PARAGEN-BENCH: A BENCHMARK FOR NEURAL NETWORK PARAMETER GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural network parameter generation is a paradigm of producing the trained weights of a target network through a learned generator. It has emerged as a new direction for acquiring model parameters beyond hand-designed algorithms and backpropagation. However, the field still lacks a benchmark that enables comprehensive and standardized comparison across existing methods and provides a systematic understanding of the current state of development. To close this gap, we introduce PARAGEN-BENCH, the first benchmark for neural network parameter generation, covering image classification, text classification, image segmentation, reasoning, coding, multimodal understanding, and reinforcement learning. It comprises 44,000 trained checkpoints across 41 architectures and 29 datasets, each paired with 50 paraphrased prompts and structured metadata capturing training trajectories and hyperparameters. We conduct a comprehensive evaluation of ten representative methods spanning three paradigms: diffusion-based, hypernetwork-based, and large language model (LLM)-conditioned approaches. Results show that larger and more structurally complex target models exhibit greater performance gaps, parameter errors cause particularly severe downstream degradation in image segmentation and reinforcement learning, and gaps remain between generated and trained LoRA adapters across reasoning, coding, and multimodal tasks. Finally, we propose simple yet effective LLM-instructed parameter generation (LPG), a two-stage baseline that combines structure-aware weight tokenization with prompt-conditioned latent generation [1].

1 INTRODUCTION

Neural network parameter generation has emerged as a new paradigm that treats model parameters as a generative modality (Wang et al., 2026; Han et al., 2026), shifting from gradient-based training toward direct generation. Recent parameter generation methods have demonstrated the potential of this paradigm across several dimensions. Parameter generation amortizes the cost of task-specific model training. Conventional approaches require training a separate neural network for each task through iterative gradient-based optimization, whereas a generator is trained once on a population of models and produces parameters for a new task without iterative optimization (Charakorn et al., 2025; Liang et al., 2025). Beyond efficiency, recent studies suggest that generated parameters have the potential to achieve performance comparable to, or even better than, the trained models used to learn the parameter distribution (Wang et al., 2024; 2025b). Moreover, parameter generation can specialize a large language model (LLM) for a target task directly from a natural language description (Charakorn et al., 2025; Huang et al., 2026).

Despite this rapid progress, the evaluation of parameter generation remains fragmented and insufficient. Existing methods are typically studied under different target architectures, task domains, conditioning types, and evaluation protocols, making direct comparison difficult. Existing model-zoo and checkpoint datasets have provided important resources for studying neural network parameters (Unterthiner et al., 2020; Schürholt et al., 2022; Peebles et al., 2022; Falk et al., 2025; Gabrielsson et al., 2025), but their coverage and scope remain limited. Most existing resources focus on a single

[1] Anonymous code is available at `https://anonymous.4open.science/r/l2w-630C`, and the benchmark is anonymously hosted at `https://huggingface.co/datasets/AnonymousAuthorEE33/ParaGen-Bench`.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

task domain or a small number of model architectures, providing limited coverage across tasks, model structures, and generation targets. These limitations make it challenging to assess the overall progress of parameter generation, identify where current methods fail, and compare different methods on equal footing. Therefore, *a substantially more diverse benchmark is needed to make the capabilities of current parameter generation methods systematically measurable and comparable*.

The First Neural Network Generation Benchmark. To fill this critical gap, we introduce the *first*, *comprehensive*, and *parameter-centric* neural network parameter generation benchmark PARAGEN-BENCH. PARAGEN-BENCH (Table 1) comprises 44,000 trained checkpoints spanning 41 architectures and 29 datasets across seven task domains: image classification (IC), text classification (TC), image segmentation (IS), reinforcement learning (RL), and LoRA adaptation (LA) including mathematical reasoning, code generation, and multimodal understanding. Each checkpoint is paired with **50** paraphrased natural-language prompts and structured metadata capturing training trajectories and hyperparameters. This broad coverage enables evaluation across different task domains, model architectures, and generation targets, exposing where current parameter generation methods succeed and where they fail. It further enables direct comparison of different methods under the same target models and downstream evaluation, avoiding confounding factors from differences in target performance, model architecture, and evaluation settings across prior studies.

