
Generative Modeling of Weights: Generalization or Memorization?

Boya Zeng¹ Yida Yin¹ Zhiqiu Xu² Zhuang Liu¹

¹Princeton University ²University of Pennsylvania

Abstract

Generative models, with their success in image and video generation, have recently been explored for synthesizing effective neural network weights. These approaches take trained neural network checkpoints as training data, and aim to generate high-performing neural network weights during inference. In this work, we examine four representative methods on their ability to generate *novel* model weights, *i.e.*, weights that are different from the checkpoints seen during training. Surprisingly, we find that these methods synthesize weights largely by memorization: they produce either replicas, or at best simple interpolations, of the training checkpoints. Current methods fail to outperform simple baselines, such as adding noise to the weights or taking a simple weight ensemble, in obtaining different and simultaneously high-performing models. We further show that this memorization cannot be effectively mitigated by modifying modeling factors commonly associated with memorization in image diffusion models, or applying data augmentations. Our findings provide a realistic assessment of what types of data current generative models can model, and highlight the need for more careful evaluation of generative models in new domains. Our project page and code are available at boyazeng.github.io/weight_memorization.

1 Introduction

Generative models, particularly diffusion models for image and video synthesis, have advanced significantly in recent years. Models such as Imagen [19], DALL-E 2 [46], and Stable Diffusion [48] demonstrate exceptional photorealism, with widespread applications in commercial art and graphics. Beyond static images, generative video models like Sora [4] and Cosmos [1] have recently gained attention, achieving impressive consistency and coherence in video synthesis. The success of these models is enabled by the strong priors for generative modeling from pre-trained representations [13, 45, 63] and the algorithmic designs tailored to the visual modalities [20, 66, 40].

Building on this success, recent studies [50, 41, 12, 58] have extended the use of generative models to synthesize weights for neural networks. These methods collect network checkpoints trained with standard gradient-based optimization, and apply generative models to learn the weight distributions and produce new checkpoints, without direct access to the training data of the original task [28, 26, 34, 5]. The weights generated by these methods can often perform comparably to conventionally trained weights: they achieve high test accuracy in image classification models and high-quality 3D shape reconstructions in neural field models, across diverse datasets and model architectures.

In this study, we seek to answer an important question: have the generative models learned to produce meaningfully distinct weights that *generalize* beyond the training set of checkpoints, or do they merely *memorize* and reproduce the training data? While prior work has focused on evaluating these methods based on the performance of the generated models on the downstream tasks, this question is critical to understanding both the fundamental mechanisms and the practicality of these methods.

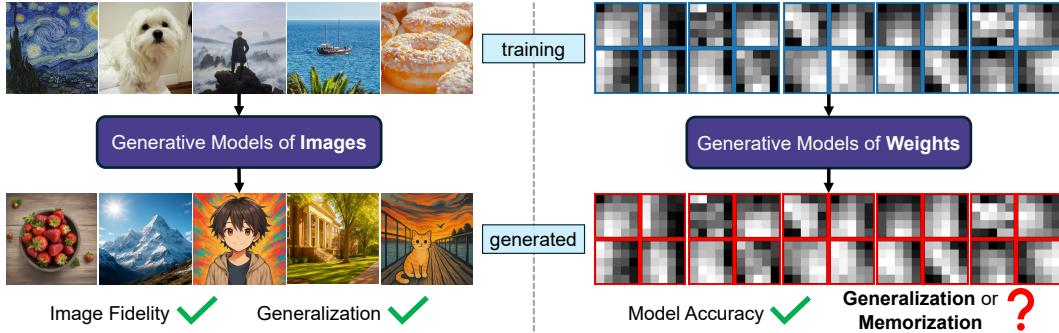


Figure 1: Building on their success in image generation [19, 46, 48], generative models have recently been applied to synthesize weights for neural networks [50, 41, 12, 58]. While they can produce effective neural network checkpoints (*e.g.*, classification models with high accuracy), it is unclear whether they can *generalize* beyond the training set to generate *novel* weights.

To investigate this question of generalization, we analyze four representative weight generation methods [50, 41, 12, 58], covering different types of generative models and downstream tasks. We first find the nearest training checkpoint to each generated checkpoint, to assess the novelty in the generated weight values. Surprisingly, *almost all* generated checkpoints *closely resemble* specific training checkpoints, showing far less novelty than a new model trained from scratch.

Beyond weight space similarity, we also examine the behaviors of generated models and their nearest training models. We compare the decision boundaries for classification models and the reconstructed 3D shapes for neural field models. In both cases, we find that these generated models, which are very close to training models in weight space, also exhibit highly similar *outputs* as the training ones.

Further, we show that current generative modeling methods offer no advantage over simple baselines for creating new model weights, in terms of producing models that differ from training checkpoints in behavior while maintaining model performance. These baselines generate new weights by adding Gaussian noise to training weights or interpolating between them. To quantify how novel a generated model’s behavior is relative to the behaviors of the training models, we compute a similarity metric for models based on their overlap in prediction errors on the test set.

We find that memorization in weight generation cannot be effectively reduced by existing mitigation strategies. Firstly, common techniques [52, 62, 14] in image diffusion—adjusting model size, training length, and regularization—do not substantially improve the novelty of the generated weight. Another strategy uses weight permutations as data augmentation to encode priors about weight structure. Yet, our results show that this alone is insufficient for generative models to effectively learn the symmetries in weight space, and may make the training distribution harder to model.

In summary, our findings consistently reveal *clear patterns of memorization in almost all generated checkpoints* from current methods, both in weight space and model behavior. We find that the generated weights largely replicate or interpolate the training weight data across all methods. This stands in stark contrast to image generation models, which are primarily recognized for their versatility and strong generalization [46, 29, 37, 53] capabilities: they can often generalize visual patterns, structures, and concepts to generate diverse outputs, rather than merely reproducing training images.

As generative modeling continues to expand into new domains and modalities [47, 67, 9, 59, 64], our findings highlight the importance of evaluating memorization in generated outputs, beyond standard quality metrics. More broadly, we hope this work can encourage researchers to consider the specific characteristics of each data modality in designing generative modeling methods.

2 Background

This section provides an overview of generative models, existing methods for generative modeling of weights, and the unique symmetries of neural network weight data.

2.1 Generative Models

Autoencoder [17] encodes data into a low-dimensional latent representation and reconstructs the input through a decoder. Diffusion models [51, 18] approximate a data distribution by learning to iteratively denoise a Gaussian noise variable. Latent diffusion models [48] compress data into latent vectors using an autoencoder and then apply diffusion models in the latent space.

2.2 Generative Modeling of Weights

Generative models are recently used to synthesize neural network weights, producing models with performance comparable to those from standard training, without additional gradient-based optimization. In this study, we analyze four representative methods, spanning unconditional and conditional generation with autoencoders and diffusion models, under each method’s primary experimental setup.

Hyper-Representations [49, 50] generate neural network weights using an autoencoder. The autoencoder is trained on checkpoints from classification models with identical architectures but different initializations. After training, kernel density estimation (KDE) is applied to the latent representations of the best-performing checkpoints. New weights are then generated by sampling a latent vector from the KDE-estimated distribution and decoding it.

G.pt [41] is a conditional diffusion model that can generate new weights for a small predefined classification model architecture, given input weights and a target loss for the generated weights. It is trained on a collection of millions of model checkpoints from tens of thousands of training runs, each paired with corresponding test losses. Once trained, G.pt produces effective model weights from randomly initialized weights and a minimal and fixed target loss.

HyperDiffusion [12] trains an unconditional diffusion model on neural field MLPs that represent 3D or 4D shapes [5]. New shapes are generated by sampling a new set of MLP weights from the diffusion model and reconstructing the mesh represented by the MLP.

P-diff [58] trains an unconditional latent diffusion model on 300 neural network checkpoints to generate new weights. These checkpoints are saved at consecutive steps during an additional training epoch of the same base image classification model, after it has converged.

Other methods. Hypernetworks [15, 3, 65, 23, 24] are neural networks trained to generate the weights of a target network based on an input representation, typically in a deterministic manner. However, as their generated weights often underperform compared to those obtained via gradient-based optimization, they are mainly used for weight initialization and neural architecture search. Unlike the generative modeling methods that we study, the training of hypernetworks relies on supervision from downstream tasks rather than a collection of trained model checkpoints.

2.3 Neural Network Symmetries

Neurons in a hidden layer have no inherent order, leading to permutation symmetry [16] in neural networks. This means that swapping neurons and adjusting weight matrices accordingly does not change a network’s function. Another form of symmetry is scaling symmetry [6], including sign flips (multiplying all incoming and outgoing weights by -1) in \tanh activations. Both G.pt and Hyper-Representations leverage permutation symmetry to augment weight data during training.

