Neural Network Parameter Diffusion

Kai Wang[1], **Zhaopan Xu**[1], **Yukun Zhou**[1], **Trevor Darrell**[2], **Zhuang Liu**[3]*, **Yang You**[2]*
[1]National University of Singapore [2]University of California, Berkeley [3]Meta AI
Code: NUS-HPC-AI-Lab/Neural-Network-Parameter-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 are not memorizing 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 [30, 65]. Diffusion processes were first utilized to progressively remove noise from inputs and generate clear images in [65]. Later works, such as DDPM [28] and DDIM [67], 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 [15] conducts sufficient ablations and finds a better architecture, which represents the pioneering effort to elevate diffusion models beyond GAN-based methods [29, 80] in terms of image quality. Subsequently, GLIDE [49], Imagen [61], DALL·E 2 [54], and Stable Diffusion [59] achieve photorealistic images adopted by artists.

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 [5, 46, 62, 66, 74] and Bayesian neural network [20, 32, 33, 48, 58]. 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, diffusion-based image generation shares commonalities with the stochastic gradient descent (SGD) learning process in the following aspects (illustrated in 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.

*equal advising

Preprint. Under review.

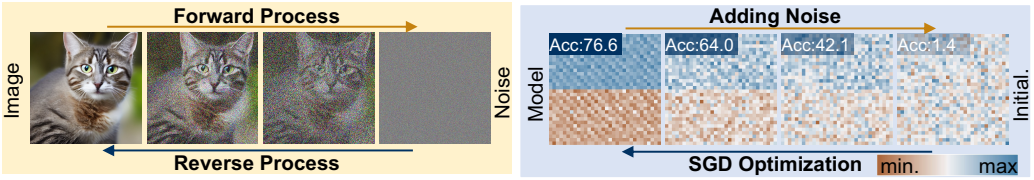


Figure 1: *The left:* illustrates the standard diffusion process in image generation. *The right:* denotes the parameter distribution of batch normalization (BN) during the training CIFAR-100 with ResNet-18. *The upper half of the bracket:* BN weights. *The lower half of the bracket:* BN biases.

Based on the observations above, we introduce a novel approach for parameter generation, named Neural Network Parameter-Diffusion (**p-diff**), 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 Neural Network Parameter 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 section:

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. In contrast to 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:

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

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

Training and inference procedures. The goal of training the 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.

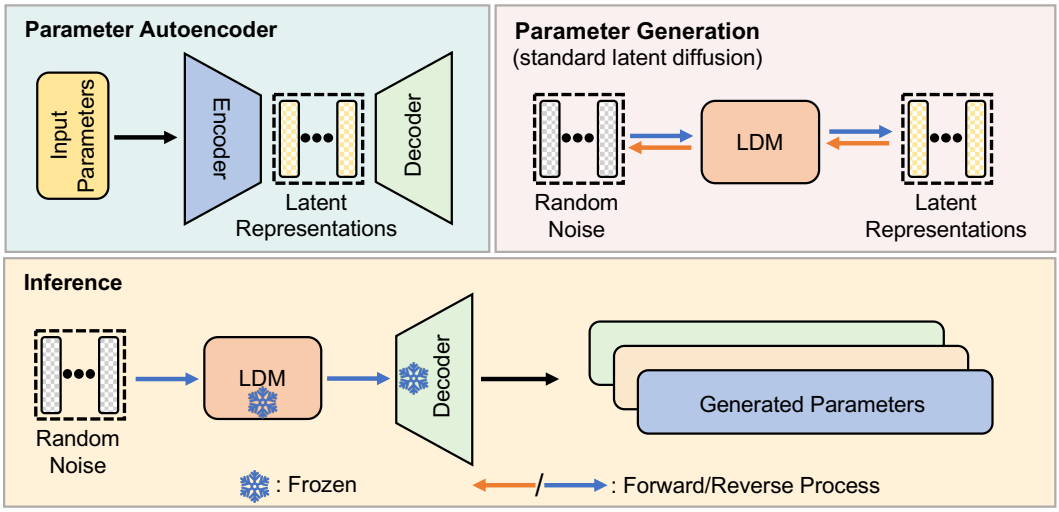


Figure 2: Our approach consists of two processes: parameter autoencoder and generation. Parameter autoencoder aims to extract the latent representations and reconstruct model parameters via the decoder. The extracted representations are used to train a standard latent diffusion model (LDM). In inference,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

2.2 Overview

We propose neural network parameter 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 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 $S = [s_1, \ldots, s_k, \ldots, s_K]$ are 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,

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

where v_k' is the reconstructed parameters of the 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 [28] as follows,

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

where t is uniform between 1 and T, the sequence of hyperparameters $\bar{\alpha}_t$ 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 feed random noise 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. In contrast to 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 our experimental setup before reporting our results and ablation studies.

3.1 Setup

Datasets and architectures. We evaluate our approach across a wide range of datasets, including MNIST [39], CIFAR-10/100 [36], ImageNet-1K [12], STL-10 [10], Flowers [50], Pets [51], and F-101 [4] to study the effectiveness of our method. We mainly conduct experiments on ResNet-18/50 [25], ViT-Tiny/Base [17], and ConvNeXt-T/B [42].

