Scaling Up Parameter Generation: A Recurrent Diffusion Approach

Kai Wang*
National University of Singapore
kai.wang960112@gmail.com

Dongwen Tang*
National University of Singapore
mtdovent@gmail.com

Wangbo Zhao
National University of Singapore
wangbo.zhao96@gmail.com

Konstantin Schürholt
University of St. Gallen
konstantin.schuerholt@unisg.ch

Zhangyang Wang†
University of Texas at Austin
atlaswang@utexas.edu

Yang You†
National University of Singapore
chinayouyang1991@gmail.com

Abstract

Parameter generation has long struggled to match the scale of today's large vision and language models, curbing its broader utility. In this paper, we introduce ***R**ecurrent Diffusion for Large-Scale **P**arameter **G**eneration (**RPG**)*, a novel framework that generates full neural network parameters—up to **hundreds of millions**—on a **single GPU**. Our approach first partitions a network's parameters into non-overlapping 'tokens', each corresponding to a distinct portion of the model. A recurrent mechanism then learns the inter-token relationships, producing 'prototypes' which serve as conditions for a diffusion process that ultimately synthesizes the parameters. Across a spectrum of architectures and tasks—including ResNets, ConvNeXts and ViTs on ImageNet-1K and COCO, and even LoRA-based LLMs—RPG achieves performance on par with fully trained networks while avoiding excessive memory overhead. Notably, it generalizes beyond its training set to generate valid parameters for previously unseen tasks, highlighting its flexibility in open-ended scenarios. By overcoming the longstanding memory and scalability barriers, RPG serves as a critical advance in '**AI generating AI**', potentially enabling efficient weight generation at scales previously deemed infeasible.

1 Introduction

Scaling up neural networks has been one of the most crucial factors driving the progress of deep learning, enabling remarkable performance across a wide range of tasks [34, 25, 45]. However, *neural network parameter generation*—from early HyperNetworks [24] to more recent diffusion-based methods [50, 68, 63]—has not kept pace with the expansion of mainstream vision and language models. Indeed, as illustrated in Fig. 1, there is an astonishing scale gap of at least 10^3 between the parameter counts of widely used models (*e.g.,* ViT, ConvNeXt, LLaMA) and the largest parameter generators available today. This discrepancy not only underscores the challenge of *scalability* but also confines existing generators to academic demonstrations rather than real-world applicability.

*equal contribution, †corresponding author
Code: https://github.com/NUS-HPC-AI-Lab/Recurrent-Parameter-Generation

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

Traditional approaches such as HyperNetworks struggle to handle even moderately sized models, primarily due to memory overheads and optimization complexity. While recent diffusion-based works attempt to break the memory bottleneck via partial-parameter or two-stage generation [68, 63], such methods may overlook key correlations among different parameter subsets.

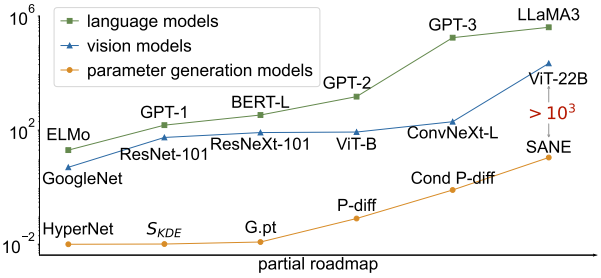


Figure 1: Roadmap of vision, language, and parameter generators. Parameter number in vision or language models is $\mathbf{10^3}$ times larger than that of generated parameters.

To illustrate the importance of capturing inter-part correlations, we conduct a simple experiment: we train two ViT-Tiny [15] models (A and B) on CIFAR-10 [33], then exchange half of their layer parameters. Despite both models performing well originally (*i.e.,* 90.4% for model A and 89.6% for model B), the hybrid model's accuracy drops dramatically to 45.8% (as seen below).

model	ViT-Tiny A	partial A + partial B	ViT-Tiny B
accuracy (%)	90.4	45.8	89.6

This striking performance gap highlights that adequately modeling parameter correlations is not a trivial detail: a robust generator must therefore *explicitly model* such relationships while also surmounting the memory pitfalls of large-scale processing.

Our contributions. We present *Recurrent Diffusion for Large-Scale Parameter Generation (RPG)*, a novel, end-to-end framework capable of generating *full* neural network parameters with up to **hundreds of millions** of weights using just a **single commodity GPU**. To the best of our knowledge, RPG is **the first** method to efficiently synthesize the parameters of models like ConvNeXt-L and LLaMA-LoRA–with up to ∼**200M** weights–in a **single pass**. This significant leap is achieved by two core technical innovations:

1. **Parameter processing for large-scale tokenization.** We devise a highly customized *parameter tokenization* strategy designed to preserve both the *layer-wise distribution* and the *cross-layer correlations*. Specifically, we (i) split weights according to their layer indices, (ii) apply layer-wise normalization to mitigate distribution shifts, (iii) slice parameters into non-overlapping *tokens* with uniform size, and (iv) introduce a lightweight *permutation state* to alleviate symmetry issues when collecting multiple checkpoints. Additionally, we employ *2D position embeddings* (layer index + token index) to ensure the network retains positional awareness of each token within the entire set. This tailored pre-processing pipeline is crucial for learning stable and high-quality parameter representations.
2. **Recurrent diffusion modeling.** Unlike prior diffusion-based generators that flatten or divide all parameters into a single sequence or chunks, we propose a *recurrent* mechanism to learn the relationships among tokens. Concretely, we map each token into a hidden space via a *recurrent model*, whose outputs serve as *prototypes*. These prototypes then condition a 1D *diffusion process* that progressively denoises the parameter tokens. By recurrently modeling global inter-token interactions and leveraging a diffusion sampler for synthesis, RPG aligns well with the dual goals of capturing token-to-token dependencies and remaining memory-efficient.

Key results and impact. Extensive evaluation on Image-Net classification, ADE20K segmentation, COCO detection, and commonsense reasoning show that RPG consistently produces weights *on par* with trained models. Notably:

- **Single-GPU feasibility.** Our recurrent design and parameter-token strategy enable inference on a single commodity GPU beyond 100M weights.
- **Generalization to unseen tasks.** RPG can generate high-performing neural network parameters for *unseen* binary classification tasks on CIFAR-10, providing evidence for its broader generalization capabilities.
- **Architectural versatility.** Our method supports diverse families such as ResNet, Vision Transformer, ConvNeXt, and LoRA-based language models, making it a unified framework for parameter generation.

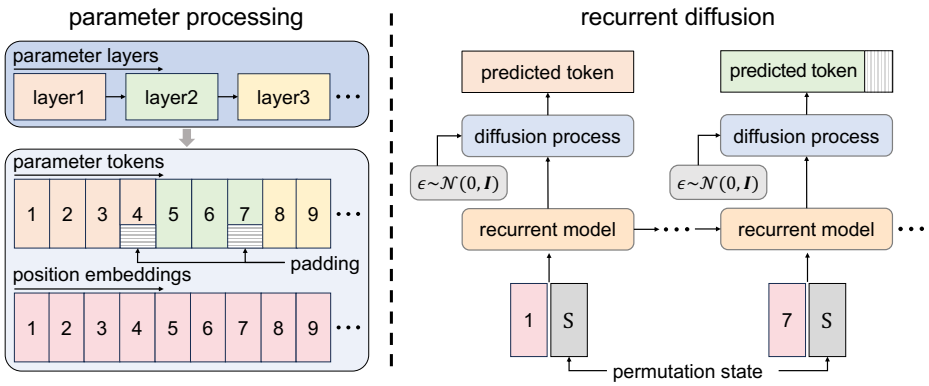


Figure 2: Illustration of **parameter processing** (left) and inference of **recurrent diffusion** (right). The recurrent model integrates permutation states and position embeddings, generating prototypes that condition the diffusion model to synthesize the full parameters.

These results place us significantly closer to the vision of '*AI creating AI*', where a single generative model can synthesize entire networks tailored for different tasks. We believe RPG's strong scalability, lightweight memory footprint, and ability to accurately capture global parameter relationships may open new frontiers in fields like rapid architecture search, efficient model adaptation, and beyond.

2 Method: Large-Scale Parameter Generation

In this section, we describe our proposed *Recurrent Diffusion for Large-Scale Parameter Generation (RPG)*, which synthesizes *full* neural network parameters under strict memory constraints. As illustrated in Fig. 2, our approach comprises two primary components: **parameter processing** and **recurrent diffusion**.

Parameter processing (Sec. 2.2) equips the model with a specialized tokenization pipeline, layer-wise normalization, permutation states, and position embeddings—all meticulously designed to mitigate issues like distribution shifts, neural symmetry, and the sheer size of modern network weights. **Recurrent diffusion** (Sec. 2.3) then learns these 'tokens' via a recurrent model whose outputs condition a 1D diffusion process, enabling us to capture *global* parameter correlations while maintaining feasibility on a single GPU.

2.1 Overview

Our pipeline for large-scale parameter generation is shown conceptually on the right of Fig. 2. Given a set of trained parameters $\{W\}$ (e.g., from various checkpoints or model variants), we first transform them into *parameter tokens*, augment them with a *permutation state*, and apply *position embeddings*. This processed sequence is fed into a *recurrent model*, which outputs per-token 'prototypes'. Finally, a *diffusion model* uses these prototypes, along with random noise, to synthesize the entire parameter vector. In practice, we can sample from this generative model to produce a diverse family of high-performing weights.

2.2 Parameter Processing

Parameter tokenization. Drawing inspiration from the success of language and vision transformers [67, 15], we divide a network's parameters into a sequence of non-overlapping *tokens*. However, direct flattening or naive partitioning often fails to preserve crucial *layer-wise* statistics and undermines global consistency. To circumvent this, we first segregate parameters by layer and then *normalize* each layer to reduce distribution shifts:

$$W_n \xrightarrow{\text{divide by layer}} [w_n[1], \ldots, w_n[I]] \xrightarrow{-\mu_i, /\sigma_i} [\hat{w}_n[1], \ldots, \hat{w}_n[I]], \tag{1}$$

$$\mu_i = \frac{1}{N}\sum_{n=1}^{N} \text{mean}(w_n[i]), \sigma_i = \frac{1}{N}\sum_{n=1}^{N} \text{std}(w_n[i]). \tag{2}$$

Where $w_n[i]$ and $\hat{w}_n[i]$ denote the original and normalized parameter blocks of the i-th layer of the n-th checkpoint in the training set, respectively. The normalization is performed using layer-wise

statistics μ_i and σ_i, which represent the mean and standard deviation of the weights in layer i, averaged across the entire training set. These statistics are previously computed and stored and later employed during the denormalization process to recover the generated parameters.