3 Memorization in Weight Generation

To evaluate the novelty of generated model weights, we compare them to the original weights used to train the generative models of weight, analyzing both their weight values and model behaviors in comparison with various baselines.

3.1 Memorization in Weight Space

A natural first step in evaluating the novelty of generated weights is to find the nearest training weights to each generated checkpoint under L_2 distance, and check for replications in weight values.

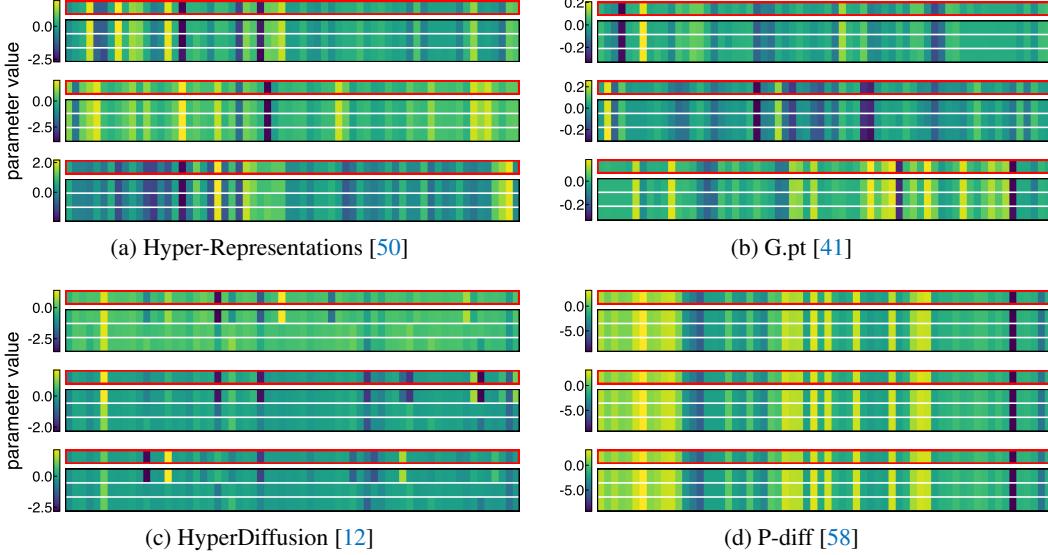


Figure 2: **Generated weights highly resemble training weights.** For each method, we display three heatmaps, showing weight values for 64 randomly selected parameter indices. In each heatmap, the top row (outlined in red) shows the values of a random generated checkpoint, and the three rows below (separated by white lines) show its three nearest training checkpoints. We observe that for every generated checkpoint, at least one training checkpoint is nearly identical to it.

However, depending on the method, permutations of weight matrices in training checkpoints or autoencoder reconstructions of training weights must also be considered.

For methods (*e.g.*, G.pt) that apply weight permutation to augment data during training, we enumerate all possible permutations of training weights to identify the closest match for each generated checkpoint. Meanwhile, we find that Hyper-Representations’ autoencoder cannot accurately reconstruct training weights, degrading model performance, as shown in Figure 3. Thus, we compare its generated weights with the reconstructed training weights instead.

Weight heatmap. For each weight generation method, we visually inspect the three nearest training checkpoints for each of three randomly selected generated checkpoints using a heatmap of weights, shown in Figure 2 (more examples in Appendices B.1 and B.2). We observe that, for all sampled generated checkpoints across all methods, there is always at least one training checkpoint that closely resembles the generated one. Further, all of p-diff’s training and generated checkpoints have nearly identical weight values. This may result from its training checkpoints being saved consecutively from the same training run, meaning that checkpoints differ only by a small number of training updates.

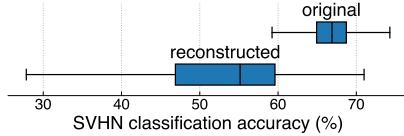


Figure 3: Reconstructing classification model weights with Hyper-Representations’ autoencoder degrades model performance.

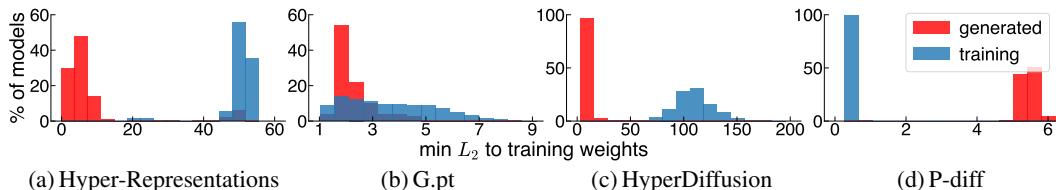


Figure 4: **Generated checkpoints are closer to training checkpoints than training checkpoints are to one another**, except for p-diff. This indicates that generated weights have lower novelty than a new model trained from scratch. The red and blue histograms represent the distributions of the L_2 distances to the nearest training checkpoints (excluding self-comparisons).

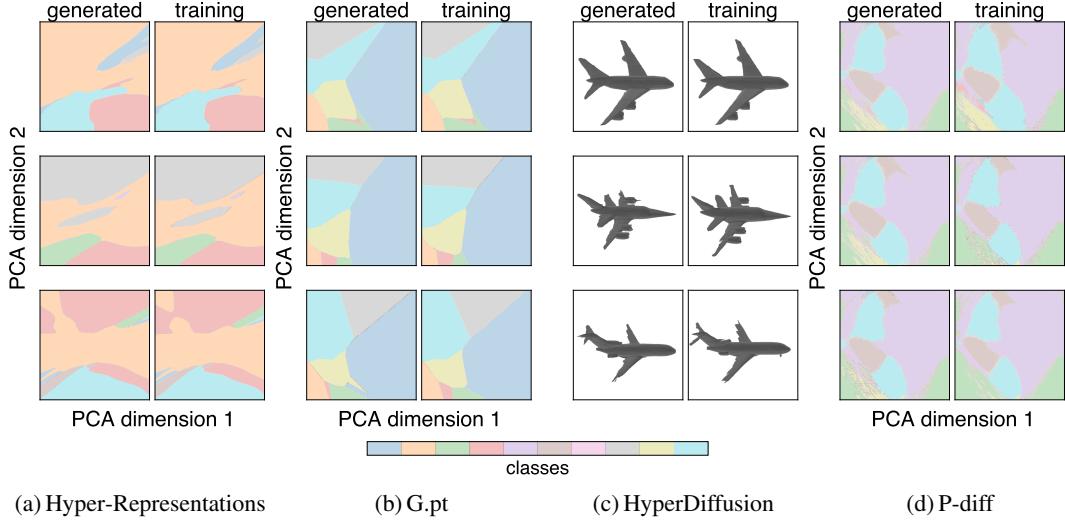


Figure 5: Generated models produce highly similar outputs to their nearest training models. Each row shows the decision boundaries or reconstructed 3D shapes of a randomly selected generated checkpoint (“generated”) and its nearest training checkpoint (“training”). For p-diff models trained on CIFAR-100, decision boundaries are shown for ten randomly selected classes.

Distance to training weights. In addition to visually inspecting weight values, we identify quantitative trends in weight value distributions that differentiate sampling a generated checkpoint from training a new model using standard gradient-based optimization.

Specifically, we compute the L_2 distance from each training and generated checkpoint to its nearest training checkpoint (excluding self-comparisons for training checkpoints), and show the distance distributions in Figure 4. For all methods except p-diff, the generated checkpoints are significantly closer to the training checkpoints than training checkpoints are to one another. For instance, 94.4% of HyperDiffusion-generated checkpoints have an L_2 distance smaller than 10 to some training checkpoints, whereas any pair of training checkpoints has an L_2 distance above 50. This indicates that these methods produce models with lower novelty than training a new model from scratch. We note that the training checkpoints used in these methods are saved from *distinct* training runs.

For p-diff, we observed that the training checkpoints are much closer to each other than the generated checkpoints are to their nearest training checkpoints. However, the low distances between training checkpoints may be expected, since they are saved from the *same* training run at *consecutive* steps.

3.2 Memorization in Model Behaviors

In Section 3.1, we showed that generated weights highly resemble the training weights. However, similar weights can still yield different behaviors. Here, we compare the behaviors of generated models to the behaviors of their nearest training models in weight space. We also assess whether generative modeling methods differ from a simple noise-addition baseline for creating new weights.

Model outputs. To understand the behaviors of generated image classification models, we project each image dataset onto two principal components, and then visualize the models’ decision boundaries. For HyperDiffusion, we reconstruct 3D shapes from the neural field models it generates.

For each method, we randomly select three generated checkpoints (additional examples in Appendices B.3 and B.4) and identify their nearest training checkpoints in weight space using the L_2 metric, as in Section 3.1. Figure 5 presents the corresponding decision boundaries or 3D shapes. We find that generated models and their nearest training models produce highly similar predictions in image classification, as indicated by the nearly identical decision regions across all class labels. Similarly, the neural field models generated by HyperDiffusion also reconstruct to nearly identical 3D shapes as training ones, with visible differences only in minor details (*e.g.*, the edges of the airplane wings).