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 [73]. 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 sets of 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 [38, 75] of original models is denoted as 'ensemble'.

Table 1: We present results in the format of 'original / ensemble / p-diff'. Our method obtains similar or higher performance than baselines. Our results are average in 3 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**

3.2 Results

Performance on partial parameter generation. 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.

Table 2: 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

Generalization on entire parameter generation. 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$\times$3, 32, conv2. 3$\times$3, 32, conv3. 3$\times$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. 2. 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.

Table 3: **p-diff main ablation experiments.** We ablate the number of original models K, the location of applying p-diff, 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 aug. 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**

3.3 Ablation studies and analysis

Extensive ablations are conducted to illustrate the characteristics of our method. We train ResNet-18 on CIFAR-100 and report the best, average, and medium accuracy (if not otherwise stated).

The number of training models. Tab. 3(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 only a few input samples are used for training.

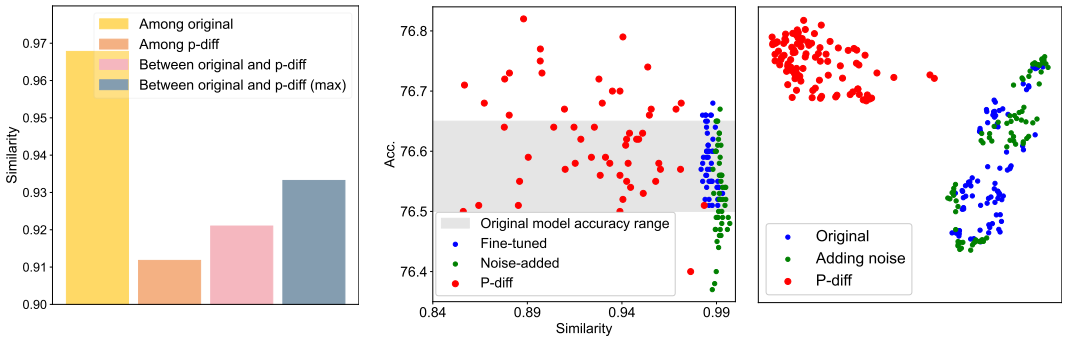
Where to apply p-diff. We default to synthesizing the parameters of the last two normalization layers. To investigate the effectiveness of p-diff on other depths of normalization layers, we also explore the performance of synthesizing the other shallow-layer parameters. To keep an equal number of BN parameters, we implement our approach to three sets of BN layers, which are between layers with different depths. As shown in Tab. 3(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. 3(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 to other optimizers. We default to evaluating the effectiveness of our approach with SGD-trained models. To verify the robustness of p-diff with other optimizers, we evaluate the generalization of our method with Adam [31] and AdamW [44]. We follow the same data preparation, training, and evaluation process as claimed in the previous section. Here, we implement our method with ResNet-18 on the CIFAR-100 dataset. As shown in Tab. 4, our approach achieves comparable results to the original models, demonstrating good generalization with both Adam and AdamW.

Table 4: Results on Adam and AdamW.

optimizer	original	best	avg.	med.
Adam [31]	70.5	70.5	64.6	69.3
AdamW [44]	71.5	71.6	67.0	70.5



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

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

(c) t-SNE visualizations of original, p-diff, and adding noise.

Figure 3: 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 [71] of latent representations of the original models, p-diff models, and adding noise operation.

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.

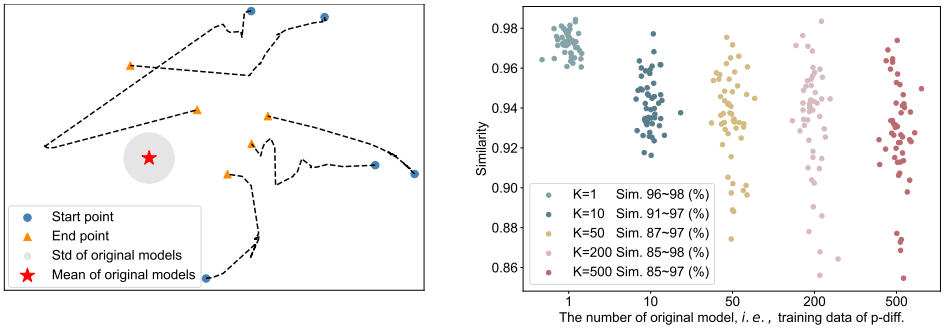
Similarity metric. We conduct experiments on CIFAR-100 [36] with ResNet-18 [25] under the default setting, *i.e.* only generating the parameters of the last two batch normalization layers. We measure the similarity between the two models by calculating the Intersection over Union (IoU) on their wrong predictions. The IoU can be formulated as follows,

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

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. 3(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).



(a) Parameter trajectories of p-diff.

(b) IoUs of generated models (Acc.$\geq$76.5%).

Figure 4: (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.