Next, each normalized layer $\hat{w}[i]$ is *tokenized* into contiguous chunks of uniform size k (plus padding if necessary):

$$\hat{w}[i] \xrightarrow{\text{tokenize}} K[i] = \left[k_i^1, k_i^2, \ldots, \text{padding}\left(k_i^J\right)\right]. \tag{3}$$

Importantly, the padded regions are excluded from the loss and do not affect gradient updates. By enforcing a consistent token size across layers, we facilitate *batch processing* of parameter tokens—a critical step for large-scale training.

Permutation state. When aggregating checkpoints from various training runs, *neural symmetry* [35] can become a major bottleneck. Essentially, distinct permutations of layer weights may yield identical final performance but appear different to a generative model. To resolve this ambiguity, we introduce a distinct *permutation state* S—encoded as a one-hot vector—for each checkpoint W. This extra condition guides the generative model to treat otherwise symmetrical weight configurations as unique exemplars, significantly stabilizing distribution learning.

Specifically, in the training phase, the permutation state (*i.e.*, a one-hot vector) is first processed by an MLP and then added to the model input, thereby explicitly representing the checkpoint's symmetric state. In the inference phase, we typically only care about the model's performance rather than the model's symmetric state. In this case, we can simply sample a permutation state at random. Alternatively, we can specify a particular permutation state to generate models under a specific symmetric state.

Position embedding. To preserve token locality and inter-layer relationships, we augment each token with a **2D sinusoidal** position embedding [15]. Specifically, for the j-th token of layer i, we assign

$$K[i] \xrightarrow{\text{pos. embedding}} e[i] = [\, e_i^1, \ldots, e_i^j, \ldots, e_i^J \,]. \tag{4}$$

The first dimension encodes the layer index, while the second dimension captures the token's in-layer position. As a result, the generative model knows not only how tokens are grouped by layer, but also each token's placement within that layer.

2.3 Recurrent Diffusion

Recurrent model. After obtaining the parameter tokens $K[i]$, permutation states S, and position embeddings $e[i]$, we feed them into a *recurrent* network $f(\cdot)$ that learns token-wise representations while capturing cross-token dependencies. We denote the output as a *prototype* P_i^j:

$$P_i^j,\, H_i^j \;=\; f\Big(H_i^{j-1},\, e_i^j,\, S\Big), \tag{5}$$

where H_i^j is the internal hidden state for the j-th token in layer i. In practice, we use Mamba [23] (a memory-efficient transformer alternative) followed by an MLP that projects features to the required dimension for our diffusion model. We find that a recurrence-based design elegantly handles long sequences of tokens, offering a strong balance between model capacity and memory usage. Later in experiments, we compare different recurrent architectures such as LSTM [29] and causal-decoder transformers [67].

Parameter diffusion. Finally, we leverage a diffusion model to generate the actual parameter values. Drawing on p-diff [68] and MAR [36], we construct a 1D convolutional network that accepts random noise ϵ together with the prototypes P. Formally, at diffusion time step t, we aim to predict the noise ϵ in a denoising objective:

$$L_{\text{diff}} \;=\; \mathbb{E}_{t,\,K,\,\epsilon}\big\|\,\epsilon \;-\; \epsilon_\theta\big(K_t,\; t,\; P\big)\big\|^2, \tag{6}$$

where $\epsilon_\theta(\cdot)$ is our denoising network with learnable parameters θ, and K_t is the noisy version of K at step t. Critically, the recurrent prototypes P act as *conditional inputs*, enforcing global coherence among tokens. Gradients from this diffusion loss flow back into the recurrent model, implicitly learning token correlations as well.

Inference. Once training is complete, the pipeline supports two inference modes. Basically, in conditional generation, we feed the permutation state S of an existing checkpoint to replicate or

perturb known parameter configurations. Further more, we can sample entirely **unseen task states** to produce novel parameter draws (Sec. 4). In both cases, the recurrent model outputs prototypes, and the diffusion model denoises random noise into a final parameter set that *can scale up to hundreds of millions of weights*.

Overall, **recurrent diffusion** elegantly decouples the complex problem of *global parameter correlation* into (i) a recurrent pass that focuses on cross-token interactions, and (ii) a diffusion pass that refines each token. This two-step design is precisely what allows RPG to handle large-scale architectures on standard hardware.

3 Main Experiments

3.1 Setup

Datasets and architectures. We evaluate our method across various tasks, including ImageNet-1K [13] for theclassification, ADE20K [77] for the semantic segmentation, COCO [40] for the object detection, and BoolQ [10], PIQA [5], SIQA [54], HellaSwag [75], and ARC [11] for the commonsense reasoning tasks. To verify the scalability, we conduct experiments on various architectures with parameter counts ranging from several to hundred million.

Trained parameters collection. We take parameters collection on the ImageNet-1K as an example. To save the cost, we finetune the *full parameters* of the models in timm [1] and save 50 checkpoints as the training data. (More details are in Appendix C.6) For each checkpoint, we assign a unique permutation state to guide the generated parameters.

Preprocessing and training details. The length of parameter tokens, permutation states, position embeddings, and prototypes is set to 8192. It is worth noting that the permutation states and position embeddings are fixed during the training. We also study the influence of the token length, varying it from 1024 to 16384. The parameter diffusion consists of 1D convolutional layers. We default to using Mamba [23] as the architecture of the recurrent model. More details can be found in Appendix B.

Inference details. We input permutation states (relevant experimental results are in Appendix C.1) and position embeddings into the recurrent model to generate the prototypes. Then, the diffusion model utilizes the prototypes as conditions, along with random noises, to synthesize the entire network parameters. We repeat the above process 10 times and report the best, average, and minimum.

3.2 Results of Large-Scale Parameter Generation

In this section, we present the results of our approach across a range of tasks including classification, semantic segmentation, object detection & instance segmentation, and language tasks. As most previous works encounter the out-of-memory issue at million-scale parameter generation, we mainly compare with the results from the trained networks, which we denote as 'original'.

On ImageNet-1K. Tab. 1 presents performance comparisons across seven architectures on ImageNet-1K. The parameter number of these architectures ranges from 3 to 197 million. Several observations can be made as follows: i) Our approach successfully generates model parameters at hundred-million scales, overcoming the out-of-memory issues faced by previous works [50, 68]. ii) The performances of the generated models are comparable with the original ones.

arch. \ acc. (%)	ResNet-18	ResNet-50	ViT-Tiny	ViT-Small	ViT-Base	ConvNeXt-A	ConvNeXt-L
parameters (M)	11.7	25.6	5.7	22.1	86.6	3.7	197.8
original	70.0	79.8	74.9	81.4	84.4	75.2	85.8
best	69.9	79.6	75.4	80.6	84.6	74.6	85.8
average	69.5	79.5	75.3	80.5	84.4	74.4	85.5
minimum	69.0	79.4	75.2	80.1	84.2	74.2	85.2

Table 1: We compare with the results of original models across seven architectures on the ImageNet-1K. Our approach successfully generates the entire model parameters that perform comparable results with the original models.

On ADE20K and COCO. For semantic segmentation, following [76], we adopt UperNet [71] as the segmentation model and train it on ADE20K [77] to prepare checkpoints. For object detection and instance segmentation, we finetune ViTDet [37] on COCO [40] to collect checkpoints and report

[1] https://github.com/huggingface/pytorch-image-models

the results of mAP Bbox and mAP Seg, respectively. All experiments here are conducted based on ViT-Base [15]. Tab. 2 presents the strong generalization of RPG to these two tasks. Specifically, compared to the original models, we achieve comparable or even slightly better results over all the above metrics.

method	ADE20K		COCO	
	mIoU(%)	mAcc(%)	mAP Bbox (%)	mAP Seg (%)
original	47.6	58.3	43.6	39.0
ours	47.1	57.5	44.5	39.6

Table 2: Accuracy comparison of original and generated parameters on ADE20K (176.5M) and COCO (110.9M). All models are built based on ViT-Base.

On commonsense reasoning. We employ DoRA [41], an upgraded version of LoRA [30], to fine-tune LLaMA-7B [65] for commonsense reasoning tasks and save the checkpoints as the training data. We report the results across 7 sub-tasks with rank = 4 and 64 in Tab. 3. The generated models consistently yield results comparable to those of the original ones.

rank	params. (M)	BoolQ		PIQA		SIQA		HellaSwag		ARC-e		ARC-c		OBQA	
		org	RPG	org	RPG	org	RPG	org	RPG	org	RPG	org	RPG	org	RPG
4	7.8	**64.3**	63.1	71.3	**72.0**	66.0	**67.5**	53.7	**56.7**	64.4	**65.3**	49.5	**49.7**	63.1	**66.0**
64	113.1	**69.3**	69.1	78.9	**79.4**	72.9	**73.9**	81.1	81.1	72.9	**73.1**	58.1	**58.3**	71.2	**72.1**

Table 3: Accuracy comparisons of original and generated DoRA with varying ranks for LLaMA-7B on the commonsense reasoning tasks. Our approach can generate comparable or even better results than original models. **Bold entries** are best results.

3.3 Ablation Studies, Analysis, and Comparison

We present the results of the generated ViT-Tiny on the ImageNet-1K, unless stated otherwise.

The effect of recurrent model. We employ the recurrent model to learn the relationship among parameter tokens. To keep other factors consistent, we simply remove the state transition function from the recurrent model for comparison, denoted as '– recurrent model'. The experimental results from Tab. 4a confirm that the recurrent model plays a crucial role in parameter generation. Without the state transition function, our approach learns each parameter token individually, overlooking the relationships among these tokens. As a result, the generated parameters perform extremely poorly.