Table 1: Comparison of PARAGEN-BENCH and existing related model zoos. D1, D2, and D3 denote the architectures, training trajectories, and training metadata.

Benchmark	Year	# Ckpts	Task					Diversity		
			IC	IS	TC	RL	LA	# D1	D2	D3
Small CNN Zoo Unterthiner et al. (2020)	2021	120K	✓	✗	✗	✗	✗	1	✗	✓
G.pt Datasets Peebles et al. (2022)	2022	23M	✓	✗	✗	✓	✗	3	✓	✓
Model Zoos Schürholt et al. (2022)	2022	2.6M	✓	✗	✗	✗	✗	3	✓	✓
Lots-of-LoRAs Gabrielsson et al. (2025)	2025	1.3K	✗	✗	✗	✗	✓	1	✗	✗
ViT Model Zoo Falk et al. (2025)	2025	250	✓	✗	✗	✗	✗	1	✓	✓
PARAGEN-BENCH (Ours)	2026	44K	✓	✓	✓	✓	✓	41	✓	✓

Limitation of Current Methods. Existing parameter generation methods are often designed for specific conditioning inputs, parameter forms, or target architectures, which limits their applicability across different generation settings. For example, T2L generates LoRA adapters from natural-language task descriptions, whereas D2NWG relies on data samples to generate full model parameters. In addition, existing methods are typically evaluated under different datasets, target models, trained-model performance, and evaluation protocols, leaving their relative strengths unclear under a unified setting. Evaluation on PARAGEN-BENCH further reveals clear limitations across different task settings. In classification, current methods struggle with large target models and architectures with more diverse layer types, with several methods reaching near-random performance. In image segmentation and reinforcement learning, current methods struggle to reliably generate parameters for functionally sensitive models, where small parameter errors can cause large downstream performance drops. In LoRA generation, current methods do not consistently recover trained-adapter performance across reasoning, coding, and multimodal tasks.

A Simple yet Effective Baseline. Based on this benchmark, we further propose a simple yet effective baseline to serve as a standard method, LLM-instructed parameter generation (`LPG`). In the first stage, `LPG` trains a variational autoencoder over structure-aware tokenized weight sequences. In the second stage, a lightweight MLP projector is trained to map prompt embeddings extracted from a frozen LLM into the VAE latent space. While maintaining its simplicity and versatility, `LPG` achieves competitive performance across diverse task domains on PARAGEN-BENCH. To sum up, the contributions of this work are fourfold:

- We introduce PARAGEN-BENCH, the first comprehensive benchmark for neural network parameter generation, covering **seven** task domains, **41** architectures, **29** datasets, and **44,000** trained checkpoints paired with natural-language prompts and structured metadata shown in Figure 1. Furthermore, we unify previously fragmented parameter generation evaluations across tasks, model architectures, and generation targets under a common benchmark setting.
- We conduct a systematic empirical study of ten representative parameter generation methods spanning diffusion-based, hypernetwork-based, and LLM-based generators, showing clear limitations on larger models and architectures with diverse layer types, strong sensitivity to parameter errors in image segmentation and reinforcement learning, and consistent performance gaps in LoRA-based LLM adaptation across mathematical reasoning, code generation, and multimodal understanding.
- We propose `LPG`, a simple yet effective two-stage baseline serving as a reference point on PARAGEN-BENCH. `LPG` combines structure-aware weight tokenization with prompt-conditioned latent generation and `LPG` achieves performance comparable to existing generation models.

