Neural Network Diffusion

Kai Wang [1] **Zhaopan Xu** [1] **Yukun Zhou** [1] **Zelin Zang** [1] **Trevor Darrell** [2] **Zhuang Liu** [*3] **Yang You** [*1]

Code: https://github.com/NUS-HPC-AI-Lab/Neural-Network-Diffusion

Abstract

Diffusion models have achieved remarkable success in image and video generation. In this work, we demonstrate that diffusion models can also *generate high-performing neural network parameters*. Our approach is simple, utilizing an autoencoder and a standard latent diffusion model. The autoencoder extracts latent representations of a subset of the trained network parameters. A diffusion model is then trained to synthesize these latent parameter representations from random noise. It then generates new representations that are passed through the autoencoder's decoder, whose outputs are ready to use as new subsets of network parameters. Across various architectures and datasets, our diffusion process consistently generates models of comparable or improved performance over trained networks, with minimal additional cost. Notably, we empirically find that the generated models perform differently with the trained networks. Our results encourage more exploration on the versatile use of diffusion models.

1. Introduction

The origin of diffusion models can be traced back to non-equilibrium thermodynamics (Jarzynski, 1997; Sohl-Dickstein et al., 2015). Diffusion processes were first utilized to progressively remove noise from inputs and generate clear images in (Sohl-Dickstein et al., 2015). Later works, such as DDPM (Ho et al., 2020) and DDIM (Song et al., 2021), refine diffusion models, with a training paradigm characterized by forward and reverse processes.

At that time, the quality of images generated by diffusion models had not yet reached a desired level. Guided-Diffusion (Dhariwal & Nichol, 2021) conducts sufficient ablations and finds a better architecture, which represents the pioneering effort to elevate diffusion models beyond GAN-based methods (Zhu et al., 2017; Isola et al., 2017) in terms of image quality. Subsequently, GLIDE (Nichol et al., 2021), Imagen (Saharia et al., 2022), DALL·E 2 (Ramesh et al., 2022), and Stable Diffusion (Rombach et al., 2022) achieve photorealistic images adopted by artists.

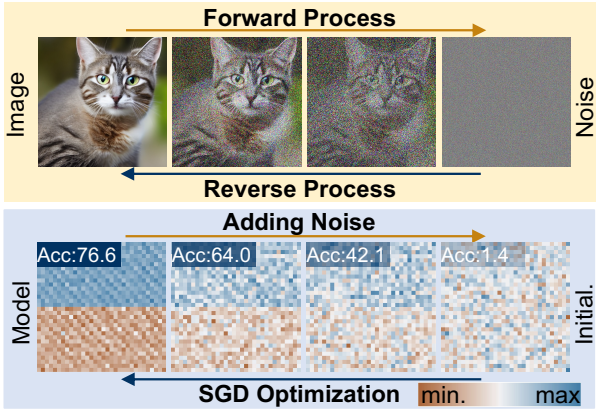


Figure 1. The top: illustrates the standard diffusion process in image generation. *The bottom:* denotes the parameter distribution of batch normalization (BN) during the training CIFAR-100 with ResNet-18. *The upper half of the bracket:* BN weights. *The lower half of the bracket:* BN biases.

Despite the great success of diffusion models in visual generation, their potential in other domains remains relatively underexplored. In this work, we demonstrate the surprising capability of diffusion models in *generating high-performing model parameters*, a task fundamentally distinct from traditional visual generation. Parameter generation focuses on creating neural network parameters that can perform well on given tasks. It has been explored from prior and probability modeling aspects, *i.e.* stochastic neural network (Sompolinsky et al., 1988; Bottou et al., 1991; Wong, 1991; Schmidt et al., 1992; Murata et al., 1994) and Bayesian neural network (Neal, 2012; Kingma & Welling, 2013; Rezende et al., 2014; Kingma et al., 2015; Gal & Ghahramani, 2016). However, using a diffusion model in parameter generation has not been well-explored yet.

Taking a closer look at the neural network training and diffusion models, the diffusion-based image generation shares commonalities with the stochastic gradient descent (SGD) learning process in the following aspects (illustrated in

[*]Equal advising, [1]National University of Singapore [2]University of California, Berkeley [3]Meta AI Research.

arXiv preprint

Fig. 1). i) Both neural network training and the reverse process of diffusion models can be regarded as transitions from random noise/initialization to specific distributions. ii) High-quality images and high-performing parameters can also be degraded into simple distributions, such as Gaussian distribution, through multiple noise additions.

Based on the observations above, we introduce a novel approach for parameter generation, named neural network diffusion (**p-diff**, p stands for parameter), which employs a standard latent diffusion model to synthesize a new set of parameters. That is motivated by the fact that the diffusion model has the capability to transform a given random distribution to a specific one. Our method is simple, comprising an autoencoder and a standard latent diffusion model to learn the distribution of high-performing parameters. First, for a subset of parameters of models trained by the SGD optimizer, the autoencoder is trained to extract the latent representations for these parameters. Then, we leverage a standard latent diffusion model to synthesize latent representations from random noise. Finally, the synthesized latent representations are passed through the trained autoencoder's decoder to yield new high-performing model parameters.

Our approach has the following characteristics: i) It consistently achieves similar, even enhanced performance than its training data, *i.e.,* models trained by SGD optimizer, across multiple datasets and architectures within seconds. ii) Our generated models have great differences from the trained models, which illustrates our approach can synthesize new parameters instead of memorizing the training samples. We hope our research can provide fresh insights into expanding the applications of diffusion models to other domains.

2. Nerual Network Diffusion

2.1. Preliminaries of diffusion models

Diffusion models typically consist of forward and reverse processes in a multi-step chain indexed by timesteps. We introduce these two processes in the following.

Forward process. Given a sample $x_0 \sim q(x)$, the forward process progressively adds Gaussian noise for T steps and obtain $x_1, x_2, \cdots, x_T$. The formulation of this process can be written as follows,

$$\begin{aligned} q(x_t|x_{t-1}) &= \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}), \\ q(x_{1:T}|x_0) &= \prod_{t=1}^{T} q(x_t|x_{t-1}), \end{aligned} \tag{1}$$

where q and $\mathcal{N}$ represent forward process and adding Gaussian noise parameterized by β_t, and $\mathbf{I}$ is the identity matrix.

Reverse process. Different from the forward process, the reverse process aims to train a denoising network to recursively remove the noise from x_t. It moves backward on the multi-step chain as t decreases from T to 0. Mathematically, the reverse process can be formulated as follows,

$$\begin{aligned} p_\theta(x_{t-1}|x_t) &= \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)), \\ p_\theta(x_{0:T}) &= p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t), \end{aligned} \tag{2}$$

where p represents the reverse process, $\mu_\theta(x_t, t)$ and $\Sigma_\theta(x_t, t))$ are the Gaussian mean and variance that estimated by the denoising network parameter θ. The denoising network in the reverse process is optimized by the standard negative log-likelihood:

$$L_{\text{dm}} = \mathcal{D}_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)), \tag{3}$$

where the $\mathcal{D}_{KL}(\cdot||\cdot)$ denotes the Kullback–Leibler (KL) divergence that is normally used to compute the difference between two distributions.

Training and inference procedures. The goal of the training diffusion model is to find the reverse transitions that maximize the likelihood of the forward transitions in each time step t. In practice, training equivalently consists of minimizing the variational upper bound. The inference procedure aims to generate novel samples from random noise via the optimized denoising parameters θ^* and the multi-step chains in the reverse process.

2.2. Overview

We propose neural network diffusion (p-diff), which aims to generate high-performing parameters from random noise. As illustrated in Fig. 2, our method consists of two processes, named parameter autoencoder and generation. Given a set of trained high-performing models, we first select a subset of these parameters and flatten them into 1-dimensional vectors. Subsequently, we introduce an encoder to extract latent representations from these vectors, accompanied by a decoder responsible for reconstructing the parameters from latent representations. Then, a standard latent diffusion model is trained to synthesize latent representations from random noise. After training, we utilize p-diff to generate new parameters via the following chain: random noise $\rightarrow$ reverse process $\rightarrow$ trained decoder $\rightarrow$ generated parameters.