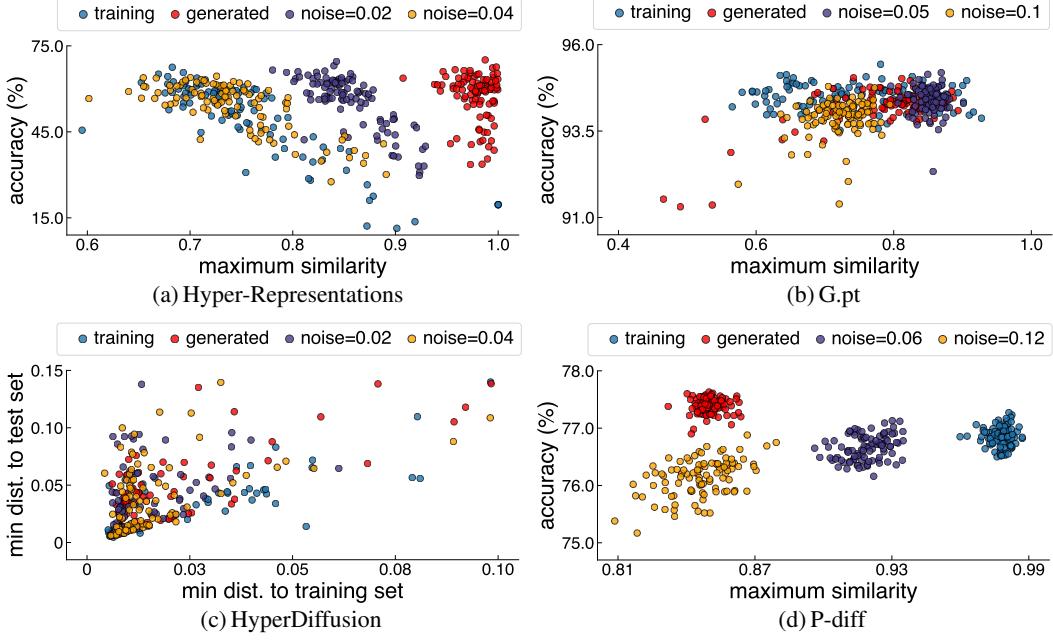


Figure 6: **Weight generation methods do not outperform noise addition in the accuracy-novelty trade-off**, except for p-diff. Novelty is measured by max error similarity (lower is better) or minimum point cloud distance to training checkpoints (higher is better). We show 100 samples per model type.

Metric for novelty. For generated weights to represent generalization, they need to behave sufficiently differently from training weights while maintaining high performance. To quantify the novelty of a generated classification model checkpoint, we adopt the model similarity metric from Wang et al. [58], which measures the Intersection over Union (IoU) of incorrect test set predictions between two model checkpoints. The formal definition of this metric is in Appendix C.1. We explore an alternative similarity metric based on the percentage of predictions that two models agree on in Appendix C.3.

To assess a checkpoint’s novelty, we compute its similarity with each training checkpoint and take the *maximum similarity*. A lower *maximum similarity* value indicates greater novelty, as it means the generated model’s classification prediction error patterns differ more from all training models.

For HyperDiffusion, which generates neural field models rather than classifiers, we use Chamfer Distance (CD), a standard metric for 3D shapes. A lower minimum CD to the *test* shapes indicates better shape quality, analogous to higher classification accuracy. A higher minimum CD to the *training* shapes suggests greater novelty, akin to lower maximum similarity in classification models.

Noise-addition baseline. We compare the accuracy and maximum similarity of the generated checkpoints against a baseline that simply adds Gaussian noise to training weights. The weight generation methods are considered superior if, at the same level of novelty relative to training models, they produce models with better performances than noise addition. Figure 6 shows the accuracy and maximum similarity distributions for training models, generated models, and noise-added models. For each weight generation method, the noise amplitudes are chosen so that the maximum similarity of noise-added models roughly matches the maximum similarity of generated models.

Accuracy-novelty trade-off. As shown in Figure 6, for G.pt and Hyper-Representations, noise-added models often achieve comparable or even higher accuracy than generated models at the same maximum similarity to training models. Similarly, for HyperDiffusion, the distributions of the minimum CD to training and test shapes show no significant difference between generated and noise-added MLPs. These results suggest that the weight generation methods may *not* offer further benefits than simply adding noise to the training weights. An exception is p-diff, where generated models achieve a better trade-off between maximum similarity and accuracy than noise-added models.

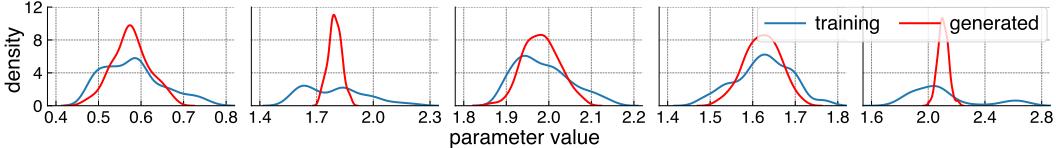


Figure 7: **Distributions of five random parameters** from the weight matrix of the first layer in the training and generated checkpoints of p-diff. The generated weights are centered around the mean of the training weights, suggesting they may be interpolations. More details are in Appendix D.2.

3.3 Understanding P-diff’s Accuracy-Novelty Trade-off

As observed in Section 3.2, different from the other methods, p-diff achieves a better trade-off between the novelty and accuracy of generated models compared to the noise-addition baseline. The generated weights even surpass the training weights in accuracy (Figure 6d).

Weight distributions. To investigate this, we examine the distribution of parameter values in generated and training models in Figure 7. The generated weight values for parameters tend to concentrate around the average of the training values. Averaging the weights of multiple models fine-tuned from the same base model is known to lead to improved accuracy [60]. Thus, p-diff may achieve higher accuracy in its generated models by producing interpolation of its training checkpoints.

Interpolation baselines. To explore this hypothesis, we generate new models using two approaches that approximate the interpolations of the training checkpoints: (1) averaging the weights of 16 randomly selected training checkpoints (“average”) and (2) fitting a Gaussian distribution to the training weight values in each parameter dimension and sampling from these distributions (“gaussian”).

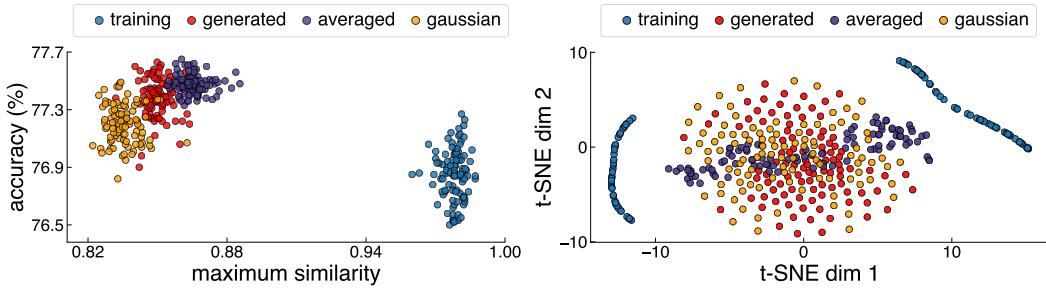


Figure 8: **P-diff generates weights with model behaviors (left) and values (right) similar to interpolations of training weights.** We compare them to two baselines: averaging training weights (“averaged”) and sampling from a Gaussian distribution fitted to training weights (“gaussian”). Model behavior is evaluated via accuracy and maximum similarity; t-SNE visualizes weight values.

Behaviors and weights. The left subplot of Figure 8 shows that the accuracy and maximum similarity of the interpolation models closely match those of p-diff. The right subplot visualizes weight distributions using t-SNE [57]. The weights generated by p-diff are close to weights from the above baselines, further suggesting that p-diff may primarily interpolate between training checkpoints. We note that this interpolation occurs within a very narrow range, as detailed in Appendix D.

Summary. The generative models of weights produce checkpoints that closely resemble training checkpoints in both weight space and model behavior, suggesting memorization and limited novelty. Moreover, they fail to outperform simple baselines in producing new models with lower similarity to training models in model behaviors while maintaining performance.

4 Analysis

In Section 3, we have demonstrated that conventional weight generation methods primarily memorize the training weights, lacking the novelty seen in image generation models (see Appendix F for

	Hyper-Representations			G.pt			HyperDiffusion		
	#params	acc. \uparrow	$L_2 \uparrow$	#params	acc. \uparrow	$L_2 \uparrow$	#params	MMD \downarrow	$L_2 \uparrow$
<i>baseline</i>									
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
<i>training epochs</i>									
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
<i>model size</i>									
+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
<i>regularization</i>									
+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 1: **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.

an example of their generalization behaviors). In this section, we examine how modeling factors influence memorization and analyze how current methods leverage the symmetries in weight space.