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. 3(b), we find that the performances of fine-tuned and noise-added models are hard to 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 [71]. 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. 3(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. 4(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. 4(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 [15, 26, 27, 28, 40, 53] are based on non-equilibrium thermodynamics [30, 65], and the its pathway is similar to GAN [6, 29, 80], VAE [32, 55], and flow-based model [16, 57]. 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 [54], Imagen [61], and Stable Diffusion [59]. The second branch aims to improve the sampling speed, including DDIM [67], Analytic-DPM [2], and DPM-Solver [45]. The final branch involves reevaluating diffusion models from a continuous perspective, like score-based models [19, 68].

Parameter generation. HyperNet [22] dynamically generates the weights of a model with variable architecture. Smash [7] introduces a flexible scheme based on memory read-writes that can define a diverse range of architectures. G.pt [52] collects 23 million checkpoints (including accuracy, loss, and error) and trains a conditional generator using a transformer-based diffusion model. This model takes the current parameters as input and uses the target loss or error as a condition to generate new parameters. However, it is challenging to achieve results comparable to those of their training data (models). MetaDiff [76] introduces a diffusion-based meta-learning method for few-shot learning, where a layer is replaced by a diffusion U-Net [60]. It does not have the capability to generate high-performing neural network weights from random noise. Diffusion-SDF [9] proposes a diffusion model for shape completion, single-view reconstruction, and reconstruction of real-scanned point clouds. They still use the diffusion model for visual generation, not for parameter generation. HyperDiffusion [18] uses a diffusion model on MLPs to generate new neural implicit fields (a representation for 3D shape). DWSNets [47] introduces a symmetry-based approach for designing neural architectures that operate in deep weight spaces. NeRN [1] shows that neural representations can be used to directly represent the weights of a pre-trained convolutional neural network. 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. More differences analysis are included in Appendix A.1 A.2 A.3.

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) [5, 46, 62, 66, 74] also learn such priors by introducing randomness to improve the robustness and generalization of neural networks. The Bayesian neural networks [20, 32, 33, 48, 58] aim to model a probability distribution over neural networks to mitigate overfitting, learn from small datasets, and estimate the uncertainty of model predictions. [21] 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. [72] combine Langevin dynamics with SGD to incorporate a Gaussian prior into the gradient. This transforms SGD optimization into a sampling process. *Bayes by Backprop* [3] 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 [17, 25, 37, 64], self-supervised learning [8, 14, 23, 24], 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 applications to parameters remain relatively underexplored. These pose a series of challenges for neural network parameter 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. This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-PhD-2021-08-008).

References

[1] Maor Ashkenazi, Zohar Rimon, Ron Vainshtein, Shir Levi, Elad Richardson, Pinchas Mintz, and Eran Treister. NeRN: Learning neural representations for neural networks. In *ICLR*, 2023.

[2] Fan Bao, Chongxuan Li, Jun Zhu, and Bo Zhang. Analytic-DPM: an analytic estimate of the optimal reverse variance in diffusion probabilistic models. In *ICLR*, 2022.

[3] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *ICML*. PMLR, 2015.

[4] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *ECCV*. Springer, 2014.

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

[6] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[7] Andrew Brock, Theo Lim, J.M. Ritchie, and Nick Weston. SMASH: One-shot model architecture search through hypernetworks. In *ICLR*, 2018.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 33, 2020.

[9] Gene Chou, Yuval Bahat, and Felix Heide. Diffusion-sdf: Conditional generative modeling of signed distance functions. In *ICCV*, pages 2262–2272, 2023.

[10] Adam Coates, Andrew Ng, and Honglak Lee. 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.

[11] Nello Cristianini, John Shawe-Taylor, et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*. Ieee, 2009.

[13] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. *NeurIPS*, 26, 2013.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[15] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *NeurIPS*, 34, 2021.

[16] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[18] Ziya Erkoç, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion. In *ICCV*, 2023.

[19] Berthy T Feng, Jamie Smith, Michael Rubinstein, Huiwen Chang, Katherine L Bouman, and William T Freeman. Score-based diffusion models as principled priors for inverse imaging. *arXiv preprint arXiv:2304.11751*, 2023.

[20] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML*. PMLR, 2016.

[21] Alex Graves. Practical variational inference for neural networks. *NeurIPS*, 24, 2011.

[22] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *ICLR*, 2017.

[23] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.

[24] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[26] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-or. Prompt-to-prompt image editing with cross-attention control. In *ICLR*, 2023.

[27] 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.

[28] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *NeurIPS*, 33, 2020.

[29] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

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

[31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[32] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[33] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *NeurIPS*, 28, 2015.

[34] Boris Knyazev, Michal Drozdzal, Graham W Taylor, and Adriana Romero Soriano. Parameter prediction for unseen deep architectures. *NeurIPS*, 34:29433–29448, 2021.

[35] Boris Knyazev, Doha Hwang, and Simon Lacoste-Julien. Can we scale transformers to predict parameters of diverse imagenet models? In *International Conference on Machine Learning*, pages 17243–17259. PMLR, 2023.

[36] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 25, 2012.

[38] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. *NeurIPS*, 7, 1994.