2.3. Parameter autoencoder

Preparing the data for training the autoencoder. In our paper, we default to synthesizing a subset of model parameters. Therefore, to collect the training data for the autoencoder, we train a model from scratch and densely

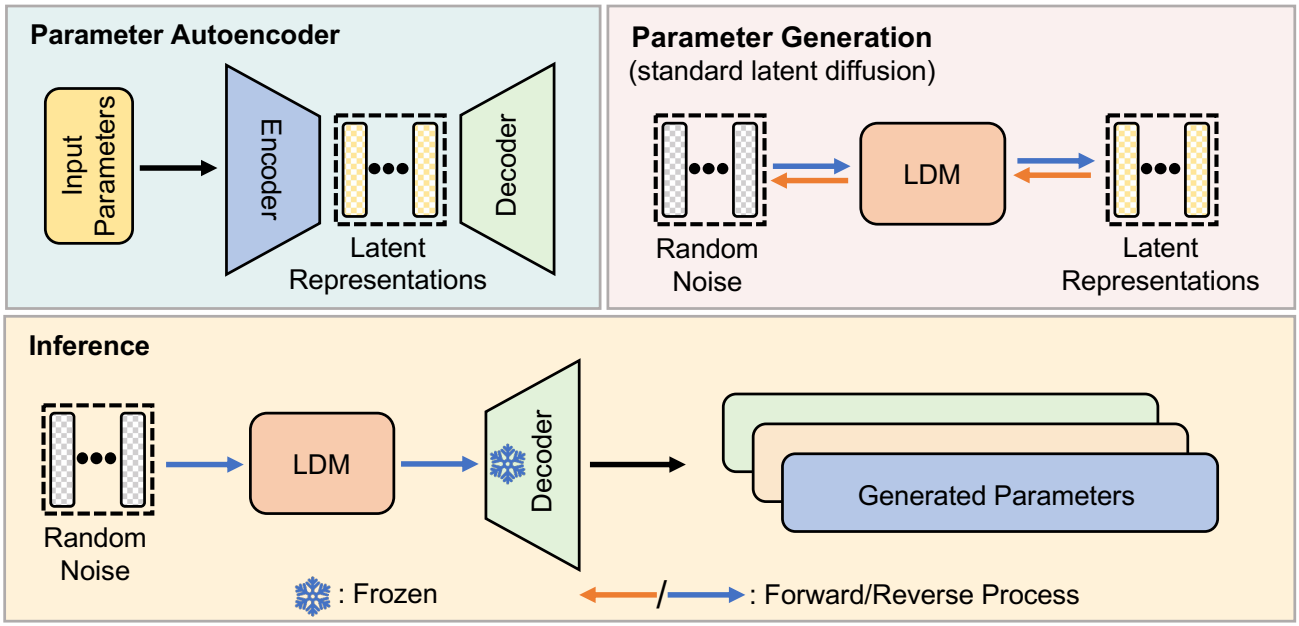


Figure 2. Our approach consists of two processes, named parameter autoencoder and generation. Parameter autoencoder aims to extract the latent representations and reconstruct model parameters via the decoder. The extracted representations are used to train a standard latent diffusion model (LDM). In the inference, the random noise is fed into LDM and trained decoder to obtain the generated parameters.

save checkpoints in the last epoch. It is worth noting that we only update the selected subset of parameters via SGD optimizer and fix the remained parameters of the model. The saved subsets of parameters $S = [s_1, \ldots, s_k, \ldots, s_K]$ is utilized to train the autoencoder, where K is the number of the training samples. For some large architectures that have been trained on large-scale datasets, considering the cost of training them from scratch, we fine-tune a subset of the parameters of the pre-trained model and densely save the fine-tuned parameters as training samples.

Training parameter autoencoder. We then flatten these parameters S into 1-dimensional vectors $V = [v_1, \ldots, v_k, \ldots, v_K]$, where $V \in \mathbb{R}^{K \times D}$ and D is the size of the subset parameters. After that, an autoencoder is trained to reconstruct these parameters V. To enhance the robustness and generalization of the autoencoder, we introduce random noise augmentation in input parameters and latent representations simultaneously. The encoding and decoding processes can be formulated as,

$$\begin{aligned} Z = [z_1^0, \ldots, z_k^0, \ldots, z_K^0] &= \underbrace{f_{\text{encoder}}(V + \xi_V, \sigma)}_{\text{encoding}}; \\ V' = [v_1', \cdots, v_k', \cdots, v_K'] &= \underbrace{f_{\text{decoder}}(Z + \xi_Z, \rho)}_{\text{decoding}}, \end{aligned} \tag{4}$$

where $f_{\text{encoder}}(\cdot, \sigma)$ and $f_{\text{decoder}}(\cdot, \rho)$ denote the encoder and decoder parameterized by σ and ρ, respectively. Z represents the latent representations, ξ_V and ξ_Z denote random noise that are added into input parameters V and latent representations Z, and V' is the reconstructed parameters. We default to using an autoencoder with a 4-layer encoder and decoder. Same as the normal autoencoder training, we minimize the mean square error (MSE) loss between V' and V as follows,

$$L_{\text{MSE}} = \frac{1}{K} \sum_{1}^{K} \|v_k - v_k'\|^2, \tag{5}$$

where v_k' is the reconstructed parameters of k-th model.

2.4. Parameter generation

One of the most direct strategies is to synthesize the novel parameters via a diffusion model. However, the memory cost of this operation is too heavy, especially when the dimension of V is ultra-large. Based on this consideration, we apply the diffusion process to the latent representations by default. For $Z = [z_1^0, \cdots, z_k^0, \cdots, z_K^0]$ extracted from parameter autoencoder, we use the optimization of DDPM (Ho et al., 2020) as follows,

$$\theta \leftarrow \theta - \nabla_\theta ||\epsilon - \epsilon_\theta(\sqrt{\overline{\alpha}_t} z_k^0 + \sqrt{1 - \overline{\alpha}_t}\epsilon, t)||^2, \tag{6}$$

where t is uniform between 1 and T, the sequence of hyperparameters $\overline{\alpha}_t$ indicates the noise strength at each step, ϵ is the added Gaussian noise, $\epsilon_\theta(\cdot)$ denotes the denoising network that parameterized by θ. After finishing the training of the parameter generation, we directly fed random noise

Table 1. We present results in the format of 'original / ensemble / p-diff'. Our method obtains similar or even higher performance than baselines. The results of p-diff is average in three runs. **Bold entries** are best results.

Network\Dataset	MNIST	CIFAR-10	CIFAR-100	STL-10	Flowers	Pets	F-101	ImageNet-1K
ResNet-18	99.2 / 99.2 / **99.3**	92.5 / 92.5 / **92.7**	76.7 / 76.7 / **76.9**	75.5 / 75.5 / 75.4	49.1 / 49.1 / **49.7**	60.9 / 60.8 / **61.1**	71.2 / 71.3 / 71.3	78.7 / 78.5 / 78.7
ResNet-50	99.4 / 99.3 / 99.4	91.3 / 91.4 / 91.3	71.6 / 71.6 / **71.7**	69.2 / 69.1 / 69.2	33.7 / 33.9 / **38.1**	58.0 / 58.0 / 58.0	68.6 / 68.5 / 68.6	79.2 / 79.2 / **79.3**
ViT-Tiny	99.5 / 99.5 / 99.5	96.8 / 96.8 / 96.8	86.7 / **86.8** / 86.7	97.3 / 97.3 / 97.3	87.5 / 87.5 / 87.5	89.3 / 89.3 / 89.3	78.5 / 78.4 / 78.5	73.7 / 73.7 / **74.1**
ViT-Base	99.5 / 99.4 / 99.5	98.7 / 98.7 / 98.7	91.5 / 91.4 / **91.7**	99.1 / 99.0 / **99.2**	98.3 / 98.3 / 98.3	91.6 / 91.5 / **91.7**	83.4 / 83.4 / 83.4	84.5 / 84.5 / **84.7**
ConvNeXt-T	99.3 / **99.4** / 99.3	97.6 / 97.6 / 97.7	87.0 / 87.0 / **87.1**	98.2 / 98.0 / 98.2	70.0 / 70.0 / **70.5**	92.9 / 92.8 / **93.0**	76.1 / 76.1 / **76.2**	82.1 / 82.1 / **82.3**
ConvNeXt-B	99.3 / 99.3 / **99.4**	98.1 / 98.1 / 98.1	88.3 / 88.4 / 88.4	98.8 / 98.8 / **98.9**	88.4 / 88.4 / **88.5**	94.1 / 94.0 / 94.1	81.4 / 81.4 / **81.6**	83.8 / 83.7 / **83.9**

into the reverse process and the trained decoder to generate a new set of high-performing parameters. These generated parameters are concatenated with the remained model parameters to form new models for evaluation. Neural network parameters and image pixels exhibit significant disparities in several key aspects, including data type, dimensions, range, and physical interpretation. Different from images, neural network parameters mostly have no spatial relevance, so we replace 2D convolutions with 1D convolutions in our parameter autoencoder and parameter generation processes.