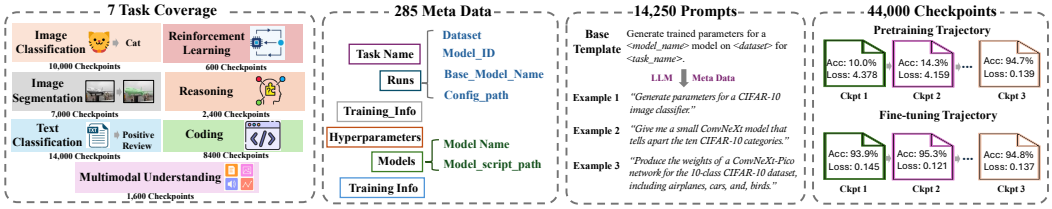


Figure 1: Overview of our proposed benchmark PARAGEN-BENCH.

2 RELATED WORK

Related Model Zoos. Existing model zoo datasets focus on narrow slices of the weight space. Small CNN Zoo (Unterthiner et al., 2020) provides 120K checkpoints but is limited to a single CNN architecture on image classification. Model Zoos (Schürholt et al., 2022) scales to 2.6M checkpoints with training trajectories across three architectures, but still covers only image classification. ViT Model Zoo (Falk et al., 2025) offers 250 ViT checkpoints with rich metadata but again within a single architecture family. G.pt Checkpoints (Peebles et al., 2022) is the largest at 23M checkpoints and the only dataset to include RL, but its architectures are restricted to simple CNNs and MLPs. Lots-of-LoRAs (Gabrielsson et al., 2025) provides 1.3K LoRA adapters for language tasks, but all share a single Mistral-7B base model. None of these datasets spans more than 2 task domains or offers meaningful architecture diversity, making it impossible to evaluate generation methods under domain shifts or architecture scaling. PARAGEN-BENCH closes this gap with 44K checkpoints across 41 architectures and seven task domains, as summarized in Table 1.

Neural Network Parameter Generation. Recent neural network parameter generation methods adopt several generation paradigms: diffusion-based approaches generate weights through iterative denoising (Wang et al., 2024; Jin et al., 2024; Soro et al., 2025b; Khan et al., 2025; Wang et al., 2025b; Huang et al., 2026), autoregressive methods synthesize weight tokens sequentially (Soro et al., 2025a), hypernetworks produce weights in a single forward pass (Ha et al., 2017; Ratzlaff & Fuxin, 2019; Zhmoginov et al., 2022; Phang et al., 2023; Schürholt et al., 2024; Charakorn et al., 2025; Liang et al., 2025), and LLM-conditioned methods leverage language models as the generation backbone (Wang et al., 2025a; Xiao et al., 2025). Alongside these methodological advances, parameter generation has evolved from unconditional generation toward personalized parameter generation (Li et al., 2025) guided by natural-language requests. Existing methods have shown that parameter generation is feasible in selected settings, but their effectiveness beyond these settings remains unclear. Moreover, existing approaches are often restricted to particular datasets or tasks. Additionally, differences in training data, target architecture, target model performance, and evaluation protocol further hinder systematic comparison across methods.

3 PARAGEN-BENCH

In this section, we introduce the problem definition (subsection 3.1), construction of PARAGEN-BENCH (subsection 3.2), and the proposed LPG baseline (subsection 3.3).

3.1 PROBLEM DEFINITION

We define neural network parameter generation as the task of producing parameters for a target model. Each entry in PARAGEN-BENCH is formalized as a tuple of six components: $\mathcal{E} = (\mathcal{D}, \mathcal{A}, \mathcal{C}, \mathcal{I}, \theta, \mathcal{P})$, where $\mathcal{D}$ identifies the training data source, including the dataset name, task domain, and data split. $\mathcal{A}$ specifies the target architecture, including its layer configuration, parameter count, and source. $\mathcal{C}$ records the training configuration such as learning rate, batch size, optimizer, and random seed. $\mathcal{I}$ stores training metadata including losses, validation metrics, and parameter statistics. θ denotes the trained parameters of $\mathcal{A}$. $\mathcal{P} = \{p_j\}_{j=1}^{M}$ denotes a set of M natural-language instructions describing the task and target architectures. Parameter generator $\mathcal{G}$ uses selected components of $\mathcal{E}$ to produce generated parameters $\hat{\theta}$ for the target architecture $\mathcal{A}$.