[39] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

[40] Alexander C Li, Mihir Prabhudesai, Shivam Duggal, Ellis Brown, and Deepak Pathak. Your diffusion model is secretly a zero-shot classifier. *arXiv preprint arXiv:2303.16203*, 2023.

[41] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[42] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *CVPR*, 2022.

[43] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440, 2015.

[44] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[45] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *NeurIPS*, 2022.

[46] Noboru Murata, Shuji Yoshizawa, and Shun-ichi Amari. Network information criterion-determining the number of hidden units for an artificial neural network model. *IEEE transactions on neural networks*, 5(6), 1994.

[47] Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces. In *ICML*, pages 25790–25816. PMLR, 2023.

[48] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[49] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

[50] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian conference on computer vision, graphics & image processing*. IEEE, 2008.

[51] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *CVPR*. IEEE, 2012.

[52] William Peebles, Ilija Radosavovic, Tim Brooks, Alexei A Efros, and Jitendra Malik. Learning to learn with generative models of neural network checkpoints, 2023.

[53] William Peebles and Saining Xie. Scalable diffusion models with transformers. *arXiv preprint arXiv:2212.09748*, 2022.

[54] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2), 2022.

[55] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *NeurIPS*, 32, 2019.

[56] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *NeurIPS*, 28, 2015.

[57] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *ICML*. PMLR, 2015.

[58] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*. PMLR, 2014.

[59] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

[60] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 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*, pages 234–241. Springer, 2015.

[61] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *NeurIPS*, 35, 2022.

[62] Wouter F Schmidt, Martin A Kraaijveld, Robert PW Duin, et al. Feed forward neural networks with random weights. In *ICPR*. IEEE Computer Society Press, 1992.

[63] Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-representations as generative models: Sampling unseen neural network weights. *NueIPS*, 35:27906–27920, 2022.

[64] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[65] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*. PMLR, 2015.

[66] Haim Sompolinsky, Andrea Crisanti, and Hans-Jurgen Sommers. Chaos in random neural networks. *Physical review letters*, 61(3), 1988.

[67] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021.

[68] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *NeurIPS*, 32, 2019.

[69] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: A simple and strong anchor-free object detector. *IEEE T-PAMI*, 44(4):1922–1933, 2020.

[70] Brandon Trabucco, Kyle Doherty, Max Gurinas, and Ruslan Salakhutdinov. Effective data augmentation with diffusion models. *arXiv preprint arXiv:2302.07944*, 2023.

[71] Laurens Van der Maaten, Geoffrey Hinton, and Laurens Van der Maaten. Visualizing data using t-sne. *JMLR*, 9(11), 2008.

[72] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *ICML*, 2011.

[73] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[74] Eugene Wong. Stochastic neural networks. *Algorithmica*, 6(1-6), 1991.

[75] 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*, pages 23965–23998. PMLR, 2022.

[76] Baoquan Zhang and Demin Yu. Metadiff: Meta-learning with conditional diffusion for few-shot learning. *arXiv preprint arXiv:2307.16424*, 2023.

[77] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018.

[78] Yifan Zhang, Daquan Zhou, Bryan Hooi, Kai Wang, and Jiashi Feng. Expanding small-scale datasets with guided imagination. *NueIPS*, 36, 2024.

[79] Andrey Zhmoginov, Mark Sandler, and Maksym Vladymyrov. Hypertransformer: Model generation for supervised and semi-supervised few-shot learning. In *International Conference on Machine Learning*, pages 27075–27098. PMLR, 2022.

[80] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017.

A Frequent Asked Questions

We list the potential frequent asked questions and the point-to-point answers as follows:

A.1 Differences between G.pt and p-diff

G.pt [52] uses diffusion process as an optimizer (from starting parameter θ to future parameter θ^* defined by their paper), so they train 23 million checkpoints (2 layer MLP with 7960 parameters and CNNs with 5370 parameters) and save their loss and error.

In G.pt, the input consists of θ, noised θ^*, and loss and error of θ^* as conditions. However, we directly generate high-performing parameters from noise.

In Fig 4 of G.pt paper, it can not achieve similar results as Adam. For example, the error of G.pt is 50 while Adam is 40. P-diff can achieve similar or even better performances than models optimized by SGD. We show the detailed comparisons between G.pt and our approach in the following Tab. 5.

items	G.pt	p-diff
target	optimizer, as an alternative for SGD and Adam	generate high-performing parameters from noise
data preparation	saving loss, error, and checkpoints during training	saving 200 converged checkpoints
input	starting checkpoints, noise, loss and error of starting and future checkpoints	random noise
performance	much worse than Adam, 10 percent drops can be found in Fig. 4 right one	similar or even better than the best results of input models
implemented architectures	MLP (2-layer, 10 hidden units); Conv + FC	BN, Conv, FC, and entire parameters that include BN, Conv, and FC
generated parameter number	less than 10, 000	2000 to 150, 000 (limited by the memory of hardware)
evaluating datasets	MNIST, CIFAR-10	MNIST, CIFAR-10/100, STL-10, Flowers, Pets, F-101, ImageNet, COCO
evaluating tasks	classification	classification, segmentation, detection, visual generation

Table 5: Main differences between G.pt and p-diff.