3. Experiments

In this section, We first introduce the setup for reproducing. Then, we report the result comparisons and ablation studies.

3.1. Setup

Datasets and architectures. We evaluate our approach across a wide range of datasets, including MNIST (LeCun et al., 1998), CIFAR-10/100 (Krizhevsky et al., 2009), ImageNet-1K (Deng et al., 2009), STL-10 (Coates et al., 2011), Flowers (Nilsback & Zisserman, 2008), Pets (Parkhi et al., 2012), and F-101 (Bossard et al., 2014) to study the effectiveness of our method. We mainly conduct experiments on ResNet-18/50 (He et al., 2016), ViT-Tiny/Base (Dosovitskiy et al., 2020), and ConvNeXt-T/B (Liu et al., 2022).

Training details. The autoencoder and latent diffusion model both include a 4-layer 1D CNNs-based encoder and decoder. We default to collecting 200 training data for all architectures. For ResNet-18/50, we train the models from scratch. In the last epoch, we continue to train the last two normalization layers and fix the other parameters. We save 200 checkpoints in the last epoch, *i.e.,* original models. For ViT-Tiny/Base and ConvNeXt-T/B, we fine-tune the last two normalization parameters of the released model in the timm library (Wightman, 2019). The ξ_V and ξ_Z are Gaussian noise with amplitude of 0.001 and 0.1. In most cases, the autoencoder and latent diffusion training can be completed within 1 to 3 hours on a single Nvidia A100 40G GPU.

Inference details. We synthesize 100 novel parameters by feeding random noise into the latent diffusion model and the trained decoder. These synthesized parameters are then concatenated with the aforementioned fixed parameters to form our generated models. From these generated models, we select the one with the best performance *on the training set*. Subsequently, we evaluate its accuracy on the validation set and report the results. That is a consideration of making fair comparisons with the models trained using SGD optimization. We empirically find the performance on the training set is good for selecting models for testing.

Baselines. 1) The best validation accuracy among the original models is denoted as 'original'. 2) Average weight ensemble (Krogh & Vedelsby, 1994; Wortsman et al., 2022) of original models is denoted as 'ensemble'.

3.2. Results

Tab. 1 shows the result comparisons with two baselines across 8 datasets and 6 architectures. Based on the results, we have several observations as follows: i) In most cases, our method achieves similar or better results than two baselines. This demonstrates that our method can efficiently learn the distribution of high-performing parameters and generate superior models from random noise. ii) Our method consistently performs well on various datasets, which indicates the good generality of our method.

3.3. Ablation studies and analysis

Extensive ablation studies are conducted in this section to illustrate the characteristics of our method. We default to training ResNet-18 on CIFAR-100 and report the best, average, and medium accuracy (if not otherwise stated).

The number of training models. Tab. 2(a) varies the size of training data, *i.e.* the number of original models. We find the performance gap of best results among different numbers of the original models is minor. To comprehensively explore the influences of different numbers of training data on the performance stability, we also report the average (avg.) and median (med.) accuracy as metrics of stability of our generated models. Notably, the stability of models generated with a small number of training instances is much worse than that observed in larger settings. This can be explained by the learning principle of the diffusion model: the diffusion process may be hard to model the target distribution well if

Table 2. **p-diff main ablation experiments.** We ablate the number of original models K, the location of applying our approach, and the effect of noise augmentation. The default settings are $K = 200$, applying p-diff on the deep BN parameters (between layer16 to 18), and using noise augmentation in the input parameters and latent representations. Defaults are marked in gray. **Bold entries** are best results.

(a) Large K can improve the performance stability of our method.

K	best	avg.	med.
1	76.6	70.7	73.2
10	76.5	71.2	73.8
50	76.7	71.3	74.3
200	**76.9**	**72.4**	**75.6**
500	76.8	72.3	75.4

(b) P-diff works well on deep layers. The index of layer is aligned with the standard ResNet-18.

parameters	best	avg.	med.
original models	76.7	76.6	76.6
BN-layer10 to 14	76.8	71.9	75.3
BN-layer14 to 16	76.9	72.2	75.5
BN-layer16 to 18	**76.9**	**72.4**	**75.6**

(c) Noise augmentation makes p-diff stronger. Adding noise on latent representations is more important than on parameters.

noise augmentation	best	avg.	med.
original models	76.7	-	-
no noise	76.7	65.8	65.0
+ para. noise	76.7	66.7	67.3
+ latent noise	76.7	72.1	75.3
+ para. and latent noise	**76.9**	**72.4**	**75.6**

Table 3. We present result comparisons of original, ensemble, and p-diff under synthesizing entire model parameters setting. Our method demonstrates good generalization on ConvNet-3 and MLP-3. **Bold entries** are best results.

(a) Result comparisons on ConvNet-3 (includes three convolutional layers and one linear layer.

Dataset\Network	ConvNet-3			
	original	ensemble	p-diff	parameter number
CIFAR-10	77.2	77.3	**77.5**	24714
CIFAR-100	57.2	57.2	**57.3**	70884

(b) Result comparisons on MLP-3 (includes three linear layers and ReLU activation function).

Dataset\Network	MLP-3			
	original	ensemble	p-diff	parameter number
MNIST	85.3	85.2	**85.4**	39760
CIFAR-10	48.1	48.1	**48.2**	155135

only a few input samples are used for training.

Where to apply p-diff. We default to synthesizing the parameters of the last two normalization layers. To investigate the effectiveness of p-diff on other depths of normalization layers, we also explore the performance of synthesizing the other shallow-layer parameters. To keep an equal number of BN parameters, we implement our approach to three sets of BN layers, which are between layers with different depths. As shown in Tab. 2(b), we empirically find that our approach achieves better performances (best accuracy) than the original models on all depths of BN layers settings. Another finding is that synthesizing the deep layers can achieve better accuracy than generating the shallow ones. This is because generating shallow-layer parameters is more likely to accumulate errors during the forward propagation than generating deep-layer parameters.

Noise augmentation. Noise augmentation is designed to enhance the robustness and generalization of training the autoencoder. We ablate the effectiveness of applying this augmentation in the input parameters and latent representations, respectively. The ablation results are presented in Tab. 2(c). Several observations can be summarized as follows: i) Noise augmentation plays a crucial role in generating stable and high-performing models. ii) The performance gains of applying noise augmentation in the latent representations are larger than in the input parameters. iii) Our default setting, jointly using noise augmentation in parameters and representations obtains the best performances (includes best, average, and medium accuracy).

Generalization on entire model parameters. Until now, we have evaluated the effectiveness of our approach in synthesizing a subset of model parameters, *i.e.,* batch normalization parameters. *What about synthesizing entire model parameters?* To evaluate this, we extend our approach to two small architectures, namely MLP-3 (includes three linear layers and ReLU activation function) and ConvNet-3 (includes three convolutional layers and one linear layer). Different from the aforementioned training data collection strategy, we individually train these architectures from scratch with 200 different random seeds. We take CIFAR-10 as an example and show the details of these two architectures (convolutional layer: kernel size $\times$ kernel size, the number of channels; linear layer: input dimension, output dimension) as follows:

- ConvNet-3: conv1. 3×3, 32, conv2. 3×3, 32, conv3. 3×3, 32, linear layer. 2048, 10.
- MLP-3: linear layer1. 3072, 50, linear layer2. 50, 25, linear layer3. 25, 10.

We present result comparisons between our approach and two baselines (*i.e.,* original and ensemble) at Tab. 3. We report the comparisons and parameter numbers of ConvNet-3 on CIFAR-10/100 and MLP-3 on CIFAR-10 and MNIST datasets. These experiments demonstrate the effectiveness and generalization of our approach in synthesizing entire model parameters, *i.e.,* achieving similar or even improved

performances over baselines. These results suggest the practical applicability of our method. However, we can not synthesize the entire parameters of large architectures, such as ResNet, ViT, and ConvNeXt series. It is mainly constrained by the limitation of the GPU memory.