4.1 Impact of Modeling Factors on Memorization

The behavior of deep networks is strongly influenced by modeling choices such as training duration, model architecture, and regularization strategy. In particular, these factors have been shown to significantly impact memorization in image diffusion models [52, 62, 21, 14]. Here, we investigate whether adjusting these factors suffice 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 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 1. 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 [52, 62, 14]. 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 [11, 35]. 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 [54, 56, 42]. Here, we apply dropout [54] 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.

4.2 Weight Space Symmetries

Computer vision researchers have developed specialized algorithms and architectures to exploit spatial, color, and texture properties of images [27, 8, 10]. Likewise, for weight data, weight space symmetries, such as permutation symmetry and scaling symmetry (introduced in Section 2.3), are promising structural priors for discriminative [22, 25, 31] and generative [41, 50] modeling tasks.

Among the four methods we study, only G.pt and Hyper-Representations incorporate permutation symmetry. G.pt applies permutation solely as a data augmentation technique, while Hyper-Representations uses it to construct positive pairs for contrastive learning. Here, we evaluate whether these symmetry-based augmentations provide meaningful benefits for generative modeling.

Transformation invariance. We assess whether Hyper-Representations, trained with permutation augmentation, effectively capture weight space symmetries. Concretely, we apply function-preserving transformations to trained networks by (1) permuting neurons in hidden layers or (2) flipping weight matrix signs before and after \tanh activations. We then reconstruct the original and transformed weights using Hyper-Representations’ autoencoder, and measure the behavior similarity and accuracy difference between their reconstructions. If the autoencoder were transformation-invariant, each pair of reconstructed models would have identical accuracy and a similarity of 1.

However, as shown in the left subplot of Figure 9, reconstructions of permuted models differ in accuracy by 6.01%, and reconstructions of sign-flipped models differ by 5.82%, compared to reconstructions of their original models. For reference, the average accuracy difference between the original models is 12.72%. Similarly, the right subplot shows that the error similarity between reconstructions of original and permuted models is 0.62, and for sign-flipped models, the similarity is only 0.19. These results indicate that the autoencoder fails to fully capture weight space symmetries, despite being trained with a contrastive loss that enforces permutation invariance.

Permutation augmentation. We investigate whether applying data augmentation based on weight symmetries can reduce the memorization in HyperDiffusion. Specifically, we add 1, 3, and 7 random weight permutations during training, effectively enlarging the dataset by factors of $\times 2$, $\times 4$, and $\times 8$, respectively. To ensure convergence, we double the training epochs.

Figure 10 shows the shapes generated by the resulting models. Even when we only add a single permutation, HyperDiffusion fails to produce meaningful shapes. As the number of added permutations increases (*e.g.*, to three), HyperDiffusion fails to converge during training (see Appendix E).

Discussion. These findings suggest that applying weight space symmetries merely as data augmentations is insufficient for encoding such symmetries into generative models and may even make the training distribution harder to model. Future generative modeling methods may benefit from architectures that are invariant to weight symmetries by design. For example, prior work on classifying or editing network weights has shown that neural networks can be represented as graphs and processed by graph neural networks explicitly designed to respect weight symmetries [22, 25, 31].

Summary. Merely adjusting modeling factors cannot effectively resolve the memorization issues in generative models of weights. Further, existing data augmentation methods based on weight symmetries are insufficient for models to learn the true weight distributions and symmetries. Thus, explicitly integrating the properties of weight data into the model design may potentially be beneficial.

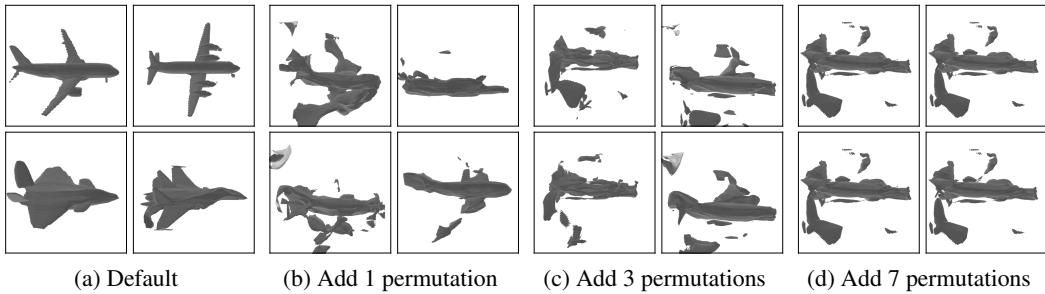


Figure 10: **HyperDiffusion fails to generate meaningful shape when any permutation is applied.** This indicates that symmetry-based augmentation is insufficient for HyperDiffusion to correctly learn the weight distributions and symmetries. Each subplot shows four random samples from a HyperDiffusion model trained with different numbers of permutations applied to the training weights.

5 Conclusion

We provide evidence that current generative modeling methods for weights primarily memorize the training data rather than generating truly novel network weights. Our analysis shows that generated checkpoints are close replications or interpolations of training checkpoints, exhibiting similar weight values and model behaviors. Notably, our analysis of model behaviors shows that the generation methods offer no clear advantage over simple baselines to create new models. Further, modeling factor adjustments or symmetry-based data augmentations cannot solve this memorization issue.

Our findings emphasize the need for careful evaluation of memorization in generative modeling, particularly as these models expand to new modalities and tasks. We hope this work can inspire future research to address the memorization issues, and further explore the practical applications of generative models for weight data and beyond.

References

- [1] Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, Prithvijit Chattopadhyay, Yongxin Chen, Yin Cui, Yifan Ding, et al. Cosmos world foundation model platform for physical ai. *arXiv preprint arXiv:2501.03575*, 2025.
- [2] Alessio Ansuini, Alessandro Laio, Jakob H Macke, and Davide Zoccolan. Intrinsic dimension of data representations in deep neural networks. In *NeurIPS*, 2019.
- [3] Andrew Brock, Theo Lim, J.M. Ritchie, and Nick Weston. SMASH: One-shot model architecture search through hypernetworks. In *ICLR*, 2018.
- [4] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, et al. Video generation models as world simulators, 2024.
- [5] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.
- [6] An Mei Chen, Haw-minn Lu, and Robert Hecht-Nielsen. On the geometry of feedforward neural network error surfaces. *Neural computation*, 1993.
- [7] Minshuo Chen, Kaixuan Huang, Tuo Zhao, and Mengdi Wang. Score approximation, estimation and distribution recovery of diffusion models on low-dimensional data. In *ICML*, 2023.
- [8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.
- [9] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *IJRR*, 2023.

- [10] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPR Workshops*, 2020.
- [11] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In *NeurIPS*, 2021.
- [12] Ziya Erkoç, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion. In *ICCV*, 2023.
- [13] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, 2021.
- [14] Xiangming Gu, Chao Du, Tianyu Pang, Chongxuan Li, Min Lin, and Ye Wang. On memorization in diffusion models. *TMLR*, 2025.
- [15] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.
- [16] Robert Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. In *Advanced Neural Computers*. 1990.
- [17] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.
- [18] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.
- [19] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022.
- [20] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016.
- [21] Zahra Kadkhodaie, Florentin Guth, Eero P Simoncelli, and Stéphane Mallat. Generalization in diffusion models arises from geometry-adaptive harmonic representations. In *ICLR*, 2024.
- [22] Ioannis Kalogeropoulos, Giorgos Bouritsas, and Yannis Panagakis. Scale equivariant graph metanetworks. In *NeurIPS*, 2024.
- [23] Boris Knyazev, Michal Drozdzal, Graham W Taylor, and Adriana Romero Soriano. Parameter prediction for unseen deep architectures. In *NeurIPS*, 2021.
- [24] Boris Knyazev, Doha Hwang, and Simon Lacoste-Julien. Can we scale transformers to predict parameters of diverse imagenet models? In *ICML*, 2023.
- [25] Miltiadis Kofinas, Boris Knyazev, Yan Zhang, Yunlu Chen, Gertjan J. Burghouts, Efstratios Gavves, Cees G. M. Snoek, and David W. Zhang. Graph neural networks for learning equivariant representations of neural networks. In *ICLR*, 2024.
- [26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- [27] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. In *NeurIPS*, 1989.
- [28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.
- [29] Tony Lee, Michihiro Yasunaga, Chenlin Meng, Yifan Mai, Joon Sung Park, Agrim Gupta, Yunzhi Zhang, Deepak Narayanan, Hannah Teufel, Marco Bellagente, et al. Holistic evaluation of text-to-image models. In *NeurIPS*, 2023.