The manner of position embeddings. In ViT, the position embeddings are learnable by default. Here, we mainly conduct the experiments with three different position embedding manners and show the details as follows:

- learnable: Initializing with 2D sinusoidal positional encoding and set to be learnable.
- encoded by index: Using 1D sinusoidal positional encoding, irrespective of the original network structure, with indices assigned from front to back.
- encoded by layer: Using 2D sinusoidal positional encoding to represent layer and token indices.

As shown in Tab. 4b, the learnable embeddings perform slightly better than the other two manners. However, we still recommend using fixed position embeddings, as they offer comparable performance while significantly reducing storage requirements compared to the learnable one.

The manner of tokenization. Considering the differences among various layers, we divide the parameters into tokens within each layer. P-diff [68] flattens the parameters into 1D vectors, while SANE [57] divides the parameters by channel within each layer. We compare the results of these 3 strategies in Tab. 4c. Our default strategy achieves better results than the others. Flattening results in a single token containing parameters from different layers, which poses challenges for optimization. Tokenizing by the channel may result in excessive padding values for each token, as the channel number is usually much smaller than the token size.

The impact of token size. Token size directly affects model capacity, as larger tokens can encode richer information. In Tab. 5, we compare the performance of models generated by RPG with token sizes ranging from 1024 to 16384. Model performance generally improves with larger token sizes, since smaller tokens provide insufficient information for effective learning; however, excessively large tokens lead to significant memory overhead, as detailed in Fig. 8a in the Appendix.

Why not auto-regression? We adopt a standard recurrent paradigm with Mamba, rather than an auto-regressive approach. This means the generated parameter tokens are not used as inputs for subsequent token predictions. Our method takes the position embedding and permutation state as inputs and synthesizes neural network parameters as outputs, forming a standard recurrent neural network. To investigate the impact of different sequential modeling approaches, we compare auto-regressive and

	best	avg.	min.
original	75.2	74.9	74.7
− recurrent model	fail	fail	fail
+ recurrent model	**75.4**	**75.3**	**75.2**

(a) Recurrent model can largely improve the performance and stability.

pos. emb.	best	avg.	min.
learnable	**75.5**	**75.4**	**75.3**
by index	75.4	75.3	75.0
by layer	75.4	75.3	75.2

(b) Learnable performs better, but saving cost needs to be considered.

tknz.	best	avg.	min.	time (h)
flatten	75.3	75.2	74.8	6.2
channel	75.3	75.1	75.0	14.2
layer	**75.4**	**75.3**	**75.2**	6.2

(c) Our tokenization achieves the best trade-off between acc. and time.

Table 4: Ablation experiments of the recurrent model, position embeddings, and tokenization with ViT-Tiny on ImageNet-1K. Defaults are marked in gray. 'fail' indicates models performing at random-chance level. **Bold entries** are best results.

arch. \ acc. (%)	token size				
	1024	2048	4096	8192	16384
ViT-Tiny	0.3	70.8	75.2	**75.3**	69.3
ViT-Small	0.1	0.7	80.5	80.5	80.4
ViT-Base	0.1	0.1	0.2	45.3	**84.4**

Table 5: Accuracy of generated models with the different toke sizes. Large tokens perform better on large models. **Bold entries** are best results.

method	best	avg.	min.
original	75.2	74.9	74.7
AR (transformer decoder-only)	fail	fail	fail
recurrent (transformer encoder-only)	75.0	74.8	74.6

Table 6: Auto-regressive (AR) fails to generate meaningful results due to accumulated errors. More details are provided in Appendix C.9.

structure	best	avg.	min.	time (h)	memory (GB)
LSTM	**75.5**	75.2	74.4	16.1	38.1
Trans.	75.0	74.8	74.6	4.2	29.1
Mamba	75.4	**75.3**	**75.2**	**4.1**	**27.8**

Table 7: We study the characteristics of three recurrent structures. Defaults are marked in gray. **Bold entries** are best results.

scales \ params.	RPG-T	RPG-S	RPG-B	RPG-L
generated model	∼0.05	∼10	∼50	∼200
recurrent part	1.3	256	1018	3076
denoising part	0.3	17	69	273

Table 8: RPG components and generated parameter counts across scales. '∼' denotes 'approximate'. All values are in millions.

recurrent formats. For a fair comparison, we use transformer architectures in both cases: a recurrent format (using an encoder-only transformer) and an auto-regressive format (AR, using a decoder-only transformer). As shown in Tab. 6, the AR approach performs poorly due to error accumulation during the inference stage. More detailed analysis can be found in Appendix C.9.

The structure of recurrent model. We mainly explore three structures of the recurrent model, including LSTM [29], Transformer [67], and Mamba [23]. In Tab. 7, we report the performances, training time, and memory cost of generating ViT-Tiny parameters on ImageNet-1K. All three structures can achieve good results. However, considering the training time and memory cost, our default Mamba is the best structure of the recurrent model.

RPG parameters vs. generated parameters. We analyze the number of parameters of RPG's recurrent and denoising components, and the number of parameters they can generate. We design four RPG variants to generate models of different sizes, as summarized in Tab. 8. (See Tab. 16 in the Appendix for architectural details.) Overall, the recurrent model contains most of the parameters, while the denoising network is kept small to speed up generation, as it is applied repeatedly. This design enables RPG to achieve high efficiency while generating large-scale parameters. Moreover, once trained, RPG can produce a virtually unlimited number of distinct models, offering substantial long-term efficiency benefits.

Efficiency of generating large-scale parameters. Rapid synthesis of large-scale parameters is crucial for evaluating the practicality of our approach. As illustrated in Tab. 9, we present the time cost for generating models of ViT-Base and ConvNeXt-L across various DDIM [61] sampling steps. All results are obtained with a single NVIDIA H100 80G GPU. Our approach shows the capability to generate models within minutes. Notably, even for ConvNeXt-L (197.7 M parameters), we can synthesize the entire parameter within 1.3 minutes. Even with only 20 sampling steps, we can achieve promising results. Meanwhile, the inference memory requirement is approximately 20GB, so RPG can be deployed on NVIDIA RTX 3090 or similar-level GPUs.

model / memory	ViT-Base / 20.7GB				ConvNeXt-L / 21.6GB			
diffusion steps	20	60	100	200	20	60	100	200
time (minute)	0.6	0.8	1.1	1.8	0.7	1.3	2.0	3.5
accuracy (%)	83.3	84.4	84.4	84.3	85.0	85.5	85.3	85.3

Table 9: GPU memory and inference time comparisons among diffusion steps. RPG can efficiently generate the entire ConvNeXt-L (197.8M parameters) in minutes with only ∼20GB of GPU memory.

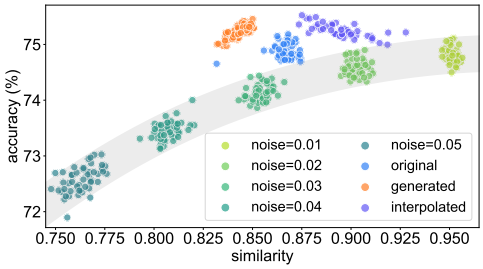


Figure 3: The figure shows the trade-off between accuracy and similarity with ViT-Tiny on ImageNet-1K. The shaded area includes the approximate range of noise-added checkpoints. This plot demonstrates the strong trade-off between accuracy and similarity and highlights our advantages over trivial interpolation.

method	CNN (s)	CNN (m)	R-18	ViT-B
params. (M)	0.003	0.011	11.7	86.6
S_{KDE30}	26.9 **46.1**	-	OOM	OOM
p-diff	<u>48.8</u> **49.0**	<u>61.9</u> **62.1**	OOM	OOM
SANE	-	57.9 **57.2**	68.6 **85.5**	-
D2NWG	38.2 **44.7**	58.8 **57.2**	94.6 **94.6**	-
RPG	49.0 **49.0**	62.0 **62.1**	95.1 **95.3**	98.9 **98.7**

Table 10: Our approach obtains the best results across all architectures and largely outperforms most previous works. The **subscripts** denote the average accuracy of corresponding original models. <u>Underline</u> denotes the reproduced results. And OOM denotes out-of-memory issue. The detailed structures of CNN (s) and CNN (m) can be found in Model Zoos [58].

Similarity analysis of generated models. We compare RPG with adding noise and model interpolation. Following p-diff [68], we use IoU to measure model similarity by comparing outputs across samples, selecting the maximum IoU with original models as our metric. In Fig. 3, as the noise level increases, both the similarity and accuracy of the models decrease. The points representing our generated models are distributed in the *upper left* region relative to the other conditions, indicating that our models can enhance diversity while maintaining accuracy. Furthermore, our method demonstrates greater diversity in comparison to trivial interpolation methods. Furthermore, we examine how similarity varies with the number of training checkpoints and analyze the diversity of symmetric states using linear mode connectivity [19, 74]; see Appendix C.1 and C.6 for details.

Comparisons with previous methods. We compare RPG with four previous works, *i.e.*, S_{KDE30} [55], p-diff [68], D2NWG [63], and SANE [57]. We report the original and generated performances for comparison. As shown in Tab. 10, among all compared methods, only RPG consistently achieves the highest results and comparable results to original models across various architectures. Another key issue is that the previous works usually fail to generate large-scale neural network parameters. (More discussion is in Appendix A.)

4 RPG Generalizes to Unseen Tasks

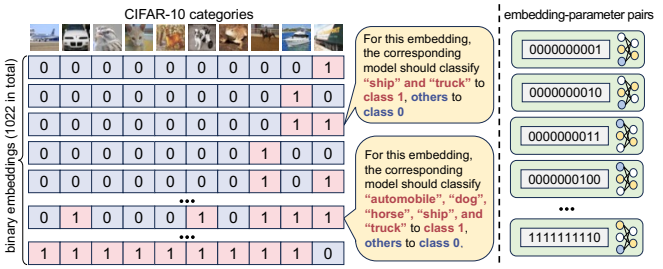
Until now, experimental results have demonstrated that our approach can efficiently generate large-scale neural network parameters if these models are included in the training set. In this section, we mainly investigate whether our approach has the ability to generate models to tackle unseen tasks. This generalization capability is crucial as it enables RPG to handle novel tasks without requiring additional training, making it particularly valuable for real-world applications. Such capabilities are essential towards generating high quality models on the fly, and could be used, e.g., for model personalization or to adjust for distribution shifts, new concepts, or new tasks by generating targeted weights.