Parameter patterns of original models. Experimental results and ablation studies demonstrate the effectiveness of our method in generating neural network parameters. To explore the intrinsic reason behind this, we use 3 random seeds to train ResNet-18 model from scratch and visualize the parameters in Fig. 3. We visualize the heat map of parameter distribution via min-max normalization in different layers individually. Based on the visualizations of the parameters of convolutional (Conv.-layer2) and fully connected (FC-layer18) layers, there indeed exist specific parameter patterns among these layers. Based on the learning of these patterns, our approach can generate high-performing neural network parameters.

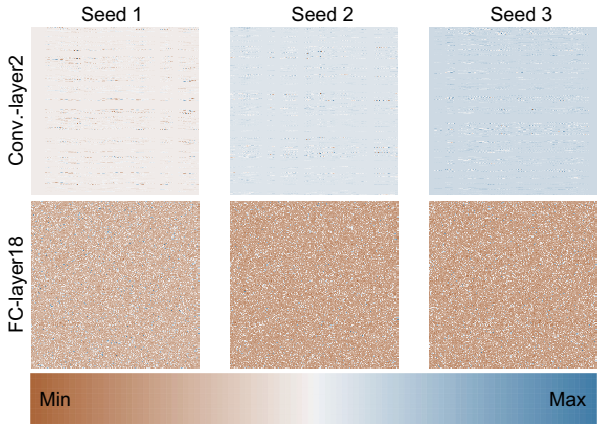


Figure 3. Visualizing the parameter distributions of convolutional (Conv.-layer2) and fully connected (FC-layer18) layers. Parameters from different layers show variant patterns while these parameters from the same layer show similar patterns. The index of layer is aligned with the standard ResNet-18.

4. Is P-diff Only Memorizing?

In this section, we mainly investigate the difference between original and generated models. We first propose a similarity metric. Then several comparisons and visualizations are conducted to illustrate the characteristics of our approach.

Questions and experiment designs. Here, we first ask the following questions: 1) Does p-diff just memorize the samples from the original models in the training set? 2) Is there any difference among adding noise or fine-tuning the original models, and the models generated by our approach? In our paper, we hope that our p-diff can generate some new parameters that perform differently than the original models. To verify this, we design experiments to study the differences between original, noise-added, fine-tuned, and p-diff models by comparing their predictions and visualizations.

Similarity metric. We conduct experiments on CIFAR-100 (Krizhevsky et al., 2009) with ResNet-18 (He et al., 2016) under the default setting, *i.e.* only generating the parameters of the last two batch normalization layers. We measure the similarity between the two models by calculating the Intersection over Union (IoU) on their wrong predictions. The IoU can be formulated as follows,

$$\text{IoU} = |P_1^{\text{wrong}} \cap P_2^{\text{wrong}}| / |P_1^{\text{wrong}} \cup P_2^{\text{wrong}}|, \quad (7)$$

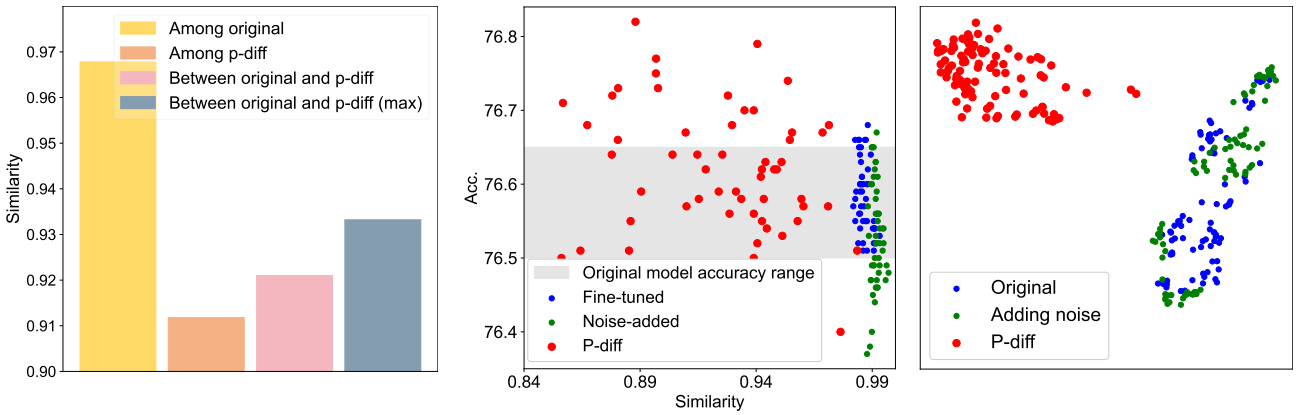
where $P_\cdot^{\text{wrong}}$ denotes the indexes of wrong predictions on the validation set, $\cap$ and $\cup$ represent union and intersection operations. A higher IoU indicates a greater similarity between the predictions of the two models. From now on, we use IoU as the similarity metric in our paper. To mitigate the influence of the performance contrasts in experiments, we select models that perform better than 76.5% by default.

Similarity of predictions. We evaluate the similarity between the original and p-diff models. For each model, we obtain its similarity by averaging the IoUs with other models. We introduce four comparisons: 1) similarity among original models; 2) similarity among p-diff models; 3) similarity between original and p-diff models; and 4) max similarity (nearest neighbor) between original and p-diff models. We calculate the IoUs for all models in the above four comparisons and report their averaged values in Fig. 4(a).

One can find that the differences among generated models are much larger than the differences among the original models. Another finding is that even the maximum similarity between the original and generated models is also lower than the similarity among the original models. It shows our p-diff can generate new parameters that perform differently with their training data (*i.e.* original models).

We also compare our approach with the fine-tuned and noise-added models. Specifically, we randomly choose one generated model, and search its nearest neighbor (*i.e.* max similarity) from the original models. Then, we fine-tune and add random noise from the nearest neighbor to obtain corresponding models. After that, we calculate the similarity of the original with fine-tuned and noise-added models, respectively. Finally, we repeat this operation fifty times and report their average IoUs for analysis. In this experiment, we also constraint the performances of all models, *i.e.,* only good models are used here for reducing the bias of visualization. We empirically set the amplitude of random noise with the range from 0.01 to 0.1 to prevent substantial performance drops.

Based on the results in Fig. 4(b), we find that the performances of fine-tuned and noise-added models are hard to



(a) Similarity comparisons of original and p-diff models.

(b) Similarity comparisons of fine-tuned, noise-added, and p-diff models.

(c) t-SNE of the latent representations of original, p-diff, and adding noise.

Figure 4. The similarity represents the Intersection of Union (IoU) over wrong predictions between/among two models (a) shows the comparisons in four cases: similarity among original models and p-diff models, similarity between original and p-diff models, and the maximum similarity (nearest neighbor) between original and p-diff models. (b) displays the accuracy and max similarity of fine-tuned, noise-added, and p-diff models. All the maximum similarities are calculated with the original models. (c) presents the t-SNE (Van der Maaten et al., 2008) of latent representations of the original models, p-diff models, and adding noise operation.

outperform the original models. Besides, the similarities between fine-tuned or noise-added and original models are very high, which indicates these two operations can not obtain novel but high-performing models. However, our generated models achieve diverse similarities and superior performances compared to the original models.