- [30] Elizaveta Levina and Peter Bickel. Maximum likelihood estimation of intrinsic dimension. In *NeurIPS*, 2004.
- [31] Derek Lim, Haggai Maron, Marc T Law, Jonathan Lorraine, and James Lucas. Graph metanetworks for processing diverse neural architectures. In *ICLR*, 2024.
- [32] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.
- [33] David J.C. MacKay and Zoubin Ghahramani. Comments on ‘maximum likelihood estimation of intrinsic dimension’ by e. levina and p. bickel (2004), 2005. URL <http://www.inference.org.uk/mackay/dimension/>.
- [34] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, 2011.
- [35] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *ICML*, 2021.
- [36] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian conference on computer vision, graphics & image processing*, 2008.
- [37] Maya Okawa, Ekdeep S Lubana, Robert Dick, and Hidenori Tanaka. Compositional abilities emerge multiplicatively: Exploring diffusion models on a synthetic task. In *NeurIPS*, 2024.
- [38] Kazusato Oko, Shunta Akiyama, and Taiji Suzuki. Diffusion models are minimax optimal distribution estimators. In *ICML*, 2023.
- [39] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 1901.
- [40] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.
- [41] William Peebles, Ilija Radosavovic, Tim Brooks, Alexei A Efros, and Jitendra Malik. Learning to learn with generative models of neural network checkpoints. *arXiv preprint arXiv:2209.12892*, 2022.
- [42] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.
- [43] Ed Pizzi, Sreya Dutta Roy, Sugosh Nagavara Ravindra, Priya Goyal, and Matthijs Douze. A self-supervised descriptor for image copy detection. In *CVPR*, 2022.
- [44] Phil Pope, Chen Zhu, Ahmed Abdelkader, Micah Goldblum, and Tom Goldstein. The intrinsic dimension of images and its impact on learning. In *ICLR*, 2021.
- [45] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021.
- [46] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.
- [47] Suman Ravuri, Karel Lenc, Matthew Willson, Dmitry Kangin, Remi Lam, Piotr Mirowski, Megan Fitzsimons, Maria Athanassiadou, Sheleem Kashem, Sam Madge, et al. Skilful precipitation nowcasting using deep generative models of radar. *Nature*, 2021.
- [48] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.
- [49] Konstantin Schürholz, Dimche Kostadinov, and Damian Borth. Self-supervised representation learning on neural network weights for model characteristic prediction. In *NeurIPS*, 2021.

- [50] Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-representations as generative models: Sampling unseen neural network weights. In *NeurIPS*, 2022.
- [51] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, 2015.
- [52] Gowthami Somepalli, Vasu Singla, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Understanding and mitigating copying in diffusion models. In *NeurIPS*, 2023.
- [53] Gowthami Somepalli, Anubhav Gupta, Kamal Gupta, Shramay Palta, Micah Goldblum, Jonas Geiping, Abhinav Shrivastava, and Tom Goldstein. Measuring style similarity in diffusion models. *arXiv preprint arXiv:2404.01292*, 2024.
- [54] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014.
- [55] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.
- [56] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.
- [57] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 2008.
- [58] Kai Wang, Zhaopan Xu, Yukun Zhou, Zelin Zang, Trevor Darrell, Zhuang Liu, and Yang You. Neural network diffusion. *arXiv preprint arXiv:2402.13144*, 2024.
- [59] Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. De novo design of protein structure and function with rfdiffusion. *Nature*, 2023.
- [60] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, 2022.
- [61] Fan Yin, Jayanth Srinivasa, and Kai-Wei Chang. Characterizing truthfulness in large language model generations with local intrinsic dimension. In *ICML*, 2024.
- [62] TaeHo Yoon, Joo Young Choi, Sehyun Kwon, and Ernest K Ryu. Diffusion probabilistic models generalize when they fail to memorize. In *ICML workshop on structured probabilistic inference & generative modeling*, 2023.
- [63] Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffusion transformers is easier than you think. *arXiv preprint arXiv:2410.06940*, 2024.
- [64] Claudio Zeni, Robert Pinsler, Daniel Zügner, Andrew Fowler, Matthew Horton, Xiang Fu, Zilong Wang, Aliaksandra Shysheya, Jonathan Crabbe, Shoko Ueda, et al. A generative model for inorganic materials design. *Nature*, 2025.
- [65] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *ICLR*, 2019.
- [66] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017.
- [67] Jan Zrimec, Xiaozhi Fu, Azam Sheikh Muhammad, Christos Skrekas, Vykintas Jauniskis, Nora K Speicher, Christoph S Börlin, Vilhelm Verendel, Morteza Haghir Chehreghani, Devdatt Dubhashi, et al. Controlling gene expression with deep generative design of regulatory dna. *Nature communications*, 2022.

Appendix

A Experimental Setup for Each Method	15
B Additional Visualization of Model Weights and Behaviors	16
B.1 Additional Random Examples of Weight Heatmaps	16
B.2 Heatmaps by Percentile of Distance to Nearest Training Checkpoint	17
B.3 Additional Random Examples of Model Outputs	18
B.4 Model Outputs by Percentile of Distance to Nearest Training Checkpoint	19
C Additional Information on Memorization in Model Behaviors	20
C.1 Metric for Checkpoint Similarity Based on Incorrect Predictions	20
C.2 Additional Information on the Noise-Addition Baseline	20
C.3 Alternative Similarity Metric: Overlap in Classification Predictions	20
D Additional Information on the Novelty of P-diff’s Generated Models	21
D.1 Comparison with Models Trained from Scratch	21
D.2 Additional Information on P-diff Weight Value Distribution	22
E Additional Information on Permutation Augmentation for HyperDiffusion	24
F Image Generation Models Generalize with Less Data than Weight Generation Models	25
G Intrinsic Dimension of Image and Weight Data	27
G.1 Main Analysis	27
G.2 Validating the Convergence and Performance of the Image Autoencoders	28
H Can Generative Models Be Used to Store the Weight Datasets?	29
I Limitations	29
J Broader Impacts	29

A Experimental Setup for Each Method

In this paper, we analyze four generative modeling methods for weights under their primary experimental setups. Below, we detail the experimental setup used for each method.

Hyper-Representations provides weight datasets and corresponding pre-trained generative models for classification models on MNIST, SVHN, CIFAR-10, and STL-10. We evaluate the pre-trained Hyper-Representations autoencoder checkpoint for 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.

While Hyper-Representations is trained on checkpoints from epochs 21 to 25 of each training run, 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 weight datasets and pre-trained generative models 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 multiple checkpoints from each MNIST training run, we use only the final checkpoint from each training run as the training weights throughout our paper. This is because 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 only provides the pre-trained generative model for neural field model weights corresponding to airplane shapes in ShapeNet. We evaluate this pre-trained checkpoint.

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 Additional Visualization of Model Weights and Behaviors

B.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 11, 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.