Can we control the performance of the generated models? G.pt [52] use the diffusion model as an optimizer. They train 23 million models and save the training loss and error as conditions. Therefore, in the inference phase, G.pt is expected to generate the model parameters with the input conditions. However, based on the results from Fig. 4 of their paper, G.pt can hardly achieve comparable results as the condition defined. There is a 10 percent drop of classification result in G.pt, indicating its poor ability to control the generated performance. To evaluate whether p-diff can accurately generate the models with defined conditions, we divide the SGD-trained models into several categories by the performances. As shown in Tab. 6, we set 4 accuracy ranges for the trained ConvNet-3 models. Then, we use one-hot vectors to encode these levels as conditions. We report the best generated results of different conditions in Tab. 6. Experimental results show that our method can precisely generate the models with different performances. This ability is crucial for real-world scenarios.

Table 6: P-diff can precisely generate models with predefined performances.

accuracy range, *i.e.*, condition	p-diff (best)
97.34±0.2	97.33
98.06±0.3	98.01
98.97±0.2	98.93
99.40±0.1	99.67

Conclusions.

- G.pt is designed for an optimizer (23 million pretrained models), while p-diff aims to be a high-performing network parameter generator (200 pretrained models).
- G.pt needs a starting parameter, error and loss of starting and target parameters, and noise. We only need noise.
- G.pt can not perform as well as their training data, while p-diff performs similarly or even better than the training data.
- G.pt only evaluates their effectiveness on classification, we show the effectiveness of classification, segmentation, detection, and visual generation.

A.2 MetaDiff vs p-diff

MetaDiff is not designed to generate high-performing parameters across an entire neural network but rather targets a specific part of it. According to their paper: "MetaDiff $\epsilon_\theta(\cdot)$ acts as a meta-optimizer, accepting the features and labels of all support samples $(u_i, y_i) \in S$ as inputs. It then learns target weights w_0 for the base learner $g_w(\cdot)$ from initial weights w_T through a denoising process (refer to Fig.2(b) from their paper)." It replaces a neural network layer with diffusion steps. However, this methodology introduces several drawbacks:

1. It substantially increases the optimization cost during training.

2. There is a significant increase in the variability of testing outcomes, stemming from uncertainties around the parameters generated in the inner loop.

In conclusion, while MetaDiff presents an innovative approach, it does not have the capability to generate high-performing neural network weights from random noise.

A.3 Detailed discussions with more related works

We also discuss the following works in detail.

[A] [63]. Hyper-Representations as Generative Models: Sampling Unseen Neural Network Weights, 2022

[B] [79]. Hypertransformer: Model generation for supervised and semi-supervised few-shot learning, 2022

[C] [34]. Parameter prediction for unseen deep architectures, 2021

[D] [13]. Predicting parameters in deep learning, 2013

[E] [47]. Equivariant Architectures for Learning in Deep Weight Spaces, 2023

[F] [35]. Can We Scale Transformers to Predict Parameters of Diverse ImageNet Models? 2023

[A] vs p-diff. [A] extends hyper-representations of model weights to sample new model weights from representation space. They build a model zoo that includes a lot of trained models. Then, an AE is trained to generate models. We use the diffusion model to learn the distribution of high-performing models and generate novel models from random noise. We also compare with [A] and find our p-diff performs better than [63].

[B] vs p-diff. [B] proposes HyperTransformer, which is used to generate models for supervised and semi-supervised tasks where labeled data are necessary. It relies on image features to generate small target CNN parameters, while p-diff generates neural network parameters from noise. The main comparison in HyperTransformer is other few-shot learning methods, e.g., MAML, and RFS. However, p-diff generates models mainly compare with SGD-trained models. HyperTransformer is designed for tasks where there are labels in the data and the neural network needs to be small CNN, while p-diff is designed for all neural networks regardless of data in tasks. Based on the above analysis, HyperTransformer is largely different from p-diff.

[C] vs p-diff. [C] (i.e. GHN-2) improves Graph Hypernetworks (GNH) [77] by updating the parameters on various GHNs and introducing normalization on the predicted parameters. Meanwhile, they also introduce a dataset named DeepNets-1M to boost the training performance.

However, GHN-2 can hardly achieve comparable results as the SGD (or other optimizers)-trained models, so it is usually regarded as an initialization method. Besides, GHN-2 is designed for NAS (neural architecture search) training. Therefore, GHN-2 predicts parameters through the given architectures and the authors mainly focus on improving the training efficiency on unseen architectures. They want to reduce the training cost of unseen architectures in NAS tasks. They do not consider too much of the diversity of generated models.

[D] vs p-diff. [D] finds that the weights of a trained network exist 95% redundancy. They only use 5% weights to predict the remaining weights. The paper also proposes that these remaining weights may not even be learned. In other words, by training only a small part (5%) of the original parameters, it is possible to achieve performance similar to or even exceeding the original network (can be regarded as a kind of regularization). 1. P-diff does not need the trained weights as inputs for predicting the other weights. 2. P-diff mainly aims at generating high-performing but novel models, not consider too much about the redundancy of parameters.