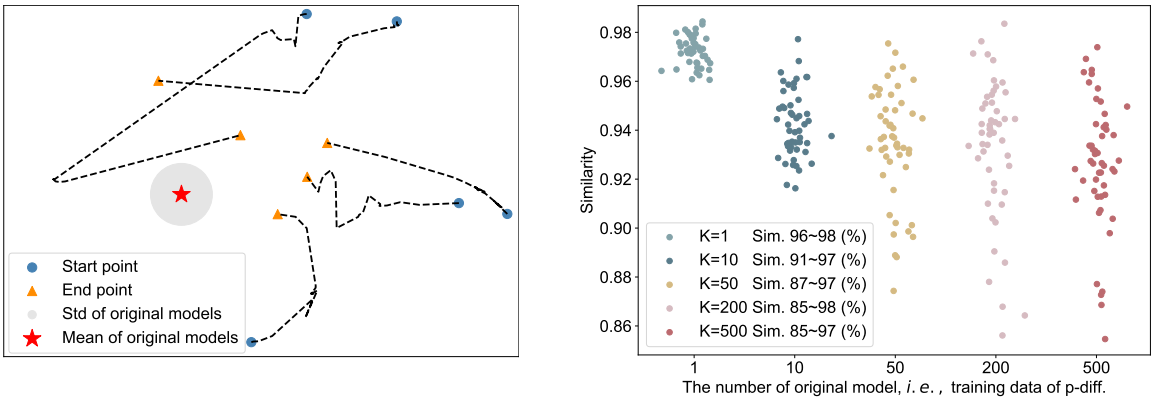
Comparison of latent representations. In addition to predictions, we assess the distributions of latent representations for the original and generated models using t-SNE (Van der Maaten et al., 2008). To identify the differences between our approach and the operation of adding noise to the latent representations of original models, we also include the adding noise operation as a comparison in Fig. 4(c). The added noise is random Gaussian noise with an amplitude of 0.1. One can find that p-diff can generate novel latent representations while adding noise just makes interpolation around the latent representations of original models.

The trajectories of p-diff process. We plot the generated parameters of different time steps in the inference stage to form trajectories to explore its generation process. Five trajectories (initialized by 5 random noise) are shown in Fig. 5(a). We also plot the average parameters of the original models and their standard deviation (std). As the time step increases, the generated parameters are overall close to the original models. Although we keep a narrow performance range constraint for visualization, there is still a certain distance between the end points (orange triangles) of trajectories and average parameters (five-pointed star). Another finding is that the five trajectories are diverse.

From memorizing to generate new parameters. To investigate the impact of the number of original models (K) on the diversity of generated models, we visualize the max similarities between original and generated models with different K in Fig. 5(b). Specifically, we continually generate parameters until 50 models perform better than 76.5% in all cases. The generated models almost memorize the original model when $K = 1$, as indicated by the narrow similarity range and high value. The similarity range of these generated models becomes larger as K increases, demonstrating our approach can generate parameters that perform differently from the original models.

5. Related Work

Diffusion models. Diffusion models have achieved remarkable results in visual generation. These methods (Ho et al., 2020; Dhariwal & Nichol, 2021; Ho et al., 2022; Peebles & Xie, 2022; Hertz et al., 2023; Li et al., 2023) are based on non-equilibrium thermodynamics (Jarzynski, 1997; Sohl-Dickstein et al., 2015), and the its pathway is similar to GAN (Zhu et al., 2017; Isola et al., 2017; Brock et al., 2018a), VAE (Kingma & Welling, 2013; Razavi et al., 2019), and flow-based model (Dinh et al., 2014; Rezende & Mohamed, 2015). Diffusion models can be categorized into three main branches. The first branch focuses on enhancing the synthesis quality of diffusion models, exemplified by models like DALL·E 2 (Ramesh et al., 2022), Imagen (Saharia et al., 2022), and Stable Diffusion (Rombach et al., 2022). The second branch aims to improve the sampling speed, including DDIM (Song et al., 2021), Analytic-DPM (Bao et al., 2022), and DPM-Solver (Lu et al., 2022). The final branch involves reevaluating diffu-



(a) Visualization of parameter trajectories of p-diff. (b) IoUs of high-performing (Acc.≥76.5%) generated models.

Figure 5. (a) shows the parameter trajectories of our approach and original models distribution via t-SNE. (b) illustrates max IoUs between generated and original models in different K settings. Sim. denotes similarity.

sion models from a continuous perspective, like score-based models (Song & Ermon, 2019; Feng et al., 2023).

Parameter generation. HyperNet (Ha et al., 2017) dynamically generates the weights of a model with variable architecture. Smash (Brock et al., 2018b) introduces a flexible scheme based on memory read-writes that can define a diverse range of architectures. (Peebles et al., 2023) collect 23 million checkpoints and train a conditional generator via a transformer-based diffusion model. MetaDiff (Zhang & Yu, 2023) introduces a diffusion-based meta-learning method for few-shot learning, where a layer is replaced by a diffusion U-Net (Ronneberger et al., 2015). HyperDiffusion (Erkoç et al., 2023) directly utilizes a diffusion model on MLPs to generate new neural implicit fields. Different from them, we analyze the intrinsic differences between images and parameters and design corresponding modules to learn the distributions of the high-performing parameters.

Stochastic and Bayesian neural networks. Our approach could be viewed as learning a prior over network parameters, represented by the trained diffusion model. Learning parameter priors for neural networks has been studied in classical literature. Stochastic neural networks (SNNs) (Sompolinsky et al., 1988; Bottou et al., 1991; Wong, 1991; Schmidt et al., 1992; Murata et al., 1994) also learn such priors by introducing randomness to improve the robustness and generalization of neural networks. The Bayesian neural networks (Neal, 2012; Kingma & Welling, 2013; Rezende et al., 2014; Kingma et al., 2015; Gal & Ghahramani, 2016) aims to model a probability distribution over neural networks to mitigate overfitting, learn from small datasets, and asses the uncertainty of model predictions. (Graves, 2011) propose an easily implementable stochastic variational method as a practical approximation to Bayesian inference for neural networks. They introduce a heuristic pruner to reduce the number of network weights, resulting in improved generalization. (Welling & Teh, 2011) combine Langevin dynamics with SGD to incorporate a Gaussian prior into the gradient. This transforms SGD optimization into a sampling process. *Bayes by Backprop* (Blundell et al., 2015) learns a probability distribution prior over the weights of a neural network. These methods mostly operate in small-scale settings, while p-diff shows its effectiveness in real-world architectures.

6. Discussion and Conclusion

Neural networks have several popular learning paradigms, such as supervised learning (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014; He et al., 2016; Dosovitskiy et al., 2020), self-supervised learning (Devlin et al., 2018; Brown et al., 2020; He et al., 2020; 2022), and more. In this study, we observe that diffusion models can be employed to generate high-performing and novel neural network parameters, demonstrating their superiority. Using diffusion steps for neural network parameter updates shows a potentially novel paradigm in deep learning.

However, we acknowledge that images/videos and parameters are signals of different natures, and this distinction must be handled with care. Additionally, even though diffusion models have achieved considerable success in image/video generation, their application to parameters remains relatively underexplored. These pose a series of challenges for neural network diffusion. We propose an initial approach to address some of these challenges. Nevertheless, there are still unresolved challenges, including memory constraints for generating the entire parameters of large architectures, the efficiency of structure designs, and performance stability.

Acknowledgments. We thank Kaiming He, Dianbo Liu, Mingjia Shi, Zheng Zhu, Bo Zhao, Jiawei Liu, Yong Liu, Ziheng Qin, Zangwei Zheng, Yifan Zhang, Xiangyu Peng, Hongyan Chang, David Yin, Dave Zhenyu Chen, Ahmad Sajedi, and George Cazenavette for valuable discussions and feedbacks.

References

Bao, F., Li, C., Zhu, J., and Zhang, B. Analytic-DPM: an analytic estimate of the optimal reverse variance in diffusion probabilistic models. In *ICLR*, 2022. URL https://openreview.net/forum?id=0xiJLKH-ufZ.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural network. In *ICML*. PMLR, 2015.

Bossard, L., Guillaumin, M., and Van Gool, L. Food-101–mining discriminative components with random forests. In *ECCV*. Springer, 2014.

Bottou, L. et al. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8), 1991.

Brock, A., Donahue, J., and Simonyan, K. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018a.

Brock, A., Lim, T., Ritchie, J., and Weston, N. SMASH: One-shot model architecture search through hypernetworks. In *ICLR*, 2018b. URL https://openreview.net/forum?id=rydeCEhs-.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *NeurIPS*, 33, 2020.

Coates, A., Ng, A., and Lee, H. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011.

Cristianini, N., Shawe-Taylor, J., et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *CVPR*. Ieee, 2009.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dhariwal, P. and Nichol, A. Diffusion models beat gans on image synthesis. *NeurIPS*, 34, 2021.

Dinh, L., Krueger, D., and Bengio, Y. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Erkoç, Z., Ma, F., Shan, Q., Nießner, M., and Dai, A. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion. *arXiv preprint arXiv:2303.17015*, 2023.