3.2 PARAGEN-BENCH CONSTRUCTION

In this subsection, we present how to construct our proposed PARAGEN-BENCH including data structure, tasks and datasets, pipeline, prompt annotation, and quality assurance. More details can be found in Appendix A.3.

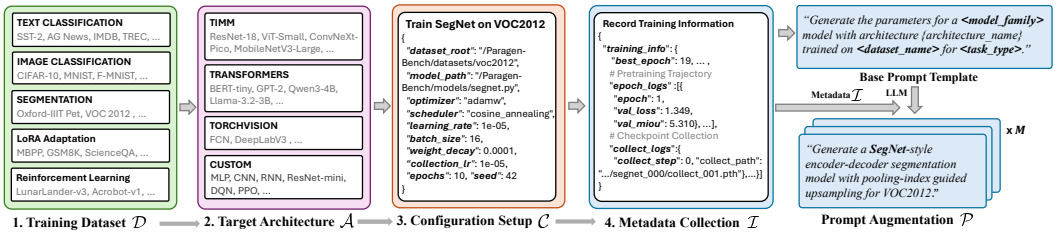


Figure 2: Overview of the construction pipeline and examples.

Construction Principles. PARAGEN-BENCH is designed around three principles. First, it spans diverse task domains, model architectures, and generation targets, enabling systematic diagnosis of where current parameter generation methods succeed and fail. Second, each checkpoint pool is constructed from converged models with verified downstream performance, ensuring that the collected checkpoints form a reliable target parameter distribution. Third, each checkpoint is accompanied by structured metadata that records its training configuration, optimization trajectory, and downstream performance, enabling traceable analysis and retrieval of checkpoints with specific properties.

Tasks, Dataset, and Architecture Selection. We select seven task domains comprising five image classification datasets, five text classification datasets, five image segmentation datasets, four mathematical reasoning datasets, three code generation datasets, four multimodal understanding datasets, and three reinforcement learning environments (Table 5). Within each domain, we pair these datasets with multiple target architectures spanning different model scales and architecture families. These selections result in 29 datasets, 41 architectures, and 285 architecture–dataset covering both full-parameter and LoRA generation.

Construction Pipeline. Figure 2 shows the PARAGEN-BENCH construction pipeline. For each architecture–dataset pair, we first initialize the corresponding metadata entry with the dataset, target architecture, and training configuration. We then train the target model to convergence and continue optimization from the converged checkpoint using a new optimizer and a learning rate selected for that pair. During collection, we record the training hyperparameters, loss and validation trajectories, downstream performance, and parameter statistics for each checkpoint. We collect 200 consecutive checkpoints for image classification, text classification, and image segmentation, and 100 checkpoints for mathematical reasoning, code generation, multimodal understanding, and reinforcement learning. Each architecture–dataset pair is then annotated with a prompt set $\mathcal{P}$ as described below.

Prompt Annotation. We construct a set of M natural-language instructions for each architecture-dataset pair. Starting from a base template populated with model and dataset metadata (see the **Prompt** example in Figure 1), we query Qwen3-VL-32B-Instruct (Bai et al., 2025) to generate diverse variations along three controlled axes: *architecture specificity*, ranging from abstract model references to detailed descriptions; *task granularity*, ranging from generic descriptions to specific expected behaviors; and *user intent*, ranging from direct requests to descriptive framing. The resulting prompt set is stored as $\mathcal{P} = \{p_j\}_{j=1}^{M}$.

Quality Assurance. We perform quality control on both checkpoint pools and prompt annotations. Checkpoint quality is verified in two stages. First, each target model is trained to convergence, and the checkpoint with the best validation performance is selected as the trained model. Second, we require the standard deviation of the downstream performance within each collected checkpoint pool to remain below two percentage points of the task metric. For prompts, we remove near-duplicate instructions using sentence-embedding cosine similarity and randomly sample 10 prompts from each architecture–dataset pair for human verification of consistency with the corresponding task, dataset, and target architecture.