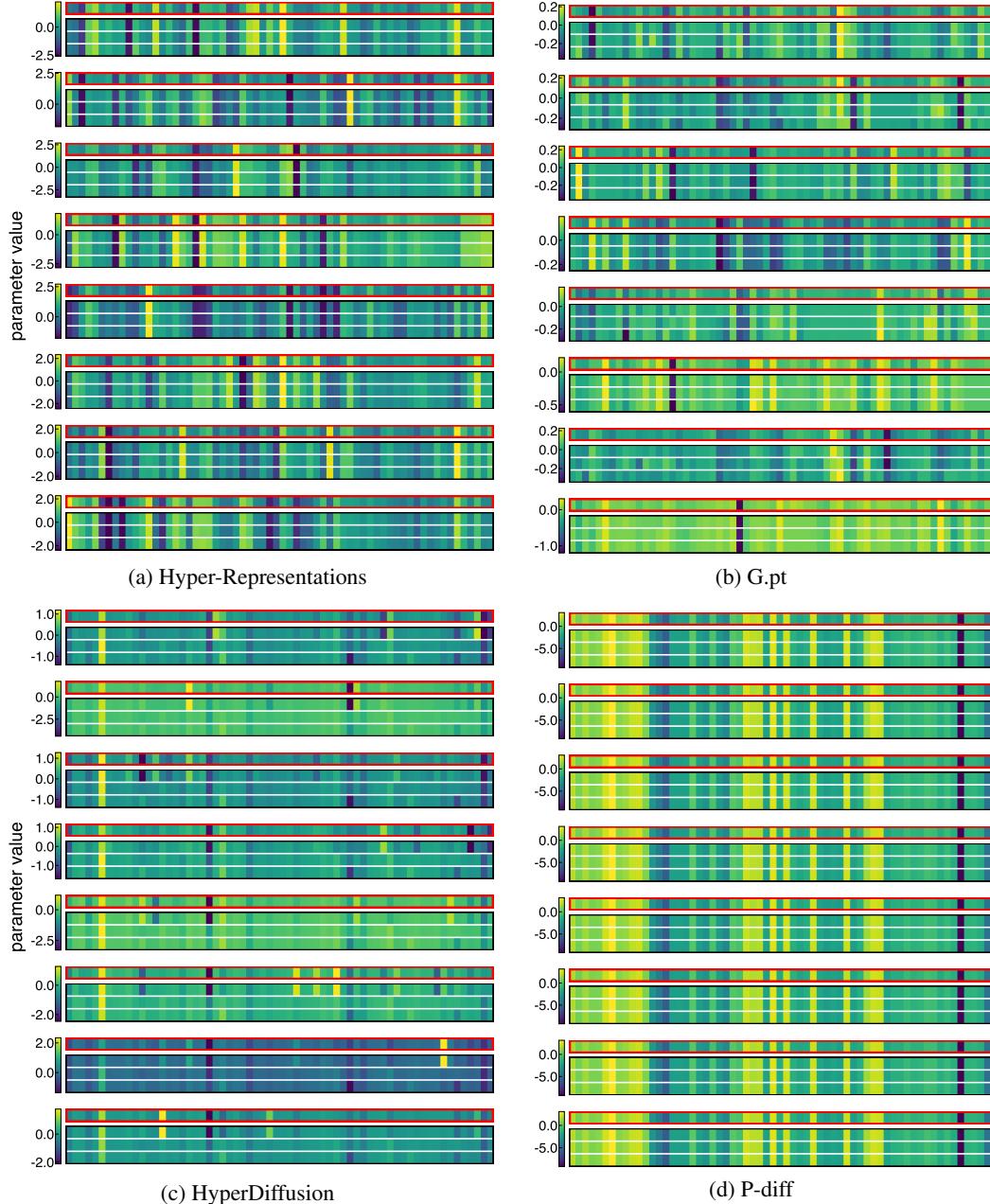


Figure 11: **Additional random examples of weight heatmap** for generated models and their nearest training models.

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

In Section 3.1 and Appendix B.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 12. 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 those 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.

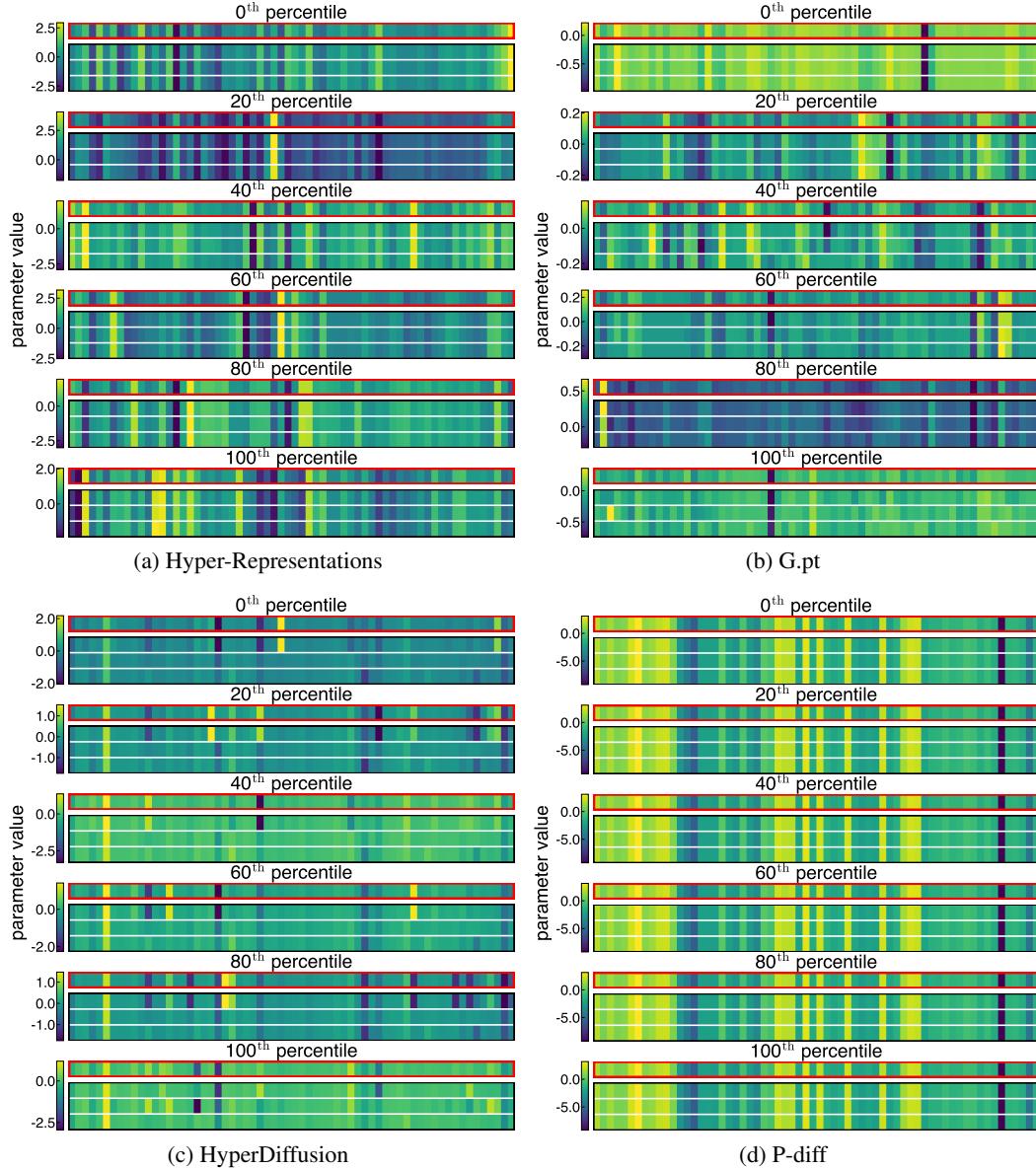


Figure 12: Heatmaps of generated checkpoints at different percentiles of distance to the nearest training checkpoint. Results are consistent with those observed for random generated weights: all generated checkpoints closely resemble their nearest training checkpoints, except for those at the 100th percentile in G.pt and HyperDiffusion, which show moderate differences.

B.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 13, 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.

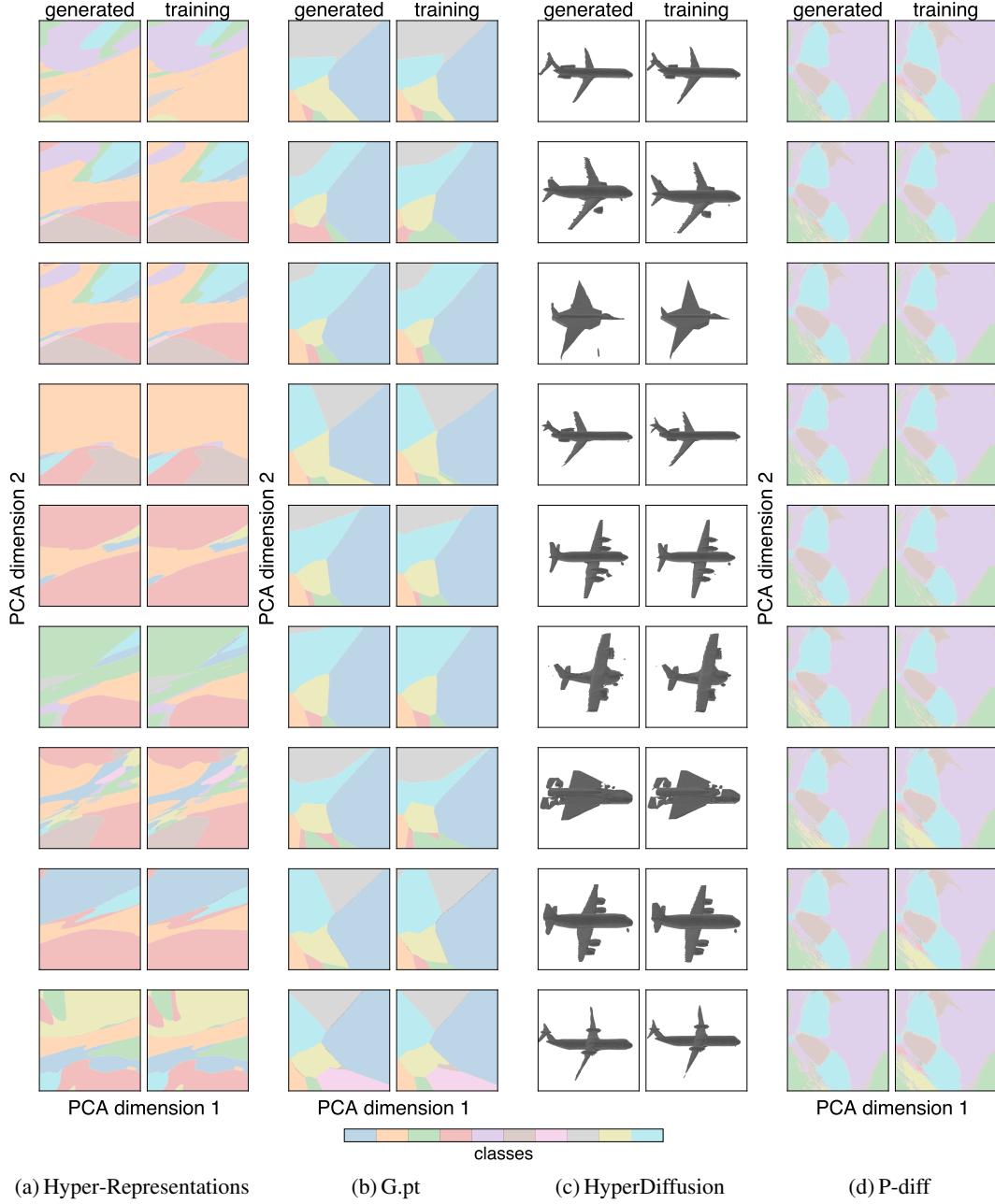


Figure 13: **Addition random examples of decision boundaries and reconstructed meshes** of generated models and their nearest training models.