[E] vs p-diff. [E] introduces a symmetry-based approach for designing neural architectures that operate in deep weight spaces. The authors focus on what neural architectures can effectively learn and process neural models that are represented as sequences of weights and biases. However, p-diff does not focus on designing neural architectures but on generating neural network parameters from random noise.

[F] vs p-diff. [F] (i.e. GHN-3) is also an initialization method and can not achieve similar results as SGD (or other optimizers)-trained models. Its motivation is to reduce the cost of pretraining the ImageNet. The authors introduce transformer to improve the efficiency and scalability of GHNs. GHN-3 + finetuning (less epochs) can achieve comparable results as SGD-trained models. P-diff can directly achieve similar or even better results than SGD-trained models without additional finetuning. Both p-diff and GHN series (GHN, GHN-2, GHN-3) can provide initialization for network training.

A.4 Bottleneck of scaling up

We estimate the memory cost of applying AE and LDM in different numbers of generated parameter.

Setting and details:
1. We record the GPU's memory cost in models with different parameter numbers.
2. There are two stages in training p-diff, AE, and diffusion.
3. We set the batch size as 200 (the same as our paper).
4. We report the memory cost using **MB** in two stages.

Findings and conclusions:
1. As shown in Tab. 7, the cost of memory increases very fast with the number of parameters.
2. We empirically find that the memory cost of generating 100K parameters raises to the upper bound of A100 40G.
3. In our entire parameter generation (Tab. 2), the maximum number of generated parameters is 150K (MLP on CIFAR-100). We reduce the default batch size from 200 to 50. *That increases the training time a lot, so hardware is really a bottleneck in scaling up our method.*

Table 7: Bottleneck of scaling up.

parameter number (K)	AE	Diffusion
2	1475	1501
10	4473	4514
70	23377	23491
100	40380	40581

A.5 Minor over baseline

Our model is trained from SGD pretrained models, so we target to achieve similar performance as SGD. More importantly, we focus on exploring whether the generated models only memorize the training data. Like visual generation, several works [70, 78] utilize diffusion as a data augmentation

to boost the original results (as diffusion can generate something new). Here, our experiments demonstrate that p-diff can also generate some novel but high-performing models.

A.6 Main change seems to be having latent diffusion?

We show the differences between G.pt and p-diff in Tab. 5. **In technical**, our contributions can be summarized as follows:
1. Considering the difference between visual signals and neural network weights, we design 1D CNNs in the autoencoder and LDM.
2. To improve the performance stability of generated models, we introduce noise augmentation on both input and latent representations. It is worth noting that this operation is different from VAE.
3. We propose a metric for evaluating the difference between the two models.
4. We comprehensively analyze the characteristics of the generated models, including showing the differences between the original and generated models.

In the fresh views aspect:
1. P-diff can provide good initialization to speed up the training. (results can be found in Tab. 9)
2. P-diff can be a customized parameter generator. We evaluate it possibility in Tab. 6
3. Several long-way-to-go targets: can we use AI to generate AI? Can we use this technology to achieve a ModelGPT? People just need to tell the computer one sentence, ModelGPT provides the model. We believe our research can inspire more and more people to explore this area.

A.7 Other baselines

We also compare our approach with hyper-representations method [63]. We report the comparison results on MNIST, SVHN, CIFAR-10, and STL-10 datasets. To make a fair comparison, we also implement our approach using the three-layer convolutional neural network defined in hyper-representations papers [63]. As illustrated in Tab. 8, we can achieve lossless or even improved results than the original models, but the hyper-representations method can not obtain lossless results in most cases.

Table 8: Comparisons between p-diff and hyper-representations.

dataset	original	hyper-representations	p-diff	improvement
MNIST	93.7	93.8	93.9	$\uparrow$ 0.1
SVHN	78.3	78.4	78.6	$\uparrow$ 0.2
CIFAR-10	49.4	49.1	49.5	$\uparrow$ 0.4
STL-10	39.4	39.3	39.6	$\uparrow$ 0.3

A.8 Can p-diff be applied for parameter initialization?

To answer this question, we design experiments as follows. We consider two cases, the first one is partial parameter generation, and the second one is entire parameter generation. We implement our approach with ResNet-18 and ConvNet-3 on CIFAR-10 and CIFAR-100, respectively. We report the epoch number at which the model converges for evaluation. As shown in Tab. 9, our method speeds up the original by at least 5 times, which indicates the benefit of our method.

Table 9: P-diff performs good for parameter initialization.

dataset	architecture	generation mode	original	p-diff	speed up ratio
CIFAR-10	ResNet-18	partial	11 epochs	2 epochs	$\uparrow$ 5.50 $\times$
CIFAR-100	ResNet-18	partial	21 epochs	4 epochs	$\uparrow$ 5.25 $\times$
CIFAR-10	ConvNet-3	entire	100 epochs	11 epochs	$\uparrow$ 9.09 $\times$
CIFAR-100	ConvNet-3	entire	100 epochs	17 epochs	$\uparrow$ 7.69 $\times$

A.9 Details of the parameter size in this work

The parameter number in p-diff experiments is from 2000 to 150, 000 (limited by the memory of hardware). We list the parameter number of our approach in the following table.