Data Release. For each architecture–dataset pair, the 50 prompts are split into 30 training, 10 validation, and 10 test prompts for prompt-conditioned methods. We release the corresponding checkpoint pool, metadata, and prompt annotations through an anonymized Hugging Face dataset.[2]

Data Analysis. Figure 3 summarizes the distribution of 44,000 checkpoints and 14,250 prompt

[2] https://huggingface.co/datasets/AnonymousAuthorEE33/ParaGen-Bench

annotations across seven task domains. The benchmark contains 10,000 (22.7%) checkpoints for image classification, 14,000 (31.8%) for text classification, 7,000 (15.9%) for image segmentation, 2,400 (5.5%) for mathematical reasoning, 8,400 (19.1%) for code generation, 1,600 (3.6%) for multimodal understanding, and 600 (1.4%) for reinforcement learning. The corresponding prompt annotations are 2,500 (17.5%), 3,500 (24.6%), 1,750 (12.3%), 1,200 (8.4%), 4,200 (29.5%), 800 (5.6%), and 300 (2.1%), respectively. Each architecture–dataset pair is associated with 50 natural-language prompts, resulting in 14,250 prompt annotations across 285 architecture–dataset pairs.

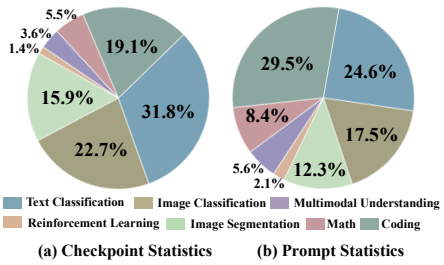


Figure 3: Statistics of checkpoints and prompts across seven task domains.

3.3 Proposed Baseline: LPG

In this subsection, we introduce our proposed simple yet effective LPG consisting of three stages: preprocessing, training, and inference.

Preprocessing. We prepare each training pair by sampling a prompt $p_j \in \mathcal{P}$ as input and tokenizing the target parameters θ as output. Following RPG (Wang et al., 2025b), we first apply per-key z-score normalization to θ, then flatten each tensor independently and split it into fixed-size chunks, yielding weight tokens $\mathbf{W}$. Each token in $\mathbf{W}$ is augmented with a hierarchical positional embedding that captures its layer, module, tensor, and chunk position.

Training Stage 1. The first stage trains a VAE (Kingma & Welling, 2013) over tokenized checkpoint weights. Given weight tokens $\mathbf{W}$, the encoder produces $\mathrm{Enc}(\mathbf{z} \mid \mathbf{W}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, and the decoder reconstructs $\hat{\mathbf{W}} = \mathrm{Dec}(\mathbf{z})$. The optimization loss is the sum of a reconstruction loss and a KL regularizer:

$$\mathcal{L}_{\mathrm{VAE}} = \|\mathbf{W} - \hat{\mathbf{W}}\|_2^2 + \beta\, \mathrm{KL}\left(\mathrm{Enc}(\mathbf{z} \mid \mathbf{W}) \,\|\, \mathcal{N}(0, I)\right), \tag{1}$$

where β is the hyperparameter to control the KL weight. This optimization process learns a compact weight latent space while keeping the prior easy to sample from.

Training Stage 2. The second stage maps the prompt to the VAE latent space. First, given a prompt $p_j \in \mathcal{P}$ and optional data samples when available, we encode it with an LLM to obtain a conditioning embedding $\mathbf{h}_j$. With the LLM and the VAE decoder frozen, we only train a lightweight MLP projector to predict a latent Gaussian $(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}}) = \mathrm{Proj}(\mathbf{h}_j)$ by minimizing:

$$\mathcal{L}_{\mathrm{proj}} = \|\mathbf{W} - \mathrm{Dec}(\hat{\boldsymbol{\mu}})\|_2^2 + \gamma\, \mathrm{KL}\big(\mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}}^2) \,\|\, \mathrm{Enc}(\mathbf{z} \mid \mathbf{W})\big), \tag{2}$$

where $\mathrm{Enc}(\mathbf{z} \mid \mathbf{W})$ is the posterior from the frozen VAE encoder trained in Stage 1 and γ is the hyperparameter. The first term $\|\mathbf{W} - \mathrm{Dec}(\hat{\boldsymbol{\mu}})\|_2^2$ encourages the decoded weights to match the target checkpoint, whereas the second term aligns the projected distribution with the VAE posterior of the same checkpoint.

