = 3	k = 5	k = 10	k = 20
MNIST (image)	7	10	11	12
MNIST (weight)	56	79	86	85
SVHN (image)	8	13	16	17
SVHN (weight)	58	81	84	43
CIFAR-10 (image)	12	19	23	24
CIFAR-10 (weight)	62	89	99	100
STL-10 (image)	11	17	19	19
STL-10 (weight)	139	201	206	222

(a) raw data

dataset	k = 3	k = 5	k = 10	k = 20
MNIST (image)	10	14	17	18
MNIST (weight)	38	55	60	61
SVHN (image)	14	23	29	31
SVHN (weight)	45	55	49	37
CIFAR-10 (image)	19	33	42	44
CIFAR-10 (weight)	49	71	80	80
STL-10 (image)	21	33	37	36
STL-10 (weight)	58	81	89	88

(b) neural representations of data

Table 12. **MLE estimates weights to have higher intrinsic dimensions than images**, across different values of the hyperparameter k. We compute estimations for both raw data and their neural representations from an autoencoder. The estimations are rounded to integers.

Images and weights. Table 11a shows the intrinsic dimensions of image and weight datasets, measured with different values of hyperparameter k. We observe that, for all datasets and values of k, MLE consistently estimates much higher intrinsic dimensions for model weights than for images.

Neural representations. Aside from raw data, intrinsic dimension measures have also been used to inspect the neural representations of data [1, 72]. Here, we use the estimators to quantify the intrinsic dimensions required for neural networks to capture the image and weight distributions. Concretely, we extract latent representations from autoencoders trained on model weights and images. For weight data, we use the pre-trained autoencoder from Hyper-Representations [58, 59]. For image data, we train a separate autoencoder with the same architecture, latent dimensions, and objectives.

Table 11b presents the MLE estimates for these latents. Consistent with our observation on raw data, the neural representations of weights have higher intrinsic dimensions than those of images. Interestingly, the neural representations of images

have higher dimension estimates than raw images. This aligns with the "hunchback" pattern reported in prior work [1, 72], where intrinsic dimension is low at the input layer due to dominant yet redundant features in images, but peaks in middle layers.

Discussion. Our results suggest that weight data have higher intrinsic dimensions than images, both in raw forms and neural representations. Although prior theoretical work has identified a negative relationship between the intrinsic dimensionality of data and the generalization of diffusion models [8, 46], it is unclear whether the memorization in generative models of weights we observed is directly linked to the higher intrinsic dimensions of weight data.

J.2. Validating the Convergence and Performance of the Image Autoencoders

In Appendix J.1, we trained autoencoders on image datasets to compare the intrinsic dimensions of the neural representations of image and weight data. Here, we verify the training of the image autoencoder by assessing its reconstruction quality in Table 13 and examining the reconstruction loss curves for test images in Figure 24. The results show that the autoencoder accurately reconstructs random test images, with the test loss stabilizing by the end of training.

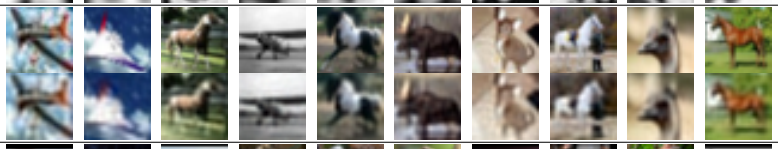
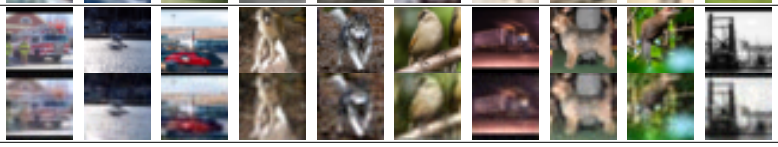
dataset	type	randomly sampled images
MNIST	original	
	reconstructed	
SVHN	original	
	reconstructed	
CIFAR-10	original	
	reconstructed	
STL-10	original	
	reconstructed	

Table 13. **Reconstructions from the image autoencoders** in Appendix J.1.

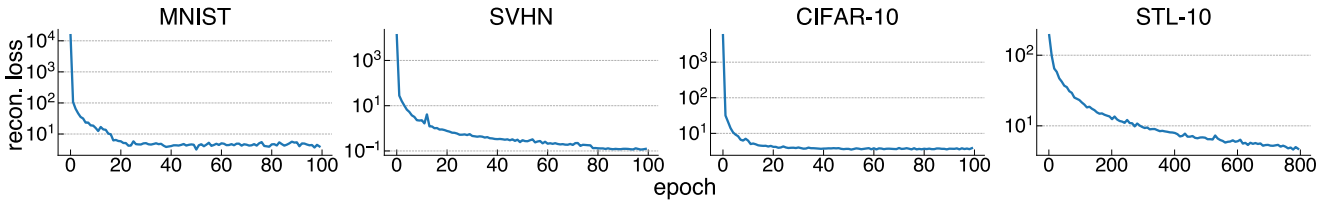


Figure 24. **Test loss curves for image autoencoder training** in Appendix J.1.

K. Can Generative Models Be Used to Store the Weight Datasets?

Hegde et al. [22] showed that diffusion models can be used to compress and store the weights of an archive of policy networks trained via Quality Diversity Reinforcement Learning (QD-RL), and enable flexible selection of specific behaviors from the policy archive. Since the generative models of weights we studied are primarily memorizing their training datasets, one might speculate whether this property could be used to compress and store the weight dataset in an alternative way.

To explore this possibility, we generate 20K checkpoints from HyperDiffusion and match each to its nearest training checkpoint. We note that only 129 (4.69%) out of 2749 training checkpoints are not replicated in generated checkpoints. Similarly, for G.pt, 4872 (47.63%) out of 10228 training checkpoints are not replicated in 50K generated checkpoints. These results suggest that generative models can indeed recover a substantial portion of the training weights.

However, the number of parameters in these generative models (223M for Hyper-Representations, 378M for G.pt, 1.4B for HyperDiffusion, and 9.6M for p-diff) far exceeds the total number of values in their respective training datasets (7.1M, 81M, 101M, and 0.6M). Therefore, storing the weight datasets implicitly within the generative models we studied would not be a space-efficient method.