4.1 Experiment Designs.

Build seen and unseen tasks. To assess RPG's capability for unseen tasks, we construct various binary classification tasks on CIFAR-10. As shown in Fig. 4, we encode each task as a 10-bit binary embedding, where each bit corresponds to a category. 1 indicates that the corresponding category should be classified as positive (class 1), while 0 indicates negative (class 0). For example, in the third embedding shown, 'ship' and 'truck' are assigned to class 1, while all other categories belong to class 0. Given this encoding strategy, we can create 2^{10} binary embeddings. After removing the two trivial cases (all 0s and 1s), we obtain 1022 valid embeddings. For each one, we collect its corresponding parameters, forming embedding-parameter pairs. These pairs are then split into *non-overlapping* sets for training and validation, allowing us to evaluate RPG's generality on unseen tasks.

Collection of the checkpoints. We use ViT-Tiny to train 1022 binary classifiers on CIFAR-10 with different binary embeddings and save 3 models for each classifier. These binary embeddings serve as conditioning inputs for the subsequent RPG training process. Of these tasks, 1002 randomly selected embedding-parameter pairs serve as the training set (seen tasks), while the remaining pairs are reserved as unseen tasks for evaluation.

Figure 4: Left: binary embeddings encode different CIFAR-10 classification tasks, where 1s indicate classes to be classified together (e.g., 'ship' and 'truck' in the first example). Right: parameter-encoding pairs, formed by network parameters with their corresponding binary embeddings.



Training and evaluation of RPG. We only use the 1002 checkpoints from seen tasks to train RPG. Meanwhile, these embeddings are also fed into RPG as conditional inputs of the recurrent model. During the training stage, the checkpoints trained by the unseen binary embeddings are not accessible. When evaluating, we input the unseen binary embeddings to the trained RPG to generate the parameters for unseen tasks. The results of the original and generated unseen models are reported for comparison.

4.2 Results for Unseen Tasks

Performance comparisons. We compare the results of RPG and original models on unseen binary embeddings in Tab. 11. We randomly select ten unseen binary embeddings for comparison. Notably, RPG yields commendable performance in these unseen tasks, even without being trained on the specific unseen embeddings. That demonstrates the strong practicality and potential of our approach in generating models under unseen tasks. The results of the remaining unseen tasks and **other datasets (ImageNet-1K and PASCAL VOC)** are shown in Appendix C.2.

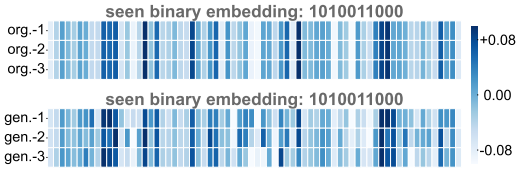
unseen tasks (embeddings)	original	RPG
0 1 0 0 0 1 0 1 1 1	97.3	94.4
0 1 1 1 1 1 0 1 1 0	98.1	96.6
0 0 1 1 1 0 1 1 1 0	97.4	95.0
0 1 0 1 1 1 1 1 1 1	98.4	96.1
0 0 1 0 0 0 0 0 0 0	98.9	96.6
0 0 0 1 1 0 0 1 0 1	96.7	92.9
1 1 1 1 1 0 1 0 0 1	97.6	94.8
1 0 1 0 0 0 0 0 1 1	98.1	95.7
0 1 0 0 0 1 0 1 1 0	97.1	93.6
1 1 0 0 0 1 1 0 0 1	97.0	94.0

Table 11: Result comparisons between original and generated models on unseen embeddings (tasks). Even without seeing the trained parameters of unseen tasks, RPG directly generates parameters that achieve competitive performance.

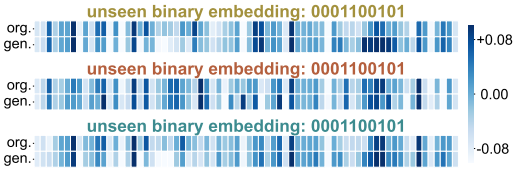
Perception of shifts in parameter distributions. In addition to comparing results, we further investigate RPG's ability to perceive shifts in parameter distributions. We select two tasks whose binary conditions differ in each element—effectively representing opposite distributional configurations—and report the results in Tab. 12. Our approach demonstrates a remarkable capacity to accurately detect such distribution changes and generate corresponding model parameters. It is worth noting that the accuracy would hover around 50% if RPG were insensitive to these distributional shifts.

binary embedding (from seen set)	1	0	1	1	1	0	1	0	0	0
accuracy (%)	98.0	99.1	98.5	94.4	98.1	92.2	99.2	97.0	97.1	99.1
flipped embedding (from unseen set)	0	1	0	0	0	1	0	1	1	1
accuracy (%)	92.3	98.9	94.0	85.4	92.2	90.8	99.3	95.3	97.6	98.1

Table 12: Comparisons of parameter distribution shifts. The accuracy is reported for each individual class. RPG can be aware of such distribution shifts accurately.



(a) Original and generated models with identical seen binary embeddings are compared. The three original models exhibit homogeneity, while the generated models display diversity and maintain high accuracy across all three models.



(b) Original and generated models with three unseen binary embeddings are compared. The results confirm that our approach can learn high-performing parameter patterns (similar distribution to original models) even when they are not included in the training set.

Figure 5: Illustration of the parameters of original and generated models in seen and unseen embeddings. We select 100 parameters of the classification head and visualize its normalized values.

Visualizations of model parameters. We visualize the original and generated models for both seen and unseen tasks in Fig. 5. For seen tasks, our approach generates diverse models compared to the original ones. Surprisingly, as shown in Fig. 5b, we find that our approach can learn unseen parameter patterns. This demonstrates the potential generalization ability of our method.

Efficiency comparison. To evaluate efficiency on unseen tasks, we compare three approaches: (i) training from scratch, (ii) fine-tuning a ViT-Tiny model pretrained on ImageNet-1K, and (iii) fine-tuning a model initialized with RPG. As shown in Table 13, RPG initialization dramatically accelerates convergence on the unseen task. These results demonstrate that RPG provides a highly effective initialization strategy, substantially reducing training time and computational costs.

epoch	from scratch	finetune	RPG init + finetune
0	50.0	52.9	94.4
1	61.4	90.3	96.8
5	69.1	96.3	97.3
10	74.9	97.1	97.5
50	86.7	97.7	97.8

Table 13: Accuracy (%) comparison for different training strategies on an unseen task. RPG initialization demonstrates superior performance and faster convergence.

5 Related Works

In this section, we mainly introduce recurrent models and previous works of parameter generation.

Recurrent models. Recurrent neural networks (RNNs) were first proposed to process sequential data. To tackle the vanishing gradient problem in early RNNs, long short-term memory (LSTM) [28, 27] was introduced. Recently, transformer-based models [67] exhibits excellent potential in sequential data processing, due to their parallelized training and scalability, such as linear attentions [69, 8], RWKV [52], Mamba [22, 12], and xLSTM [4].

Parameter generation. The core idea of parameter generation is to learn the distribution of trained parameters. Stochastic neural networks [60, 6, 21] and Bayesian neural networks [47, 32, 20] model the priors or probability distributions over the parameters. However, these methods are limited by their generality to large-scale parameter generation. HyperNetworks [24], *i.e.* is proposed to generate various architectures' parameters for a larger network. Smash [7] extends the range of architectures via a memory read-writes scheme. With the development of diffusion models, many works [50, 9, 16, 68, 63, 39, 38, 31, 50] adopt diffusion models to generate neural network parameters. Hyper-Representations [55, 56, 57] use an autoencoder to capture the latent distribution of trained models. COND P-DIFF [31] and Tina [38] introduce text-controlled parameter generation method. Unfortunately, the above methods have a common drawback: *can not generate large-scale parameters, such as whole parameters of ResNet, ViT, ConvNeXt, or LoRA*. Therefore, our approach brings new inspiration to the field of parameter generation.

6 Discussion and Conclusion

Our approach demonstrates promising results in large-scale parameter generation across various vision, language and unseen tasks. However, we acknowledge that achieving true 'AI generating AI' remains a distant goal. Firstly, while our method shows potential in generating models for unseen tasks, it currently faces limitations in generating parameters for novel model architectures. Secondly, our approach is constrained by modeling parameter relationships within a single task, potentially limiting its practical applicability. More importantly, future work should focus on simultaneously modeling parameter relationships across diverse architectures and tasks. Such an approach could yield a more powerful and versatile parameter generator, potentially advancing us closer to the 'AI generating AI' era. We hope our approach will inspire and encourage future research in neural network parameter generation.

Acknowledgments and Disclosure of Funding

We sincerely thank Zhiyuan Liang, Zhuang Liu, Gongfan Fang, Xuanlei Zhao, Zangwei Zheng, Ziheng Qin, and Tianlong Chen 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] Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836*, 2022.

[2] Sudarshan Babu, Richard Liu, Avery Zhou, Michael Maire, Greg Shakhnarovich, and Rana Hanocka. Hyperfields: Towards zero-shot generation of nerfs from text. *arXiv preprint arXiv:2310.17075*, 2023.

[3] Sudarshan Babu, Pedro Savarese, and Michael Maire. Hypernetwork designs for improved classification and robust meta-learning.

[4] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

[5] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *AAAI*, volume 34, pages 7432–7439, 2020.

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

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

[8] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 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] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

[11] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[12] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. Ieee, 2009.

[14] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In *NeurIPS*, volume 34, pages 8780–8794, 2021.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

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

[17] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[18] Gongfan Fang, Xinyin Ma, and Xinchao Wang. Structural pruning for diffusion models. In *NeurIPS*, 2023.