Inference. At inference time, we encode an unseen prompt p, optionally augmented with representative data samples when available, with the frozen LLM to obtain the conditioning embedding $\mathbf{h}$. The projector predicts $(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}}) = \mathrm{Proj}(\mathbf{h})$, from which we sample $\mathbf{z} \sim \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}}^2)$. The decoder produces a generated weight-token sequence $\hat{\mathbf{W}} = \mathrm{Dec}(\mathbf{z})$, which is reassembled into parameter tensors and denormalized using the stored per-key statistics to recover the parameters $\hat{\theta}$.

4 Experiments

Our experiment design on ParaGen-Bench covers a broad spectrum: from the task perspective, ranging from classic tasks such as classification (§ 4.2), segmentation(§ 4.3), to recent advanced settings including mathematical reasoning, code generation, multitask multimodal understanding (§ 4.4), and reinforcement learning(§ 4.5); from the target model perspective, spanning full neural networks trained from scratch to LoRA adapters of large language and vision-language models.

4.1 EXPERIMENTAL SETUP

We evaluate ten recent representative parameter generation baselines, along with our proposed method LPG, on the tasks and datasets detailed in § 3.2, and categorize them into three paradigms:

- Diffusion-based methods generate weights through iterative denoising, including P-Diff Wang et al. (2024), RPG Wang et al. (2025b), D2NWG Soro et al. (2025b), Tina Li et al. (2025), and LoRAGen Huang et al. (2026);
- Hypernetwork-based methods generate weights in a single forward pass, including SANE (Schürholt et al., 2024), HyperTransformer (Zhmoginov et al., 2022), DnD (Liang et al., 2025), and T2L (Charakorn et al., 2025);
- LLM-conditioned methods use language-model representations as conditioning inputs, represented by NeuroGen (Wang et al., 2025a). We use official implementations when available, adapt D2NWG into an end-to-end pipeline, and reimplement Tina and HyperTransformer following their papers.

4.2 EVALUATION ON CLASSIFICATION TASK

Evaluation on Image Classification. Table 2 reports image classification accuracy on CIFAR-10 and TinyImageNet. On targets below 0.3M parameters, most methods generate usable weights close to the trained models. At MobileNetV3 with 4.2M parameters, P-Diff, D2NWG, TINA, SANE, and NeuroGen collapse to near-random accuracy. RPG reaches 89.80% and LPG reaches 90.32%. On the TinyImageNet targets with 11M to 28M parameters, RPG is the only method that produces usable weights, and a large gap to the trained models remains. Two factors explain this pattern. Firstly, scaling to larger targets increases the dimensionality of the generation problem. P-Diff and NeuroGen output one flattened vector whose length equals the parameter count, and D2NWG does the same under the model-wise vectorization used here. TINA and SANE split the weights into tokens but still model the full token sequence together. The second is the structure of the target. MobileNetV3 contains depthwise convolutions and BatchNorm running statistics, whose value ranges differ from standard layer weights and cannot be recovered accurately by a generator that applies the same normalization to every value. *Methods that generate the whole weight vector at once fail as the target grows or contains BatchNorm and depthwise layers, and RPG's recurrent design remains effective at larger scales because it generates the weights step by step with a fixed token width.*

Table 2: Full Parameter generation on image and text classification. Top-1 accuracy (%, ↑), mean ± std over 5 training seeds and 10 generations per seed. **T1**: diffusion-based; **T2**: hypernet-based; **T3**: LLM-conditioned. **n.s.**: not scalable. ✗: structurally incompatible. **Bold** and underline mark the best and second-best among generation methods. *Existing methods show two consistent limitations: (1) performance gaps increase on larger target models, and (2) architectures with more diverse layer types are more difficult to generate.*