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

In Section 3.2 and Appendix B.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 14. 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.

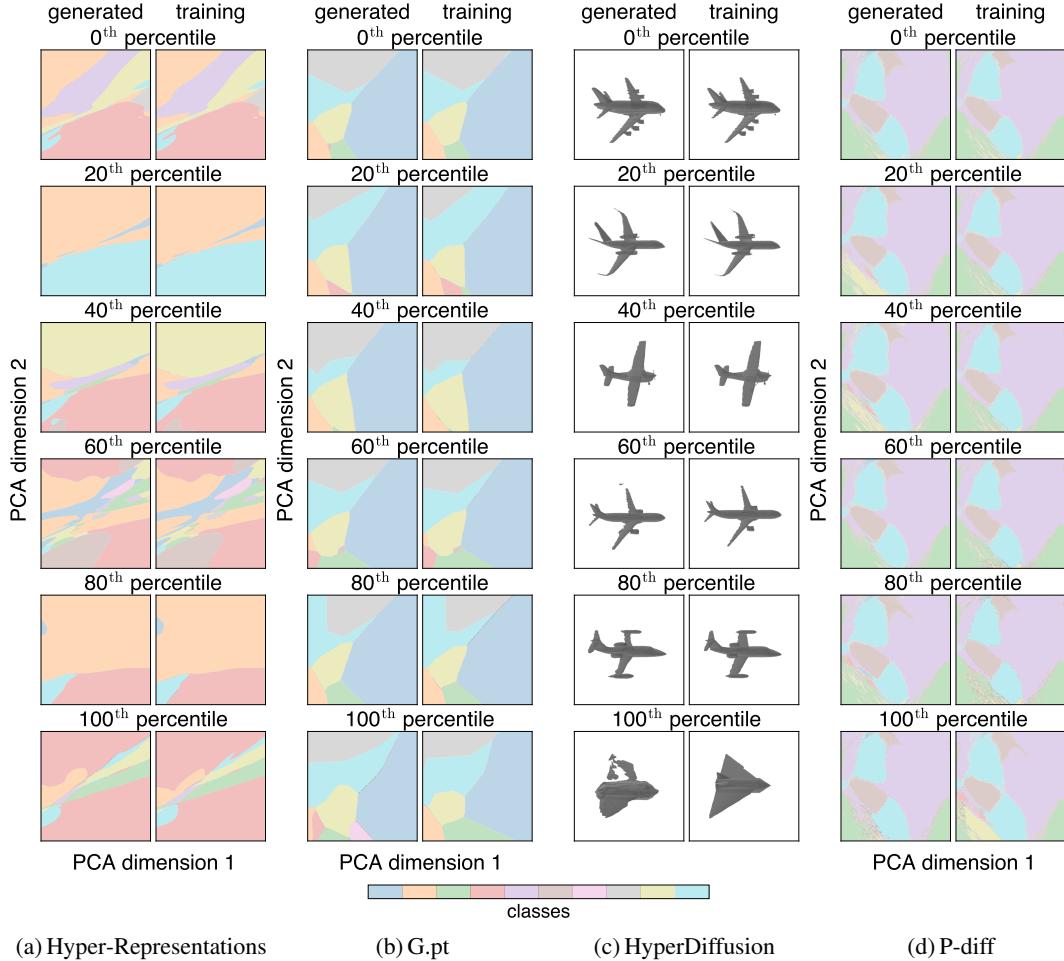


Figure 14: Decision boundaries and reconstructed meshes of generated checkpoints at different percentiles of distance to the nearest training checkpoint. Results are consistent with those observed for random generated weights: all generated checkpoints closely resemble their nearest training checkpoints 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.

C Additional Information on Memorization in Model Behaviors

C.1 Metric for Checkpoint Similarity Based on Incorrect Predictions

In Section 3.2, we used the metric from Wang et al. [58] 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:

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

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.

C.2 Additional Information on the Noise-Addition Baseline

In Section 3.2, we introduced a noise-addition baseline to compare the generated models with a simpler alternative 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.

C.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 2 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 2: **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 $\max(2x - 1, 0)$.

D 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 11) and decision boundaries (Figures 5 and 13). 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.

D.1 Comparison with Models Trained from Scratch

We train 20 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 3, 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 [32]
learning rate	5e-4
weight decay	5e-4
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$
batch size	128
learning rate schedule	cosine decay
training epochs	300
augmentation	RandomResizedCrop [55] & RandAug (9, 0.5) [10]

Table 3: **Training recipe** for CIFAR-100 classification models trained from scratch.

In Figure 15, 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.

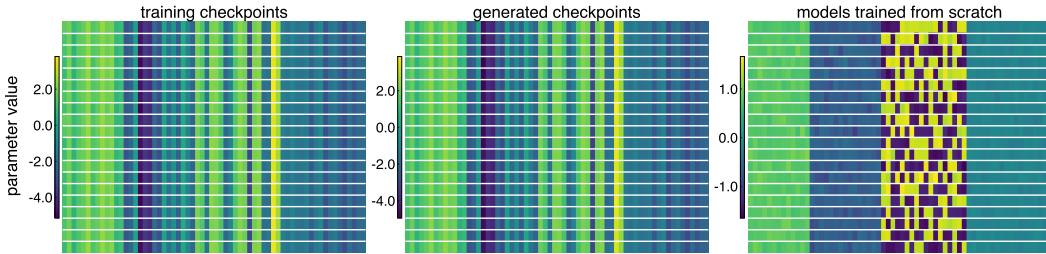


Figure 15: **P-diff’s training and generated checkpoints show limited diversity compared to models trained from scratch.** Each row (separated by white lines) is a model checkpoint; each column is a randomly selected parameter index. The same indices are used across all three subplots.

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 4, 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	44.1

Table 4: **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.

D.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 16 presents the same plot without smoothing, confirming that the moderate smoothing does not affect the observed trends.

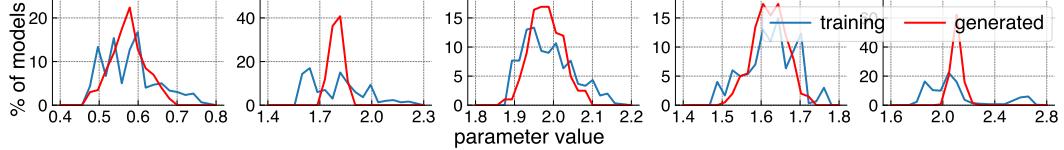


Figure 16: **Distributions of 5 randomly selected parameters** from the weight matrix of the first layer in the training and generated checkpoints of p-diff. This figure corresponds to Figure 7 but without smoothing applied to the distribution curves.

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 17, the concentration of generated weights around the mean of training weights persists across this larger set of parameters.