[19] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pages 3259–3269. PMLR, 2020.

[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. In *NeurIPS*, 2011.

[22] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[23] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2024.

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

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[26] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, volume 33, pages 6840–6851, 2020.

[27] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

[28] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1):31, 1991.

[29] Sepp Hochreiter, Sepp Schmidhuber Jürgen, Hochreiter, and Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[30] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

[31] Xiaolong Jin, Kai Wang, Dongwen Tang, Wangbo Zhao, Yukun Zhou, Junshu Tang, and Yang You. Conditional lora parameter generation. *arXiv preprint arXiv:2408.01415*, 2024.

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

[33] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

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

[35] Daniel Kunin, Javier Sagastuy-Brena, Surya Ganguli, Daniel LK Yamins, and Hidenori Tanaka. Neural mechanics: Symmetry and broken conservation laws in deep learning dynamics. In *ICLR*, 2021.

[36] Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization. *arXiv preprint arXiv:2406.11838*, 2024.

[37] Yanghao Li, Hanzi Mao, Ross Girshick, and Kaiming He. Exploring plain vision transformer backbones for object detection. In *ECCV*, pages 280–296. Springer, 2022.

[38] Zexi Li, Lingzhi Gao, and Chao Wu. Text-to-model: Text-conditioned neural network diffusion for train-once-for-all personalization. *arXiv preprint arXiv:2405.14132*, 2024.

[39] Lequan Lin, Dai Shi, Andi Han, Zhiyong Wang, and Junbin Gao. Unleash graph neural networks from heavy tuning. *arXiv preprint arXiv:2405.12521*, 2024.

[40] 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.

[41] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

[42] Jonathan Lorraine, Kevin Xie, Xiaohui Zeng, Chen-Hsuan Lin, Towaki Takikawa, Nicholas Sharp, Tsung-Yi Lin, Ming-Yu Liu, Sanja Fidler, and James Lucas. Att3d: Amortized text-to-3d object synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17946–17956, 2023.

[43] 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 *NeurIPS*, volume 35, pages 5775–5787, 2022.

[44] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *CVPR*, pages 15762–15772, 2024.

[45] AI Meta. Introducing meta llama 3: The most capable openly available llm to date. *Meta AI*, 2024.

[46] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

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

[48] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *ICML*, pages 8162–8171. PMLR, 2021.

[49] Zizheng Pan, Bohan Zhuang, De-An Huang, Weili Nie, Zhiding Yu, Chaowei Xiao, Jianfei Cai, and Anima Anandkumar. T-stitch: Accelerating sampling in pre-trained diffusion models with trajectory stitching. *arXiv preprint arXiv:2402.14167*, 2024.

[50] William Peebles, Ilija Radosavovic, Tim Brooks, Alexei A Efros, and Jitendra Malik. Learning to learn with generative models of neural network checkpoints. *arXiv preprint arXiv:2209.12892*, 2022.

[51] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, pages 4195–4205, 2023.

[52] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

[53] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10684–10695, 2022.

[54] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

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

[56] Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-representations for pre-training and transfer learning. *arXiv preprint arXiv:2207.10951*, 2022.

[57] Konstantin Schürholt, Michael W Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. *arXiv preprint arXiv:2406.09997*, 2024.

[58] Konstantin Schürholt, Diyar Taskiran, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Model zoos: A dataset of diverse populations of neural network models. In *NeurIPS*, 2022.

[59] Junhyuk So, Jungwon Lee, Daehyun Ahn, Hyungjun Kim, and Eunhyeok Park. Temporal dynamic quantization for diffusion models. In *NeurIPS*, volume 36, 2024.

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

[61] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[62] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023.

[63] Bedionita Soro, Bruno Andreis, Hayeon Lee, Song Chong, Frank Hutter, and Sung Ju Hwang. Diffusion-based neural network weights generation. *arXiv preprint arXiv:2402.18153*, 2024.

[64] Alberto Tamajo. Imagenet1k-coarse-classes. https://github.com/albertotamajo/imagenet1k-coarse-classes, 2023. This repository organizes the ImageNet1k dataset into 10 coarse classes.

[65] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[66] Rejin Varghese and Sambath M. Yolov8: A novel object detection algorithm with enhanced performance and robustness. pages 1–6, 2024.

[67] A Vaswani. Attention is all you need. In *NeurIPS*, 2017.

[68] Kai Wang, Zhaopan Xu, Yukun Zhou, Zelin Zang, Trevor Darrell, Zhuang Liu, and Yang You. Neural network diffusion. *arXiv preprint arXiv:2402.13144*, 2024.

[69] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[70] Ross Wightman. GitHub repository: Pytorch image models. *GitHub repository*, 2019.

[71] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *ECCV*, pages 418–434, 2018.

[72] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[73] Xingyi Yang, Daquan Zhou, Jiashi Feng, and Xinchao Wang. Diffusion probabilistic model made slim. In *CVPR*, pages 22552–22562, 2023.

[74] David Yunis, Kumar Kshitij Patel, Pedro Henrique Pamplona Savarese, Gal Vardi, Jonathan Frankle, Matthew Walter, Karen Livescu, and Michael Maire. On convexity and linear mode connectivity in neural networks. In *OPT 2022: Optimization for Machine Learning (NeurIPS 2022 Workshop)*, 2022.

[75] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[76] Wangbo Zhao, Jiasheng Tang, Yizeng Han, Yibing Song, Kai Wang, Gao Huang, Fan Wang, and Yang You. Dynamic tuning towards parameter and inference efficiency for vit adaptation. *arXiv preprint arXiv:2403.11808*, 2024.

[77] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017.

We organize our appendix as follows.

Discussion with More Related Works

Experimental Setting and Other Details

Additional Experimental Results

A Discussion with More Related Works

A.1 Overview of comparisons

We compare RPG and four other methods from three aspects: scalability, performance, and generalization. Only our RPG excels in all three aspects simultaneously.

method	scalability	performance	generalization
S_{KDE30} [55]	✘	✘	✘
p-diff [68]	✘	✔	✘
SANE [57]	✔	✘	✘
D2NWG [63]	✔	✔	✘
HyperNetwork [24]	✘	✔	✔
HyperFields [2]	✘	✔	✔
ATT3D [42]	✘	✔	✔
HyperNetwork (MAML) [3]	✔	✘	✔
RPG (ours)	✔	✔	✔

Table 14: Comparison of RPG and existing methods on key aspects: scalability, performance, and generalization.

A.2 Discussion with Hyper-Representations

We mainly compare with three HyperRepresentation methods [55, 57, 56]. These methods use an autoencoder to learn the latent features of trained models, so they call the latent feature HyperRepresentation. This HyperRepresentation is then used for analyzing the model's performance or characteristics, or for sampling to generate new models or pre-trained parameters.

- [55] utilizes kernel density estimation (KDE) to sample model parameters on the learned HyperRepresentation space. They also emphasize the importance of layer-wise loss normalization in the learning process of HyperRepresentation. This work achieves parameter generation in small CNNs from Model Zoos [58] with 2864 parameters.

- [56] focuses on using HyperRepresentation to sample the pre-trained model parameters. They also evaluate the ability of transfer learning by using a trained parameter autoencoder to initialize on unseen dataset. This work can be regarded as a cheap parameter initialization method.
- [57] divides the neural network weights into subsets and utilizes a sequential autoencoder for neural embeddings (SANE) on there subsets with a sliding window. This work can generate the entire parameter of ResNet-18. However, its generation process does not directly derive parameters from noise. Instead, it relies on a half-trained model for Kernel Density Estimation (KDE) sampling.

We summarize the main differences as follows:

- *Hyper-representations as generative models* is hard to achieve comparable results as their original models that are used for training, but our approach obtains comparable results.
- *Hyper-representation for pre-training and transfer learning* focuses on parameter initialization while our approach targets to learn the distribution of high-performing neural network parameters.
- *SANE* is the latest method among these three HyperRepresentation methods. However, SANE uses a sliding window to model the relationship of a small part of trained parameters. Our approach uses a recurrent model among all parameters.
- Our approach can synthesize many popular vision and language parameters, such as ConvNeXt-L and LoRA parameters of LLaMA-7B, with a maximum parameter count of approximately 200M, which is much larger than previous works.

  Additionally, the parameters generated by our model can directly achieve almost peak performance without any finetuning. And the generation process is entirely synthesized from noise, eliminating the need for a few prompt examples that are trained for a few epochs, as required by the S_{KDE} [57] sampling.

  The capability of large-scale and high-accuracy generation makes our method more applicable in practical scenarios, significantly bridging the gap between theoretical parameter generation and practical application.

A.3 Discussion with HyperNetworks

The early HyperNetworks [24] can only generate a small number of parameters and cannot scale up, which limits their applications in many domains. However, in certain areas, researchers improve HyperNetworks to develop some interesting works.

- HyperFields [2] introduces a dynamic HyperNetwork that maps text directly to NeRF parameters, enabling zero-shot or few-shot 3D scene generation. By combining NeRF distillation with this architecture, it achieves faster training and broader generalization than traditional optimization-based methods.
- ATT3D [42] trains a single model across many text prompts, avoiding per-prompt optimization. This makes text-to-3D generation faster, more generalizable, and capable of smooth interpolations for new assets and animations.
- HyperNetwork with MAML [3] improves HyperNetworks' design by fixing gradient scaling, regularization, and momentum issues. The resulting HyperNetworks achieve higher accuracy and stronger robustness in meta-learning than standard models.

We summarize the main differences as follows:

- HyperFields and ATT3D aim to generate small, domain-specific networks tailored to particular functions. In contrast, our work explores the unified generation of large models—such as vision transformers and LoRA adapters for large language models—that has emerged in recent years.
- HyperNetwork with MAML focuses on meta-learning and tends to produce a better initialization rather than directly generating good parameters. In contrast, our work aims to directly generate high-quality parameters.

A.4 Details and limitations of G.pt

A primary limitation of G.pt [50] is the training data collection cost. By default, they collect 23 million checkpoints to train the parameter generator. Besides, they only evaluate the effectiveness of G.pt on small architectures, such as a low-dimensional MLP layer or a Convolutional layer with limited channels. The maximum number of generated parameters does not exceed 10,000.