	Method	Image Classification							Text Classification						
		CIFAR-10				TinyImageNet			SST-2				GoEmotions		
		CNN	ResNet	ViT	MNV3	RN18	MNV4	CNX-T	MLP	RNN	Trans.	BERT	ELE	DBv3	DRB
	Params (M)	0.003	0.172	0.268	4.215	11.27	21.85	27.97	0.008	0.008	0.794	3.225	9.55	21.65	43.14
	Trained Model	48.46	81.64	80.63	94.38	49.85	52.09	85.57	81.42	81.19	81.77	84.40	45.21	43.13	57.81
T1	P-Diff	48.52±0.18	75.45±2.52	**80.28±0.07**	10.57±0.00	0.50±0.00	0.50±0.00	0.50±0.00	81.02±0.19	**82.12±0.29**	79.21±0.32	49.08±0.00	0.00±0.00	0.00±0.00	0.00±0.00
T1	RPG	48.66±0.20	72.76±3.72	70.13±0.16	89.80±3.49	**43.99±4.96**	**36.25±0.53**	**74.74±0.28**	81.18±0.28	80.49±0.35	80.43±0.27	83.86±0.46	**42.76±0.38**	**37.41±0.42**	**52.84±0.29**
T1	D2NWG	**48.91±0.34**	75.06±1.74	79.36±0.63	10.00±0.00	0.50±0.00	0.50±0.00	0.50±0.00	81.18±0.17	81.00±0.08	64.05±10.51	49.89±0.83	9.83±2.73	4.28±1.57	7.10±2.07
T1	TINA	46.79±0.02	78.74±0.05	80.25±0.01	12.58±1.17	0.50±0.00	0.50±0.00	0.50±0.00	80.77±0.03	81.25±0.05	**81.35±0.05**	64.18±0.06	2.07±2.32	4.90±0.95	6.87±1.89
T2	SANE	47.27±0.98	49.55±17.31	64.44±1.25	11.72±1.57	0.50±0.00	0.50±0.00	0.49±0.03	**81.24±0.35**	81.42±0.55	78.43±0.50	78.12±1.86	0.58±0.88	5.41±3.42	6.44±2.22
T2	HyperTrans.	44.28±2.74	76.91±0.44	**n.s.**	**n.s.**	**n.s.**	**n.s.**	**n.s.**	✗	✗	✗	✗	✗	✗	✗
T3	NeuroGen	48.25±0.09	10.00±0.00	70.27±0.15	10.00±0.00	0.48±0.30	0.50±0.00	0.47±0.30	81.19±0.03	81.69±0.08	29.14±0.37	56.87±0.14	0.00±0.00	0.00±0.00	0.00±0.00
T3	LPG	48.21±0.14	**79.47±2.39**	79.44±0.20	**90.32±3.82**	0.50±0.00	0.50±0.00	0.50±0.00	80.34±0.14	78.34±0.14	80.09±0.21	**84.17±0.36**	9.27±0.00	8.19±0.00	5.28±0.00

Evaluation on Text Classification. Table 2 reports text classification results on SST-2 and GoEmotions. On SST-2, most methods generate usable weights for targets up to the 3.2M-parameter BERT-tiny, and LPG reaches 84.17% against 84.40% for the trained model. On GoEmotions with 9.5M to 43M parameters, RPG is the only method that produces usable weights, with 42.76%, 37.41%, and 52.84% against 45.21%, 43.13%, and 57.81% for the trained models. The comparison between BERT-tiny and MobileNetV3 separates the effect of structure from that of size. The two targets have similar parameter counts of 3.2M and 4.2M. SANE reaches 78.12% on BERT-tiny and 11.72% on MobileNetV3, and TINA reaches 64.18% and 12.58%. This is because BERT-tiny has a

more uniform parameter structure than MobileNetV3 and does not contain BatchNorm statistics or depthwise convolutions. *Targets built from a single layer type are easier to generate than targets of the same size with mixed layer types, and the scale limit observed on image classification also applies to text classification.*