Feng, B. T., Smith, J., Rubinstein, M., Chang, H., Bouman, K. L., and Freeman, W. T. Score-based diffusion models as principled priors for inverse imaging. *arXiv preprint arXiv:2304.11751*, 2023.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML*. PMLR, 2016.

Graves, A. Practical variational inference for neural networks. *NeurIPS*, 24, 2011.

Ha, D., Dai, A. M., and Le, Q. V. Hypernetworks. In *ICLR*, 2017. URL https://openreview.net/forum?id=rkpACe1lx.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.

Hertz, A., Mokady, R., Tenenbaum, J., Aberman, K., Pritch, Y., and Cohen-or, D. Prompt-to-prompt image editing with cross-attention control. In *ICLR*, 2023. URL https://openreview.net/forum?id=_CDixzkzeyb.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *NeurIPS*, 33, 2020.

Ho, J., Chan, W., Saharia, C., Whang, J., Gao, R., Gritsenko, A., Kingma, D. P., Poole, B., Norouzi, M., Fleet, D. J., et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

Jarzynski, C. Equilibrium free-energy differences from nonequilibrium measurements: A master-equation approach. *Physical Review E*, 56(5), 1997.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kingma, D. P., Salimans, T., and Welling, M. Variational dropout and the local reparameterization trick. *NeurIPS*, 28, 2015.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 25, 2012.

Krogh, A. and Vedelsby, J. Neural network ensembles, cross validation, and active learning. *NeurIPS*, 7, 1994.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

Li, A. C., Prabhudesai, M., Duggal, S., Brown, E., and Pathak, D. Your diffusion model is secretly a zero-shot classifier. *arXiv preprint arXiv:2303.16203*, 2023.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755. Springer, 2014.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. A convnet for the 2020s. In *CVPR*, 2022.

Long, J., Shelhamer, E., and Darrell, T. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., and Zhu, J. DPM-solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *NeurIPS*, 2022. URL https://openreview.net/forum?id=2uAaGwlP_V.

Murata, N., Yoshizawa, S., and Amari, S.-i. Network information criterion-determining the number of hidden units for an artificial neural network model. *IEEE transactions on neural networks*, 5(6), 1994.

Neal, R. M. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

Nichol, A., Dhariwal, P., Ramesh, A., Shyam, P., Mishkin, P., McGrew, B., Sutskever, I., and Chen, M. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

Nilsback, M.-E. and Zisserman, A. Automated flower classification over a large number of classes. In *2008 Sixth Indian conference on computer vision, graphics & image processing*. IEEE, 2008.

Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. Cats and dogs. In *CVPR*. IEEE, 2012.

Peebles, W. and Xie, S. Scalable diffusion models with transformers. *arXiv preprint arXiv:2212.09748*, 2022.

Peebles, W., Radosavovic, I., Brooks, T., Efros, A. A., and Malik, J. Learning to learn with generative models of neural network checkpoints, 2023. URL https://openreview.net/forum?id=JXkz3zm8gJ.

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2), 2022.

Razavi, A., Van den Oord, A., and Vinyals, O. Generating diverse high-fidelity images with vq-vae-2. *NeurIPS*, 32, 2019.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *ICML*. PMLR, 2015.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*. PMLR, 2014.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pp. 234–241. Springer, 2015.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, K., Gontijo Lopes, R., Karagol Ayan,

B., Salimans, T., et al. Photorealistic text-to-image diffusion models with deep language understanding. *NeurIPS*, 35, 2022.

Schmidt, W. F., Kraaijveld, M. A., Duin, R. P., et al. Feed forward neural networks with random weights. In *ICPR*. IEEE Computer Society Press, 1992.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*. PMLR, 2015.

Sompolinsky, H., Crisanti, A., and Sommers, H.-J. Chaos in random neural networks. *Physical review letters*, 61 (3), 1988.

Song, J., Meng, C., and Ermon, S. Denoising diffusion implicit models. In *ICLR*, 2021. URL https://openreview.net/forum?id=St1giarCHLP.

Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. *NeurIPS*, 32, 2019.

Tian, Z., Shen, C., Chen, H., and He, T. Fcos: A simple and strong anchor-free object detector. *IEEE T-PAMI*, 44(4): 1922–1933, 2020.

Van der Maaten, L., Hinton, G., and Van der Maaten, L. Visualizing data using t-sne. *JMLR*, 9(11), 2008.

Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient langevin dynamics. In *ICML*, 2011.

Wightman, R. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

Wong, E. Stochastic neural networks. *Algorithmica*, 6(1-6), 1991.

Wortsman, M., Ilharco, G., Gadre, S. Y., Roelofs, R., Gontijo-Lopes, R., Morcos, A. S., Namkoong, H., Farhadi, A., Carmon, Y., Kornblith, S., et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, pp. 23965–23998. PMLR, 2022.

Zhang, B. and Yu, D. Metadiff: Meta-learning with conditional diffusion for few-shot learning. *arXiv preprint arXiv:2307.16424*, 2023.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017.

A. Experimental Settings

In this section, we introduce detailed experiment settings, datasets, and instructions of code for reproducing.

A.1. Training recipe

We provide our basic training recipe with specific details in Tab. 4. This recipe is based on the setting of ResNet-18 with CIFAR-100 dataset. We introduce these details of general training hyperparameters, autoencoder, and latent diffusion model, respectively. It may be necessary to make adjustments to the learning rate and the training iterations for other datasets.

Training Setting	Configuration
K, *i.e.*, the number of original models	200
batch size	200
Autoencoder	
optimizer	AdamW
learning rate	1e-3
training iterations	30, 000
optimizer momentum	betas=(0.9, 0.999)
weight decay	2e-6
ξ_V, *i.e.*, noise added on the input parameters	0.001
ξ_Z, *i.e.*, noise added on the latent representations	0.1
Diffusion	
optimizer	AdamW
learning rate	1e-3
training iterations	30, 000
optimizer momentum	betas=(0.9, 0.999)
weight decay	2e-6
ema β	0.9999
betas start	1e-4
betas end	2e-2
betas schedule	linear
T, *i.e.*, maximum time steps in the training stage	1000

Table 4. Our basic training recipe based on CIFAR100 dataset and ResNet-18 backbone.

A.2. Datasets

We evaluate the effectiveness of p-diff on 8 datasets. To be specific, **CIFAR-10/100** (Krizhevsky et al., 2009). The CIFAR datasets comprise colored natural images of dimensions 32×32, categorized into 10 and 100 classes, respectively. Each dataset consists of 50,000 images for training and 10,000 images for testing. **ImageNet-1K** (Deng et al., 2009) derived from the larger ImageNet-21K dataset, ImageNet-1K is a curated subset featuring 1,000 categories. It encompasses 1,281,167 training images and 50,000 validation images. **STL-10** (Coates et al., 2011) comprises 96×96 color images, spanning 10 different object categories. It serves as a versatile resource for various computer vision tasks, including image classification and object recognition. **Flowers** (Nilsback & Zisserman, 2008) is a dataset comprising 102 distinct flower categories, with each category representing a commonly occurring flower species found in the United Kingdom. **Pets** (Parkhi et al., 2012) includes around 7000 images with 37 categories. The images have large variations in scale, pose, and lighting. **F-101** (Bossard et al., 2014) consists of 365K images that are crawled from Google, Bing, Yelp, and TripAdvisor using the Food-101 taxonomy.

In the appendix, we extend our p-diff in object detection, semantic segmentation, and image generation tasks. Therefore, we also introduce the extra-used datasets in the following. **COCO** (Lin et al., 2014) consists of over 200,000 images featuring complex scenes with 80 object categories. It is widely used for object detection and segmentation tasks. We implement image generation task on CIFAR-10.

Table 5. Comparison of using 1D CNNs and fully connected (FC) layers. 1D CNNs perform better than FC layers, especially in memory and time.

Arch.	Method	Dataset	Time (s)↓	Best↑	Average↑	Median↑	Worst↑	Memory (MB)↓
ConvNet-3	FC	MNIST	17	98.0	90.1	93.6	70.2	1375
ConvNet-3	1D CNNs	MNIST	**16**	**99.2**	**92.1**	**94.2**	**73.6**	**1244**

A.3. Instructions for code

We have submitted the source code as the supplementary materials in a zipped file named as 'p-diff.zip' for reproduction. A README is also included for the instructions for running the code.