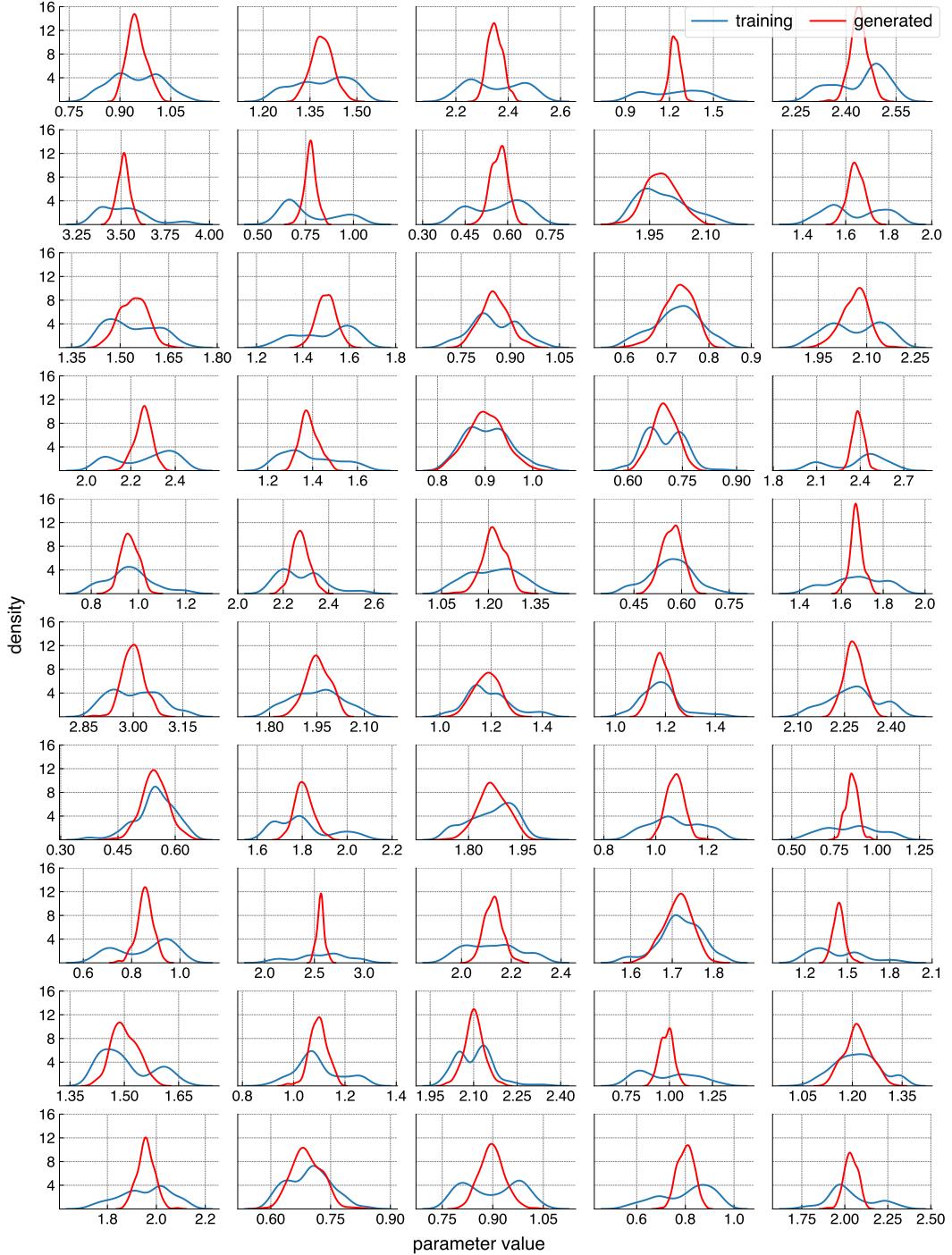


Figure 17: Distributions of 50 randomly selected parameters from the weight matrix of the first layer in the training and generated checkpoints of p-diff. This figure extends the analysis of Figure 7 to a broader set of parameters, further confirming the observed trend of generated weight values concentrating around the average of the training values.

E 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.

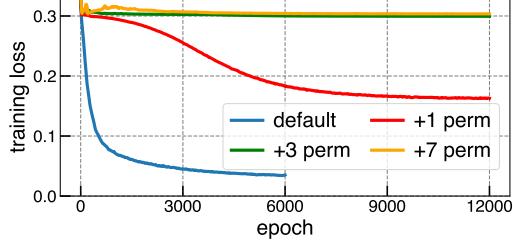


Figure 18: **HyperDiffusion fails to converge when three or more permutations are added.**

Figure 18 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 10, where the generated shapes do not represent any meaningful object.

F Image Generation Models Generalize with Less Data than Weight Generation Models

In our study, we demonstrate that current generative models for weights primarily memorize training data, drawing a comparison with the generalization of image generation models. Here, we provide a concrete example to showcase that image generation models can already generalize with the same amount of data used for weight generation model training. Concretely, we compare HyperDiffusion, an unconditional diffusion-based weight generation method, with a standard Denoising Diffusion Probabilistic Model (DDPM) [18] trained on Flowers [36].

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 5: 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 [43] to compute the similarity scores between generated and training images. Table 5 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.

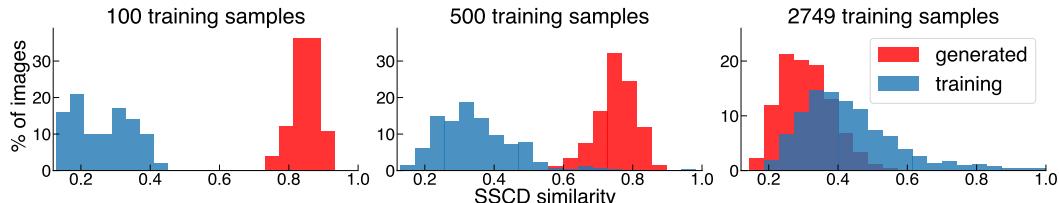


Figure 19: Image diffusion models transition from memorization to generalization with more data. The red histograms and blue curves show the distributions of SSCD similarity between each generated image and its most similar training image (red) and between each training image and its most similar training 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 19 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.

G Intrinsic Dimension of Image and Weight Data

G.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) [30, 33]. It can characterize data beyond simple linear structure as identified in alternative methods such as the Principal Component Analysis [39]. 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

$$\bar{m}_k = \left[\frac{1}{n(k-1)} \sum_{i=1}^n \sum_{j=1}^{k-1} \log \frac{T_k(x_i)}{T_j(x_i)} \right]^{-1}, \quad (2)$$

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 [44], 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. [44].

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	$k = 3$	$k = 5$	$k = 10$	$k = 20$	dataset	$k = 3$	$k = 5$	$k = 10$	$k = 20$
MNIST (image)	7	10	11	12	MNIST (image)	10	14	17	18
MNIST (weight)	56	79	86	85	MNIST (weight)	38	55	60	61
SVHN (image)	8	13	16	17	SVHN (image)	14	23	29	31
SVHN (weight)	58	81	84	43	SVHN (weight)	45	55	49	37
CIFAR-10 (image)	12	19	23	24	CIFAR-10 (image)	19	33	42	44
CIFAR-10 (weight)	62	89	99	100	CIFAR-10 (weight)	49	71	80	80
STL-10 (image)	11	17	19	19	STL-10 (image)	21	33	37	36
STL-10 (weight)	139	201	206	222	STL-10 (weight)	58	81	89	88

(a) raw data

(b) neural representations of data

Table 6: **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 6a shows the intrinsic dimensions of image and weight datasets, measured with different values of hyperparameter k . We observe that across all 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 [2, 61]. 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 [49, 50]. For image data, we train a separate autoencoder with the same architecture, latent dimensions, and objectives.

Table 6b 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 [2, 61], 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 [7, 38], it is currently unclear whether the memorization in generative models of weights we observed is directly linked to the higher intrinsic dimensions of weight data.

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

In Appendix G.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 7 and examining the reconstruction loss curves for test images in Figure 20. 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 7: **Reconstructions from the image autoencoders** in Appendix G.1.

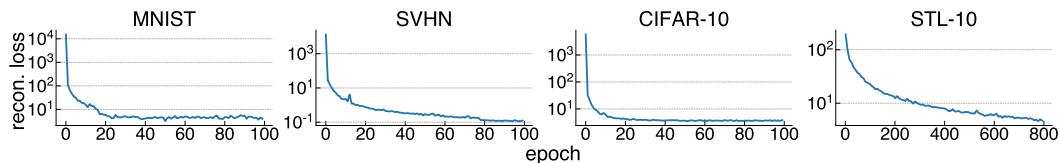


Figure 20: **Test loss curves for image autoencoder training** in Appendix G.1.

H Can Generative Models Be Used to Store the Weight Datasets?

Since the generative models of weights we studied are primarily memorizing the training datasets, one might speculate whether this property could be used to store the 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 model parameters would not be a space-efficient method.

I Limitations

This study focuses on four representative methods for generative modeling of neural network weights and shows that these models primarily memorize the training weights. However, we recognize that this does not fully cover all generative modeling methods for weight generation, and other methods may exhibit different degrees or forms of memorization.

J Broader Impacts

Our paper demonstrates that generative models of weights exhibit substantial memorization of training data, which cannot be effectively mitigated by modifying modeling factors commonly associated with memorization in image generation, or applying data augmentations based on weight-space symmetries. We hope these findings will help the research community better understand memorization in generative models and guide future efforts to mitigate the risk of memorizing sensitive information, especially when such models are trained on private datasets.