Table 10: Parameter numbers in different architectures in our experiments.

dataset	architecture	generation mode	parameter number
CIFAR-100	ResNet-18 BN	partial	2048
CIFAR-100	ResNet-18 shortcut	partial	8192
CIFAR-100	ResNet-18 linear	partial	51200
CIFAR-10	ConvNet-3	entire	24714
CIFAR-10	MLP	entire	39760
CIFAR-100	ConvNet-3	entire	70884
CIFAR-100	MLP	entire	155135

A.10 Why access to training set when selecting generated model?

As the normal SGD training process can only access the training set, we thus select the models based on their results on the training dataset for a fair comparison. Meanwhile, the experimental results in Fig. 6(a) indicate that p-diff exhibits a high level of consistency between the training and validation sets. We use item avg. in Tab. 3 15 16 17 18 to report the average performance of the 100 novel parameters generated when evaluated directly on the test set.

A.11 In-dataset fine-tuning comparison with hyper-representations [63].

We finetune ResNet-18 normalization layers with CIFAR-10 and save finetuned weights at different epochs. We train it for 10 epochs. Then, we use weights at different epochs to train p-diff and autoencoder with 1D CNNs. Finally, finetuning the weights to the 10 epoch (same as training). We report finetune performance between original, AE + Kernel Density Estimation (KDE) , and p-diff. As shown in Tab. 11, the models initialized with generated weights learn faster and achieve better results.

Table 11: Results of in-dataset fine-tuning. '-' denotes empty (10 + 0 epoch fine-tuning).

Method/Epoch	0	1	5	10
original	13.7	40.7	90.4	92.2
AE+KDE [63]	91.2	91.9	92.3	-
p-diff	92.2	92.2	92.5	-

A.12 How about using Kernel Density Estimation(KDE) on top of the autoencoder?

We implement KDE on top of the autoencoder with our default setting that CIFAR-100 dataset and ResNet-18 backbone. We report the best, average, and median accuracy of generated models by p-diff, VAE and autoencoder with KDE sampling. Besides, we investigate the diversity of models generated by different methods. We calculate the similarity of 100 models generated by each method that performs similarly to the original models. The minimal similarity is shown in Tab. 12. AE+KDE performs better than VAE. P-diff achieves the best results on best, average, and median accuracy. Based on the similarity, p-diff generated models are more diverse than VAE and AE+KDE generated.

B Experimental Settings

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

Table 12: Performance and similarity comparisons among original, VAE, AE+KDE, and p-diff.

method	best	avg.	med.	minimal similarity
original	76.6	-	-	1.00
VAE	76.7	62.7	70.8	0.92
AE+KDE	76.7	70.3	73.4	0.93
p-diff	**76.9**	**72.4**	**75.6**	**0.85**

B.1 Training recipe

We provide our basic training recipe with specific details in Tab. 13. 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 13: Our basic training recipe based on CIFAR100 dataset and ResNet-18 backbone.

B.2 Datasets

We evaluate the effectiveness of p-diff on 8 datasets. To be specific, **CIFAR-10/100** [36]. 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** [12] 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** [10] 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** [50] is a dataset comprising 102 distinct flower categories, with each category representing a commonly occurring flower species found in the United Kingdom. **Pets** [51] includes around 7000 images with 37 categories. The images have large variations in scale, pose, and lighting. **F-101** [4] consists of 365K images that are crawled from Google, Bing, Yelp, and TripAdvisor using the Food-101 taxonomy.

Table 14: 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**

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** [41] 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.

C Explorations of Designs and Strategies

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

C.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. 14, 1D CNNs consistently outperform FC in all architectures. Meanwhile, the memory occupancy of 1D CNNs is smaller than FC.

Table 15: 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

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

Variational autoencoder (VAE) [11] 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. 16, 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.

C.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 normaliza-

Table 16: 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)

tion (GN), and instance normalization (IN) on CIFAR-10, MNIST, and CIFAR-100, respectively. We also implement our method without normalization for additional comparison. Their best, average, and medium performances of 100 generated models are reported in Tab. 15. 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. 15. 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 17: 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

D 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.

D.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. 17(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.

D.2 The intensity of noise added into latent representations

Similar to Sec. D.1, we also ablate the noise intensity added into latent representations. As shown in Tab. 17(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.

Table 18: 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

D.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 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. 17, we find our approach consistently achieves similar or improved performance compared to the original models.

E Open Explorations

E.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. 18. 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.

E.2 Potential applications

Neural network parameter 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.

F Other Finding and Comparison Results

F.1 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. 5. 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.