B. Explorations of Designs and Strategies

In this section, we introduce the reasons for the designs and strategies of our approach.

B.1. Why 1D CNNs?

Considering the great differences between visual data and neural network parameters, we default to using 1D CNNs in parameter autoencoder and generation. The detailed designs of 1D CNNs can be found in the following. Each layer in 1D CNNs includes two 1D convolutional layers with a normalization layer and an activation layer. More details of the 1D CNNs can be found at *core/module/modules* in our code zip file.

Here naturally raises a question: are there alternatives to 1D CNNs? We can use pure fully connected (FC) layers as an alternative. To answer this question, we compare the performance of FC layers and 1D CNNs. The experiments are conducted on MNIST with ConvNet-3 as the backbone. Based on our experimental results in Tab. 5, 1D CNNs consistently outperform FC in all architectures. Meanwhile, the memory occupancy of 1D CNNs is smaller than FC.

Table 6. Comparison of using batch normalization, group normalization, and instance normalization in our approach. We also report the results without normalization. 'norm.' denotes normalization. Default settings are marked in gray. **Bold entries** are best results.

(a) Results on CIFAR-10.

norm.	best	avg.	med.
original	94.3	-	-
no norm.	94.0	82.8	80.1
BN	88.7	84.3	88.2
GN	94.3	**89.8**	93.9
IN	**94.4**	88.5	**94.2**

(b) Results on MNIST.

norm.	best	avg.	med.
original	99.6	-	-
no norm.	99.5	84.1	98.4
BN	99.3	86.7	99.1
GN	99.6	**93.2**	99.3
IN	99.6	92.7	**99.4**

(c) Results on CIFAR-100.

norm.	best	avg.	med.
original	76.7	-	-
no norm.	76.1	67.4	69.9
BN	75.9	70.7	73.3
GN	76.8	72.1	**75.8**
IN	**76.9**	**72.4**	75.6

B.2. Is variational autoencoder an alternative to our approach?

Variational autoencoder (VAE) (Cristianini et al., 2000) can be regarded as a probabilistic generative model and achieve many remarkable results in the generation area. We also implement VAE to generate neural network parameters. We first introduce the details of VAE in our experiment. We implement vanilla VAE using the same backbone of the autoencoder in p-diff for a fair comparison. We evaluate the VAE generator in the case of different K and compare its best, average, and medium performances with p-diff generated models. Based on the results in Tab. 7, our approach outperforms VAE by a large margin in all cases. Another interesting finding is that the average performance of VAE generated models goes down as the number of original models increases.

B.3. Which normalization strategy is suitable?

Considering the intrinsic difference between images and neural network parameters, we explore the influence of different normalization strategies. We ablate batch normalization (BN), group normalization (GN), and instance normalization (IN) on CIFAR-10, MNIST, and CIFAR-100, respectively. We also implement our method without normalization for additional

Table 7. Comparisons between VAE and our proposed p-diff. VAE performs worse than our approach, especially on the metric of average and medium accuracy.

(a) Result of VAE

num. of original models	best	avg.	med.
1	75.6	61.2	70.4
10	76.5	65.8	71.5
50	76.5	63.0	71.8
200	76.7	62.7	70.8
500	76.7	62.6	71.9

(b) P-diff vs VAE, improvements are reported in ().

num. of original models	best	avg.	med.
1	76.6 (+1.0)	70.7 (+9.5)	73.2 (+2.8)
10	76.5 (+0.0)	71.2 (+5.4)	73.8 (+2.3)
50	76.7 (+0.2)	71.3 (+8.3)	74.3 (+2.5)
200	76.9 (+0.2)	72.4 (+9.7)	75.6 (+4.8)
500	76.8 (+0.1)	72.3 (+9.7)	75.4 (+3.5)

comparison. Their best, average, and medium performances of 100 generated models are reported in Tab. 6. Based on the results, we have the following observations: 1) BN obtains the worst overall performance on all three metrics. Since BN operates in the batch dimension and introduces undesired correlations among model parameters 2) GN and IN perform better than without normalization, *i.e.* 'no norm.' in the Tab. 6. That could be explained by some outlier parameters affecting the performance a lot. 3) From the metrics, we find our method has good generalization among channel-wise normalization operations, such as GN and IN.

Table 8. We design ablations about the intensity of input noise ξ_V and latent noise ξ_Z, generating variant types of parameters. 'para.' denotes parameter. Default settings are marked in gray. **Bold entries** are best results.

(a) Ablation of input noise ξ_V.

para. noise	best	avg.	med.
1e-4	76.7	72.1	75.6
1e-3	**76.9**	**72.4**	75.6
1e-2	76.3	70.4	74.4
1e-1	76.8	71.4	75.1

(b) Ablation of latent noise ξ_Z.

latent noise	best	avg.	med.
1e-3	76.7	67.3	73.2
1e-2	76.6	70.1	74.7
1e-1	**76.9**	72.6	**75.6**
1e-0	76.7	**74.0**	75.0

(c) Ablation of types of parameters.

para. type	original	best	avg.	med.
linear	76.6	76.6	47.3	71.1
conv	76.2	76.2	71.3	76.1
shortcut	75.9	76.0	73.6	75.7
bn	76.7	76.9	72.4	75.6

C. More Ablations

In this section, we introduce more ablation studies of our method. Same as the main paper, if not otherwise stated, we default to training ResNet-18 on CIFAR-100 and report the best, average, and medium accuracy.

C.1. The intensity of noise added into input parameters

In the main paper, we ablate the effectiveness of the added noise into input parameters. Here, we study the impact of the intensity of this noise. Specifically, we explore four levels of noise intensity and report their best, average, and medium results in Tab. 8(a). One can find that, our default intensity achieves the best overall performance. Both too-large and too-small noise intensities fail to obtain good results. That can be explained by that the too-large noise may destroy the original distribution of parameters while too-small noise can not provide enough effectiveness of augmentation.

C.2. The intensity of noise added into latent representations

Similar to Sec. C.1, we also ablate the noise intensity added into latent representations. As shown in Tab. 8(b), the performance stability of generated models becomes better as the noise intensity increases. However, too-large noise also breaks the distribution of the original latent representations.

C.3. The generalization on other types of parameters

In the main paper, we investigate the effectiveness of our approach in generating normalization parameters. We also evaluate our approach on other types of parameters, such as linear, convolutional, and shortcut layers. Here, we show the details

Table 9. Exploring the influence of maximum time steps in the training stage. We conduct experiments on CIFAR-10, MNIST, and CIFAR-100 datasets, respectively. **Bold entries** are best results.

(a) Results on CIFAR-10.

maximum step	best	avg.	med.
10	94.4	82.0	93.8
100	94.3	94.3	94.3
1000	94.4	88.5	94.2
2000	94.3	85.8	94.2

(b) Results on MNIST.

maximum step	best	avg.	med.
10	99.6	89.9	98.9
100	99.6	99.6	99.6
1000	99.6	92.7	99.4
2000	99.6	94.1	99.5

(c) Results on CIFAR-100.

maximum step	best	avg.	med.
10	76.6	70.6	74.9
100	76.8	75.9	76.5
1000	76.9	72.4	75.6
2000	76.8	73.1	75.1

of the above three type layers as follows: 1) linear layer: the last linear layer of ResNet-18. 2) convolutional layer: first convolutional layer of ResNet-18. 3) shortcut layer: the shortcut layer between 7th and 8th layer of ResNet-18. The training data preparation is the same as we mentioned in the main paper. As illustrated in Tab. 8, we find our approach consistently achieves similar or improved performance compared to the original models.

D. Open Explorations

D.1. Do we need to train 1000-step diffusion model?

We default to training the latent diffusion model via random sampling from 1000 time steps. Can we reduce the number of time steps in the training stage? To study the impact of the time steps, we conduct an ablation and report the results in Tab. 9. Several findings can be summarized as follows: 1) Too small time steps might not be strong enough to generate high-performing models with good stability. 2) The best stability performances are obtained by setting the maximum time steps as 100. 3). Increasing the maximum time steps from 1000 to 2000 can not improve the performance. We will further upgrade our design based on this exploration.