A.5 Details and limitations of p-diff

P-diff [68] directly flattens all parameters into a 1D vector, disregarding the inter-layer parameter relationships. Furthermore, p-diff faces challenges in scaling up to large-scale parameter generation.

A.6 More related works

Diffusion models. Diffusion models [26, 48, 14] gain increasing popularity in recent years, due to their superiority in image generation. Ever since its advent, many works have been done focusing on improving the generation quality and efficiency of diffusion models. For generation quality, [53] propose to conduct diffusion in the latent space, enabling high-resolution image synthesis. [51] leverage the transformer [67] to explore scalability of diffusion models, proving the possibility of generating higher quality images by increasing model size. As for efficiency problem, efficient samplers [61, 43, 62], efficiency models [18, 59, 73], and global acceleration approaches [44, 49] are proposed to increase diffusion models' efficiency. These methods facilitate generating high quality images with less computational and/or memory cost. Although diffusion models for image generation have achieved great success, improving quality and efficiency in large-scale parameter generation remains to be explored.

A.7 Practical advantages of RPG in real-world applications

RPG offers several practical advantages in real-world scenarios. We list some of them below:

- *Efficient initialization:* RPG provides superior parameter initialization compared to random weights, significantly reducing training time and computational costs.
- *Few-shot learning:* For tasks with limited training data, RPG-generated parameters serve as informed priors that enable better performance than training from scratch.
- *Rapid model deployment:* RPG enables near-instantaneous model adaptation for new tasks without requiring extensive training, making it valuable for real-time applications where quick deployment is critical.

B Experimental Setting and Other Details

B.1 Training recipe

In this section, we provide detailed training recipes and supplementary information. The number of parameters generated by our approach ranges from approximately 3K to 200M. The significant disparity necessitates different training settings. Generally, as the number of parameters increases, the learning process becomes more challenging, requiring higher training costs, particularly for generating parameters beyond 50 million. Therefore, our training settings are divided into two categories: the default setting and the setting for parameters over 50 million, shown in Tab. 15.

Data parallelism: When the number of parameters is less than 50 million, we adopt a single GPU to run the training process. For larger number of parameters, we employ distributed data parallelism to facilitate the training.

Diffusion batch size: In our approach, the diffusion model is shared across all tokens. Typically, all tokens can be fed as a single batch into the diffusion model for training. However, in practice, we randomly select a subset of tokens from a long sequence for training, rather than feeding all parts at once. This approach significantly reduces memory usage without compromising performance. The 'diffusion batch size' in Tab. 15 refers to the number of tokens fed into the diffusion model during a single training iteration.

training setting	number parameters $<$ 50M	number parameters $>$ 50M
batch size	16	8
optimizer	AdamW	AdamW
learning rate	3e-5	1e-5
training steps	80,000	120,000
weight decay	1e-5	1e-5
mixed precision	bfloat16	bfloat16
diffusino batch size	1024	512

Table 15: Training recipe in detail.

B.2 Detailed structure of recurrent diffusion

In this section, we provide specific details about the proposed recurrent model and diffusion model in RPG. More detailed configurations can be found in Tab. 16.

module	setting	RPG-Tiny	RPG-Small	RPG-Base	RPG-Large
adequate number of parameters		$<$50K	50K$\sim$10M	5M$\sim$50M	$>$50M
recurrent (Mamba)	d_model of 1st layer	256	4096	8192	12288
	d_model of 2nd layer	256	4096	8192	16384
	d_state	32	128	128	128
	d_conv	4	4	4	4
	expand	2	2	2	2
	parameter counts	1.3M	256M	1018M	3076M
diffusion (1D CNN)	encoder channels	(1, 32, 64, 128)	(1, 32, 64, 128)	(1, 32, 64, 128)	(1, 64, 96)
	decoder channels	(128, 64, 32, 1)	(128, 64, 32, 1)	(128, 64, 32, 1)	(96, 64, 1)
	token size	256	4096	8192	16384
	kernel size	7	7	7	7
	default solver	DDPM	DDPM	DDPM	DDIM
	sampling steps	1000	1000	1000	60
	β-start & β-end	(0.0001, 0.02)	(0.0001, 0.02)	(0.0001, 0.02)	(0.0001, 0.02)
	betas schedule	linear	linear	linear	linear
	number time steps	1000	1000	1000	1000
	parameter counts	0.3M	17M	69M	273M

Table 16: Detailed information about four different sizes of recurrent diffusion. The *adequate number of parameters* implies that our model is usually adequate to generate parameters in that scale, which is empirical results instead of an exact rule. It also necessitates considering other factors such as parameter sensitivity.

Details of recurrent model. By default, the recurrent model consists of two Mamba layers [22]. As the increasing of parameters to generate, we need a larger recurrent model to capture the information in these parameters. The size of the recurrent model is mainly determined by the token size, which varies according to the number of parameters to be generated. Based on the token size, we categorize our model into four versions: Tiny, Small, Base, and Large.

Details of diffusion model. Following p-diff [68], our diffusion model adopts a 1D convolutional architecture. The parameters of the diffusion model are significantly fewer than those of the recurrent model. We feed the prototypes from the recurrent model as conditions into the diffusion model by directly adding them to the feature map.

The setting of our main experiments. In our main experiments, ViT-Base, ConvNeXt-Large, ADE20K, COCO, and DoRA rank 64 used the setting of RPG-Large. CNN (s) and CNN (m) used the setting of RPG-Tiny. And all other experiments used the setting of RPG-Base.

B.3 Description of datasets

In this section, we introduce the datasets used in the paper, including those for classification, semantic segmentation, object detection&instance segmentation, and commonsense reasoning.

Classification: ImageNet-1k [13] is a large-scale visual database for visual object recognition research. It contains over 1 million images across 1000 categories and is widely used for training and benchmarking deep learning models. **CIFAR-10** [33] dataset consists of 60,000 32×32 colorful images in 10 different classes. It is commonly used for training machine learning and computer vision algorithms, providing a standard benchmark for image classification task.

Semantic segmentation: ADE20K [77] is a dataset for semantic segmentation and scene parsing, containing over 20,000 images annotated with pixel-level labels for 150 object categories. It is used to train models to understand and segment various objects and scenes in an image, making it valuable for applications in autonomous driving, robotics, and image editing.

Instance segmentation & object detection: COCO [40] dataset is a large-scale object detection, segmentation, and captioning dataset. It contains over 330,000 images in various resolutions, with more than 200,000 labeled instances across 80 object categories. COCO is widely used for training and evaluating models in object detection, segmentation, and image captioning tasks.

Commonsense reasoning: BoolQ [10]: Yes/No questions based on natural passages. **PIQA** [5]: Questions about physical tasks and actions. **SIQA** [54]: Questions about social interactions. **HellaSwag** [75]: Choosing the correct ending for stories. **ARC** [11]: Multiple-choice science questions. **OBQA** [46]: Questions requires multi-step reasoning, commonsense knowledge, and rich text comprehension.

C Additional Experimental Results

C.1 Effectiveness of permutation state

The effect of permutation state. RPG incorporates a permutation state operation to address parameter symmetries, which become particularly pronounced when collecting checkpoints from multiple training runs. To evaluate this, we collect checkpoints from different numbers of training runs (1, 3, and 10) and compare the performance with and without permutation state. These results demonstrate that permutation state operation effectively addresses parameter symmetries and enables stable results even when incorporating checkpoints from multiple training runs.

collected runs	original	w/o permutation state	w/ permutation state
1	88.2	88.0	88.1
3	88.3	fail	88.1
10	88.5	fail	88.2

Table 17: Permutation state effectively mitigates parameter symmetries when collecting checkpoints from different training runs.

Analysis of LMC for evaluating model similarity. Linear mode connectivity [19, 40] (LMC) is a technique used to assess the similarity between deep neural network models by examining the performance of linear interpolations between their weight vectors. This metric would actually suggest diversity in the sense of models belonging to different basins [1] which is more meaningful.

In this experiment, we use ResNet-18 on CIFAR-10 with different training and sampling schemes, and test the performance of linear model weights interpolation. The results in Tab. 18 suggest that LMC can be broken in sampled models when multiple permutation states are applied. This further implies that RPG can learn the distribution of parameters, and its generating process is not merely a linear interpolation of the training data.

method \ accuracy (%)	A	$0.5A + 0.5B$	B
trained on checkpoints from the same basin	95.1	95.0	95.0
trained on checkpoints from 2 basins and sampled with 1 permutation state	95.0	94.8	94.9
trained on checkpoints from 2 basins and sampled with 2 permutation states	94.9	18.4	94.9

Table 18: LMC of two RPG-generated models under different training and sampling settings.

C.2 More results of Section 4

Results of generating models for unseen tasks. In Section 4, we show the potential of our approach in generating models for unseen tasks. In this part, we provide more results. First, we compare the performance of original and generated models using all unseen embeddings in Tab. 19. Results demonstrate that our approach consistently achieves good results in unseen tasks.

unseen binary embeddings	original	best accuracy	average accuracy	standard deviation
0 1 0 0 0 1 0 1 1 1	97.3	94.4	93.9	0.6
0 1 1 1 1 1 0 1 1 0	98.1	96.6	94.9	2.1
0 0 1 1 1 0 1 1 1 0	97.4	95.0	94.2	1.1
0 1 0 1 1 1 1 1 1 1	98.4	96.1	95.8	0.3
0 0 1 0 0 0 0 0 0 0	98.9	96.6	95.2	2.3
0 0 0 1 1 0 0 1 0 1	96.7	92.9	91.6	1.1
1 1 1 1 1 0 1 0 0 1	97.6	94.8	94.1	0.7
1 0 1 0 0 0 0 0 1 1	98.1	95.7	91.8	3.7
0 1 0 0 0 1 0 1 1 0	97.1	93.6	90.7	4.3
1 1 0 0 0 1 1 0 0 1	97.0	94.0	90.1	3.6
1 0 1 0 0 0 1 1 0 1	97.3	91.3	90.7	0.8
0 1 1 1 1 0 0 0 1 0	96.3	95.4	89.4	6.3
1 1 0 1 1 1 0 1 0 0	97.6	92.6	90.5	3.2
0 1 1 1 0 0 1 1 1 0	96.3	90.8	89.1	1.9
0 1 0 0 1 1 1 0 1 0	96.3	91.9	88.4	4.4
0 0 1 0 0 0 1 1 0 1	97.5	93.7	88.0	5.6
0 0 0 1 1 0 1 1 1 1	96.5	90.8	85.5	7.0
1 0 0 1 0 0 1 1 0 1	96.4	86.7	83.7	3.6
1 0 0 1 0 1 0 0 0 0	97.7	85.6	83.2	2.0
0 0 1 1 0 0 0 1 0 1	96.3	90.2	79.2	9.6

Table 19: Performance comparisons between original and generated models on unseen tasks. Specifically, we generated 10 models for each unseen task, with binary embeddings in the unseen set as condition. The results consistently show that our generated models perform comparably to the original models.