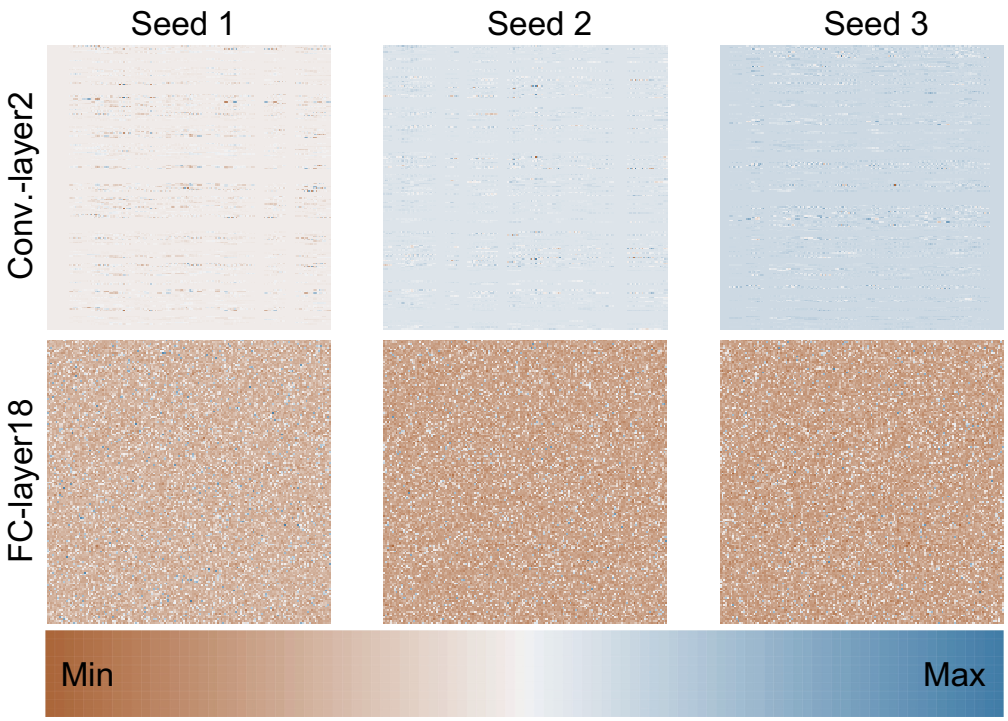
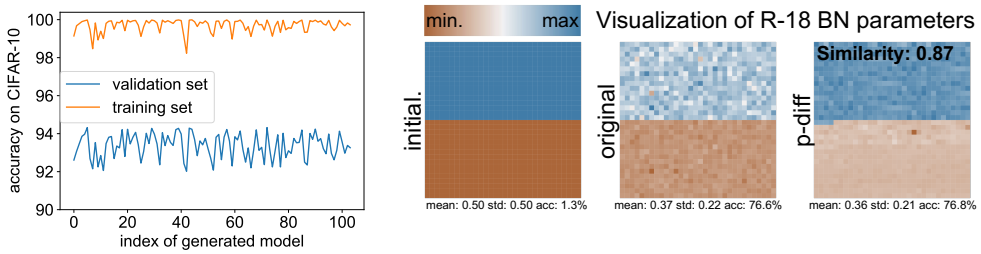


Figure 5: Visualizing the parameter distributions of convolutional (Conv.-layer2) and fully connected (FC-layer18) layers. Parameters from different layers show variant patterns while these parameters from the same layer show similar patterns. The index of layer is aligned with the standard ResNet-18.

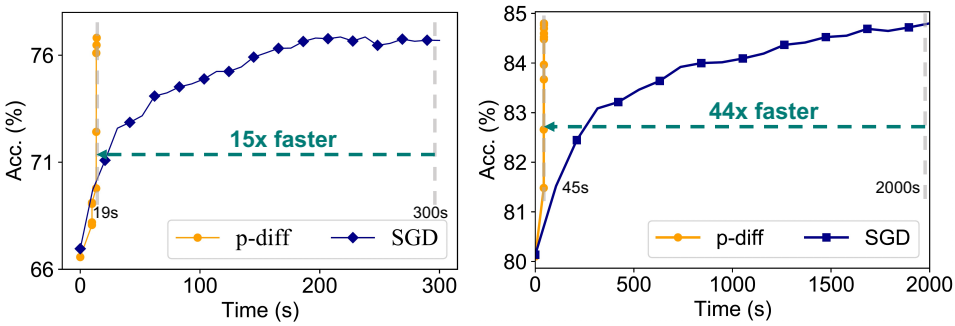


(a) Accuracy distribution in p-diff models. (b) Visualization of initial, SGD-trained, p-diff generated model.

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.

F.2 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.



(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.

F.3 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 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.

F.4 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.

G 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.

G.1 Object detection

Faster R-CNN [56] 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) [69] 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 [56] and FCOS [69] with ResNet-50 backbone on the COCO [41] 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. 19. 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 19: P-diff in object detection task. We report the mAP of best original model and best p-diff generated model.

G.2 Semantic segmentation

Fully Convolutional Network (FCN) [43] 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 [43] 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. 20. 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 20: P-diff in semantic segmentation task. We report mean IoU and pixelwise accuracy of best original model and best p-diff model.

G.3 Image generation

DDPM [28] is a diffusion-based method in image generation, where UNet [60] 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. 21. 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 21: P-diff in image generation task. We report the FID score on the CIFAR-10 dataset.