D.2. Potential applications

Neural network diffusion can be utilized or help the following potential research areas. 1) **Parameters initialization**: our approach can generate high-performing initialized parameters. Therefore, that would speed up the optimization and reduce the overall cost of training. 2) **Domain adaptation**: our approach may have three benefits in the domain adaptation area. First, we can directly use the diffusion process to learn the well-performed models trained by different domain data. Second, some hard adaptations can be achieved by our approach. Third, the adaptation efficiency might be improved largely.

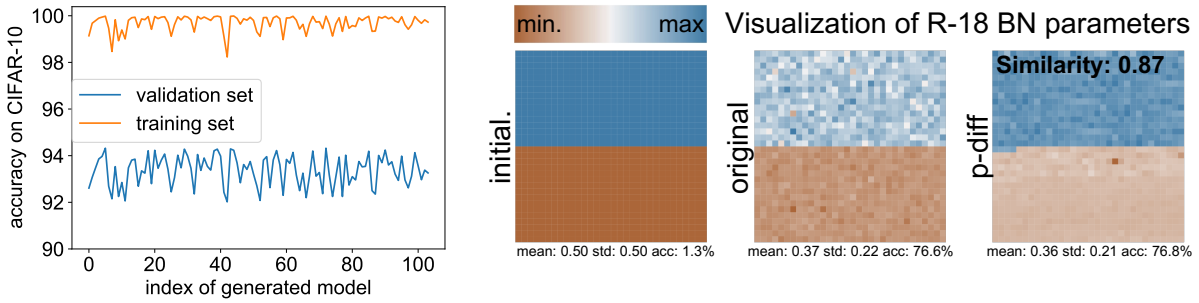
E. Other Finding and Comparison Results

E.1. How to select generated parameters?

P-diff can rapidly generate numerous high-performance models. How do we evaluate these models? There are two primary strategies. The first one is to directly test them on the *validation set* and select the best-performing model. The second one is to compute the loss of model outputs compared to the ground truth on the *training set* to choose a model. We generated hundred model parameters with performance distributions in different intervals and displayed their accuracy curves on both the training and validation sets in Fig. 6(a). The experimental results indicate that p-diff exhibits a high level of consistency between the training and validation sets. To provide a fair comparison with baseline methods, we default to choose the model that performs the best results on the training set and compare it with the baseline.

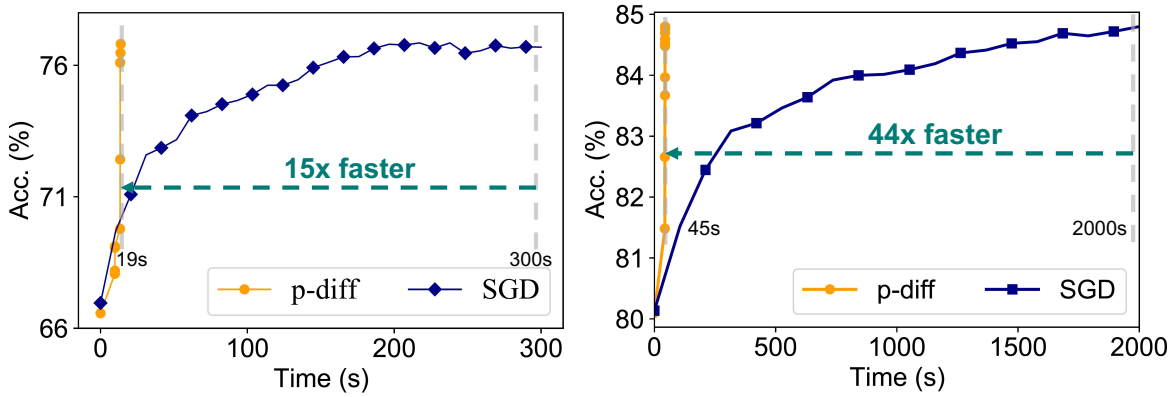
E.2. Parameter visualization

To provide a more intuitive understanding, we compare the parameters generated by our approach, SGD optimization (original), and randomly initialized. Taking ResNet-18 as an example, we report the mean, std, accuracy (acc.), and IoU of the normalization layer parameters of training on CIFAR-100 in Fig. 6(b). There is a significant difference between the parameters generated by our approach and the randomly initialized parameters, mean: 0.37 vs 0.36, std: 0.22 vs 0.21 The IoU between ours and SGD is 0.87. This visualization and results confirm that the diffusion process can learn the patterns of



(a) Accuracy distribution in p-diff models. (b) Visualization of initial, SGD-trained, p-diff generated model.

Figure 6. P-diff can generate models with great consistency on both training and validation sets contrast compared to the original model. (a) shows the accuracy distribution of training and validation sets in hundred p-diff models. (b) displays a heat map of initial, SGD-trained, p-diff generated parameters of the normalization layer in ResNet-18.



(a) Acc. of R-18. (b) Acc. of ViT-Base.

Figure 7. We compare the accuracy curves of our method and SGD under three cases. (a): ResNet-18 on CIFAR-100. (b): ViT-Base on ImageNet-1K. Our approach speeds up at least 15 $\times$ than standard SGD process.

high-performance parameters and generate new good models from random noise. More importantly, our generated model has a great behavior contrast compared to the original model, which is reflected in the low IoU value.

E.3. Efficiency of parameter generation

To evaluate the generation efficiency of our method, we compare the validation accuracy curves of our method and SGD training among the following cases: 1) parameter diffusion with ResNet-18 on CIFAR-100; 2) parameter diffusion with ViT-Base on ImageNet-1K. We utilize the random initialized parameters for our method and SGD to make a fair comparison. As illustrated in Fig. 7, our method can speed up at least 15 $\times$ compared to the SGD without performance drops. On ImageNet-1K, we can speed up by 44 $\times$ when compared to the vanilla SGD optimization, which illustrates the more significant potential when applying our approach to large training datasets.

F. Generalization on Other Tasks

We implement our method for other visual tasks, *i.e.,* object detection, semantic segmentation, image generation. Experimental results illustrate the ability of our method to generalize to various tasks.

F.1. Object detection

Faster R-CNN (Ren et al., 2015) utilizes a region proposal network (RPN) which shares full-image convolutional features with the detection network to improve Fast R-CNN on object detection task. The FCOS (Fully Convolutional One-

Stage) (Tian et al., 2020) model is a single-stage object detection model that simplifies the detection process by eliminating the need for anchor boxes. In the object detection task, We implement Faster R-CNN (Ren et al., 2015) and FCOS (Tian et al., 2020) with ResNet-50 backbone on the COCO (Lin et al., 2014) dataset based on torch/torchvision. Considering the time cost of data for p-diff, we directly use the pre-trained parameters as our first training data, then fine-tune it to obtain other training data. The parameters of the boxing predictor layer are generated by p-diff. We report the results in Tab. 10. Our method can get models with similar or even better performance than the original model in seconds.

model/performance	best original mAP	best p-diff mAP
Faster R-CNN	36.9	37.0
FCOS	39.1	39.1

Table 10. P-diff in object detection task. We report the mAP of best original model and best p-diff generated model.

F.2. Semantic segmentation

Fully Convolutional Network (FCN) (Long et al., 2015) was designed to efficiently process and analyze images at the pixel level, allowing for the semantic segmentation of objects within an image. Following the approach in object detection, we implement semantic segmentation task using FCN (Long et al., 2015) with ResNet-50 backbone to evaluate a subset of COCO val2017, on the 20 categories that are present in the Pascal VOC dataset. We generate a subset of the parameters of backbone and report the results in Tab. 11. Our approach can generate high-performing neural network parameters in semantic segmentation task.

model/performance	original		p-diff	
	mean IoU	pixelwise acc.	mean IoU	pixelwise acc.
FCN	60.5	91.4	60.7	91.5

Table 11. P-diff in semantic segmentation task. We report mean IoU and pixelwise accuracy of best original model and best p-diff model.

F.3. Image generation

DDPM (Ho et al., 2020) is a diffusion-based method in image generation, where UNet (Ronneberger et al., 2015) is used to model the noise. In the image generation task, we use p-diff to generate a subset of model parameters of UNet. For comparison, we evaluate the p-diff model's FID score on the CIFAR-10 dataset and report the results in Tab. 12. The best p-diff generated UNet get similar performance to the original model.

model/performance	original FID	p-diff FID
DDPM UNet	3.17	3.19

Table 12. P-diff in image generation task. We report the FID score on the CIFAR-10 dataset.