More experiments for ViT-Small on Coarse ImageNet-1K [64]. To verify the generalization potential of RPG on larger-scale datasets and models, we evaluated RPG on the ImageNet-1K dataset (coarse-grained 10 classes) [64] using the ViT-Small architecture. In Tab. 20, experimental results show that, despite not using any training data when generating parameters for new tasks, RPG achieves performance comparable to that of traditionally trained models. Moreover, RPG demonstrates strong generalization ability and adaptability across increasingly large datasets and models (from CIFAR10 to ImageNet, and from ViT-Tiny to ViT-Small), further validating its potential for practical applications.

unseen tasks (embeddings)	original	RPG
0 0 0 0 0 1 0 1 0 1	91.5	89.7
0 1 1 0 1 0 0 0 0 1	88.5	87.0
1 1 1 1 1 1 0 0 1 0	89.1	88.1
1 1 0 0 0 0 1 0 1 1	89.3	87.3
1 0 1 0 1 0 1 1 0 1	91.4	90.4
1 1 0 1 1 1 0 0 0 0	88.6	87.2
1 0 1 0 1 1 1 1 1 0	92.5	90.1
0 1 1 1 1 0 1 0 1 0	89.5	87.5
1 0 1 1 0 1 0 0 0 1	90.1	87.9
1 1 0 1 1 0 0 0 0 0	90.2	89.2

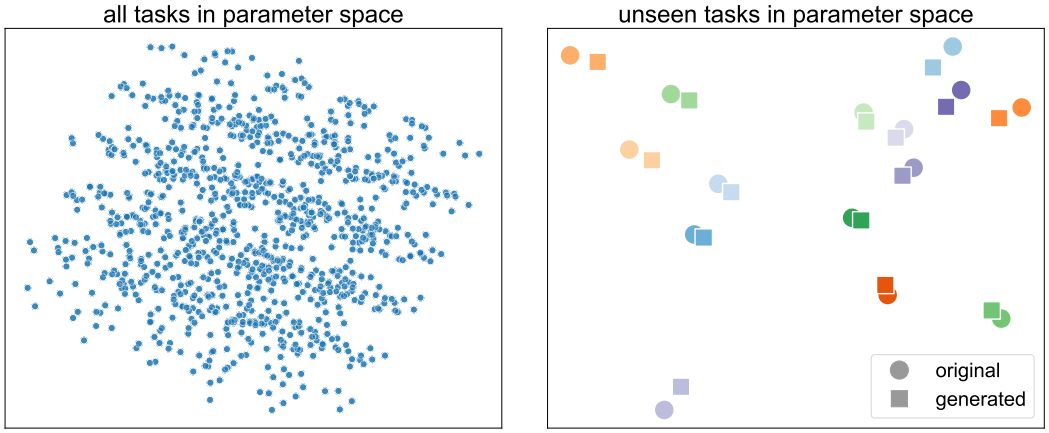
Table 20: Result comparisons between original and generated models for generated ViT-Small on Coarse ImageNet-1K [64].

More experiments for YOLOv8n [66] on PASCAL VOC [17]. For more complex scenarios, we employ YOLOv8n to demonstrate RPG's ability to generate models for unseen tasks. We use the PASCAL VOC dataset, which contains 20 classes, and design 20-bit positional embeddings to specify the target detection requirements ('1' indicates that a class must be detected, and '0' otherwise). We then train 500 YOLOv8n models on 500 randomly sampled tasks and collect their checkpoints. Performance is evaluated using mAP50-95, comparing models fine-tuned from pretrained weights against those initialized with generated parameters. The results in Tab. 21 show that RPG provides strong initializations for new tasks, enabling faster adaptation and reducing the need for extensive training. This confirms our method's scalability and effectiveness on more complex tasks.

unseen tasks (embeddings)	epoch-0	epoch-1	epoch-2	epoch-5
0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1	0.00 / 0.29	0.01 / 0.30	0.06 / 0.33	0.18 / 0.39
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0	0.01 / 0.25	0.02 / 0.30	0.05 / 0.32	0.21 / 0.39
0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0	0.01 / 0.28	0.09 / 0.33	0.12 / 0.35	0.32 / 0.38
0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1	0.00 / 0.24	0.04 / 0.25	0.04 / 0.29	0.13 / 0.35
0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0	0.00 / 0.19	0.03 / 0.22	0.11 / 0.26	0.21 / 0.33

Table 21: Result comparisons between original and generated models for generated YOLOv8n [66] on PASCAL VOC [17].

PCA visualization of classification head parameters. We also provide a visualization of the parameters of the classification head (a two-layer fully connected structure with total 38,976 parameters) for 1022 tasks as described in Section 4 using Principal Component Analysis (PCA), which presents the structure of the parameter space in Fig. 6a. Our generated model achieves an average accuracy of 91.2% across all binary classification tasks, which indicates that our method has effectively learned this structure. Furthermore, we evaluate the parameters corresponding to unseen tasks and compared their positions in Fig. 6b between the original and generated parameters. It is noteworthy that, even though the original parameters of these tasks are not included in the training data, the generated parameters consistently appeared in close proximity to the original ones. This observation further highlights the capability of our method to model the structure of the parameter space, even for tasks not previously encountered.



(a) Visualization of the classification head of all 1022 tasks. This reveals that there is an inherent structure among the high-dimensional parameters, which can be captured by deep learning models.

(b) Visualization of the classification head in some unseen tasks. Parameters associated with the same task are indicated by a consistent color. Our method can capture the structure of the parameter space.

Figure 6: Principal Component Analysis (PCA) visualization of the classification head. The figures demonstrate the presence of an inherent structure in the parameter space and highlight our method's effectiveness in capturing this structure for unseen tasks.

Conditional generation guided by LLM. To demonstrate the application scenarios of our model, we utilize large language model to generate binary embeddings to guide RPG in generating corresponding classification models. For the first example in Fig 7, we give the LLM a prompt: 'Give me a model to select all living things.' With the binary embedding provided by the LLM, our RPG then generates a ViT-Tiny classifier. After that, We use images in CIFAR-10 to evaluate the model's accuracy. The model should classify 'bird', 'cat', 'deer', 'dog', 'frog', and 'horse' to the positive class, which we used as the ground truth. The result is 97.1%. Some other examples are in Tab 22. This experiment demonstrates our method's capability to generate neural network parameters based on natural language guidance, highlighting the potential applications of our method.

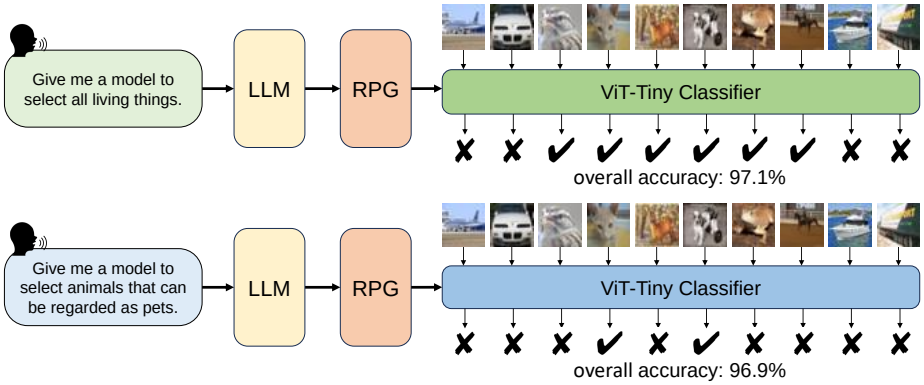


Figure 7: Illustration of RPG-generated models guided by binary embeddings from a large language model (Qwen2.5-3B [72]), demonstrating parameter generation conditioned by natural language.

prompt	expected embedding	acc. (%)
Give me a model to select all living things.	0,0,1,1,1,1,1,1,0,0	98.7
Find all vehicles that operate on roads.	0,1,0,0,0,0,0,0,0,1	95.9
Classify all mammals.	0,0,0,1,1,1,0,1,0,0	97.6
Select all man-made objects.	1,1,0,0,0,0,0,0,1,1	97.7
Find all things that are both man-made and can operate on water.	0,0,0,0,0,0,0,0,1,0	98.4
Select all animals that can be regarded as pets.	0,0,0,1,0,1,0,0,0,0	96.8
Select all things that can fly.	1,0,1,0,0,0,0,0,0,0	87.7
Find all animals with fur.	0,0,1,1,1,1,0,1,0,0	70.4
Select all pets commonly found in households.	0,0,1,1,0,1,0,0,0,0	83.3
Identify all cold-blooded animals.	0,0,0,0,0,0,1,0,0,0	98.9

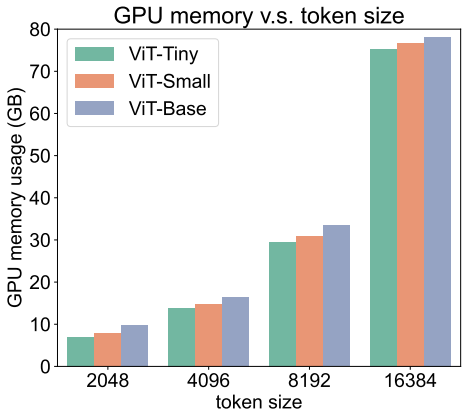
Table 22: The table demonstrates examples of generating from abstract prompts using LLM and RPG. We are able to achieve high accuracy on abstract prompts. The *expected embedding* is used for evaluating the accuracy.

C.3 Training memory cost analysis

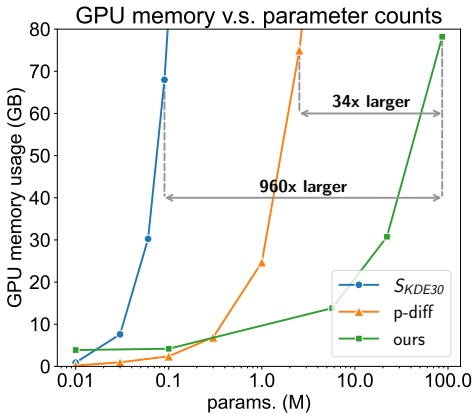
In this section, we analyze the GPU memory utilization during training. GPU memory consumption is usually highly correlated with two factors: i) the size of the generative model and ii) the size of generated parameters. We analyzed the impact of these two factors on the GPU memory utilization during the training of our approach.

GPU Memory vs. Token Size. In Tab. 5, we present the relationship between token size and RPG's learning capability. Despite the fact that larger token sizes (i.e., larger models) carry a greater volume of information, they also lead to substantial GPU memory costs, as illustrated in Fig. 8a. This implies that, when the performance of the generated models is comparable, we prefer to use models with smaller token sizes.

GPU memory v.s. parameter counts. We conduct experiments to show the relationship between GPU memory and generated parameter counts in Fig. 8b. In previous methods, the relationship between GPU memory consumption and the number of parameters in the generated model was quadratic [55] or directly proportional [68]. This limits their practicality and application range. In contrast, our approach demonstrates remarkable efficiency: with equivalent GPU memory usage, it can generate models with 34 to 960 times more parameters compared to previous methods.



(a) Visualization of GPU memory v.s. token size. GPU memory usage increases proportionally to the token size. Thus, the token size cannot get larger infinitely; we need to choose a proper token size.



(b) Visualization of GPU memory v.s. parameter counts. Our method can generate much more parameters than existing approaches e.g. S_{KDE30} [55] using a single NVIDIA H100 80G GPU.

Figure 8: Training memory cost analysis. *Left:* GPU memory v.s. token size. *Rihgt:* GPU memory v.s. parameter counts.

C.4 Inference memory cost & sampling time

In this section, we present more information about the sampling, including memory usage, inference time, and the balance between sequential and parallel inference. In Tab. 9, we show the sampling time and memory usage for ViT-Base and ConvNeXt-L. Here, we present the sampling time and memory usage for other models. In Tab. 23, we adopt DDPM as the solver and conduct 1000-step sampling. Since the diffusion model in RPG is shared among all the parameter tokens, we can adopt different inference modes to find a balance between memory usage and inference speed:

- **fully parallel:** All tokens are fed into the diffusion model simultaneously. This approach results in a high memory usage but achieves a high generation speed.
- **sequential:** Tokens are fed into the diffusion model one by one. This approach significantly reduces memory usage, as the model only occupies memory for inferring a single token at a time. This enable us to generate parameters of models listed on a GPU with less than 8GB of memory .
- **partially parallel (default):** In partial parallel mode, we set 256 tokens as a batch for the diffusion model inference. This approach significantly boosts speed with a slight increase in GPU memory usage, reaching an optimal trade-off between memory and speed. We adopt this as the default setting.

Based on the results in Tab. 23, our approach can be flexibly applied to many other GPUs as it can achieve a good trade-off between memory and time.

metrics	inference mode	ResNet-18	ResNet-50	ViT-Tiny	ViT-Small	ConvNeXt-A
time (minute)	sequential	18.6	38.0	9.8	33.8	6.8
	partially parallel	1.8	3.3	1.1	2.9	0.9
	fully parallel	1.7	3.3	1.1	2.9	0.9
memory cost (GB)	sequential	6.3	6.4	6.2	6.4	6.2
	partially parallel	10.3	10.5	10.3	10.5	10.3
	fully parallel	30.8	50.5	19.4	45.9	15.2

Table 23: We show the inference time and memory cost under different inference modes. All information in this table is collected from a single NVIDIA H100 80G GPU. We report the time and memory required to generate a single model.

C.5 Parameter sensitivity vs. performance

According to conventional understanding, larger parameter quantities are generally more challenging to learn. However, our experiments reveal that this rule is not absolute and demonstrates instability in learning some small model parameters.

This motivates us to investigate the relationship between parameter sensitivity and generation quality. Specifically, we add Gaussian noise with weights of 0.01, 0.10, and 1.00 to the original parameters to measure model sensitivity, as shown in Tab. 24. We observe that as noise weight increases, performance decreases for all models, with smaller models being more affected than larger ones. This indicates that smaller models are relatively more sensitive. Additionally, we notice that the performance gap between the original and generated models widens as sensitivity of the model increases. This demonstrates a strong correlation between a model's sensitivity and the difficulty of generating its parameters.

model	params. (M)	sensitivity	accuracy decline			
			ours	noise (0.01)	noise (0.10)	noise (1.00)
ConvNeXt-A	3.7	+++	0.85	62.83	0.60	0.03
ResNet-18	11.7	++	0.39	53.56	0.46	0.00
ViT-Base	86.6	+	0.09	5.39	0.02	0.00

Table 24: The accuracy decline reflects the accuracy gap between the original model and the generated model or the model after adding noise. We add Gaussian noise with weights of 0.01, 0.10, and 1.00 to the parameters to measure model sensitivity. Results demonstrate that smaller models are relatively more sensitive than larger ones. The more plus signs (+) , the higher the sensitivity.

C.6 Details of trained checkpoint collection

This section mainly supplements the collection of checkpoints in Section 3. First, we obtain the pre-trained models, which either come from model libraries (such as timm [70]) or are trained by ourselves. Then, we fine-tune the full model for one epoch, and save 50 checkpoints during this epoch uniformly. In Tab 17, the term *collected runs* refers to the number of times the entire process, from pre-training to fine-tuning and saving, is repeated. This is done without fixing the seed, resulting in checkpoints from entirely different permutations.

In addition, we also compared the impact of the number of collected checkpoints on the similarity between models. Based on the results in Tab. 25, the analysis reveals that using too few checkpoints lead to low diversity. However, as the number of checkpoints increases, the similarity initially decreases and then stabilizes around 0.84 when the count exceeds 50.

number of checkpoints	1	10	50	200	400
similarity	0.93	0.89	0.84	0.83	0.84

Table 25: Comparison between the number of checkpoints and similarity.

C.7 Impact of the diffusion process

We conduct experiments to analyze the effect of the diffusion process. We compare the performance of our method with and without the diffusion process. The results are obtained by training with ViT-Tiny. The findings are as follows: (1) using the diffusion process allows for generating an infinite number of models, whereas the version without the diffusion process can produce only one model; (2) incorporating the diffusion process increases the diversity between the original and generated models (0.84 vs. 0.91); (3) the model generated without the diffusion process tends to memorize the ensemble of original models more strongly, as indicated by a high similarity score of 0.93.

diffusion process	number of generated models	similarity	similarity with ensembled original models	accuracy (%)
✓	infinite	0.84	0.85	75.3
✗	only one model	0.91	0.93	75.3

Table 26: Comparison of with and without diffusion processes.

C.8 Computational cost for training

Training RPG requires computational resources. However, all experiments can be completed within 144 GPU hours (8 GPUs $\times$ 18 hours), as shown in Tab. 27.

model	number of parameters	training cost (GPU hours)
ViT-Tiny	6M	4
DoRA (rank4)	8M	4
ViT-Small	22M	10
ViT-Base	86M	54
DoRA (rank64)	113M	73
ADE20K	177M	132
ConvNeXt-L	198M	144
ViT-Tiny (Section-4)	6M	40

Table 27: Training cost of RPG for various model architectures.

C.9 Why not auto-regression?

It is worth noting that our approach does not employ an auto-regressive method, *i.e.*, we do not feed the output back as input again. Our method takes position embedding and permutation state as inputs and synthesizes neural network parameters as outputs, forming a standard recurrent neural network. We have attempted to train our model using an auto-regressive approach such as a decoder-only transformer architecture, whose results are shown in Tab 6. To ensure a fair comparison, we also applied a causal mask to the transformer encoder-only structure, ensuring that information can only be passed in one direction. Due to the accumulation of errors during the auto-regressive process in inference, the parameters generated at the end of sequence become nearly indistinguishable from noise, leading to poor performance. In contrast, in RPG, noise is only introduced in the diffusion model and does not accumulate in the recurrent process, ensuring stable parameter generation.

C.10 Generating from noise vs. mapping from embedding

When generating neural network parameters, methods that synthesize weights from noise—such as RPG—offer significant advantages in memory efficiency and scalability over approaches that map learnable embeddings directly to full parameter tensors.

- Lower memory footprint: Embedding-based decoders must produce the entire parameter tensor in one go, causing memory usage to scale linearly with model size. In contrast, RPG generates weights sequentially, drastically reducing peak memory consumption.
- Superior scalability: RPG's autoregressive generation process naturally accommodates large models without requiring storage or computation of massive, dense parameter tensors at once.
- Favorable compute-memory trade-off: While RPG incurs a modest increase in compute due to sequential generation, it achieves orders-of-magnitude memory savings—making it far more practical for deployment in resource-constrained or large-scale settings.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: All the claims are supported by the experimental results.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss it in the conclusion and discussion section.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: This paper does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide detailed recipes in the Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We release all the code to the public.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We include all necessary details.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report the best, min, and average results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We introduce this in our main paper and appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We strictly follow NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss it in the conclusion and discussion section, and appendix.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to

generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.

- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All involved papers are cited properly.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: The paper does not release new assets.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: This paper's development does not involve LLMs.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.