Table 3: Parameter generation on image segmentation. Mean IoU (%, ↑), mean ± std over 5 training seeds and 10 generations per seed. **n.s.**: not scalable. ✗: structurally incompatible. **Bold** and underline mark the best and second-best among generation methods. *Small parameter errors can lead to large performance drops in image segmentation. At similar parameter scales, segmentation shows substantially larger gaps than classification.*

	Method	Oxford-IIIT Pet			ISIC2018			VOC2012		
		UNet	MNV3	ViT	UNet	MNV3	SegNet	SegF	FCN	DLV3
	Params (M)	0.227	0.291	1.245	0.227	0.291	0.787	27.36	33.01	39.70
	Trained Model	56.41	70.39	65.74	74.10	86.57	79.34	67.96	61.14	66.50
T1	P-Diff	52.67 ± 3.66	31.79 ± 9.93	63.94 ± 0.83	72.51 ± 1.02	37.01 ± 8.08	54.98 ± 17.56	1.22 ± 2.21	3.31 ± 2.71	3.49 ± 2.58
T1	RPG	50.29 ± 2.26	19.83 ± 1.91	**65.49 ± 0.14**	69.85 ± 2.09	37.37 ± 3.21	**77.60 ± 0.82**	**42.35 ± 2.81**	**39.20 ± 2.02**	**10.81 ± 1.06**
T1	D2NWG	25.24 ± 11.69	9.96 ± 0.00	62.49 ± 0.01	62.06 ± 15.47	40.29 ± 0.05	58.52 ± 16.12	5.41 ± 0.00	5.41 ± 0.00	5.41 ± 0.00
T1	TINA	**55.01 ± 0.27**	19.20 ± 0.34	62.58 ± 0.03	73.12 ± 0.07	39.80 ± 0.04	73.52 ± 0.15	0.07 ± 0.01	0.06 ± 0.01	0.11 ± 0.02
T2	SANE	39.18 ± 9.19	19.92 ± 0.75	59.07 ± 1.18	64.63 ± 1.53	41.27 ± 3.12	61.94 ± 6.31	7.90 ± 0.00	7.90 ± 0.00	7.90 ± 0.00
T2	HyperTrans.	54.31 ± 0.26	**n.s.**	**n.s.**	69.86 ± 0.05	**n.s.**	**n.s.**	**n.s.**	**n.s.**	**n.s.**
T3	NeuroGen	54.46 ± 1.04	19.25 ± 0.02	65.07 ± 0.08	63.95 ± 1.36	39.59 ± 0.02	76.65 ± 0.62	5.41 ± 0.00	3.41 ± 0.01	3.12 ± 0.04
T3	LPG	53.51 ± 2.17	**55.39 ± 7.41**	62.60 ± 1.58	**73.48 ± 0.94**	**53.47 ± 4.59**	75.68 ± 0.52	4.21 ± 1.26	4.13 ± 0.87	3.90 ± 1.52

4.3 Evaluation on Image Segmentation Task

Table 3 reports segmentation performance on Oxford-IIIT Pet, ISIC2018, and VOC2012. On the small targets, most methods recover weights close to the trained models, and the variance is much higher than in classification, with standard deviations of 17.56% mIoU for P-Diff on ISIC2018 SegNet and 11.69% for D2NWG on Oxford-Pet UNet. On VOC2012, every method except RPG collapses on SegFormer and FCN, and all methods fail on DeepLabV3. RPG retains 62.32% of the trained performance on SegFormer, against 87.34% on TinyImageNet ConvNeXt-Tiny at a similar parameter count. This difference comes from the task. A segmentation network produces a label for every pixel, and a small error in the weights changes the prediction of many pixels at once. A classifier produces one label per image, and the same error rarely changes the predicted label. *These results indicate that segmentation is substantially more sensitive to small errors in generated parameters than classification, leading to much larger downstream performance degradation.*

Table 4: Parameter generation on LoRA adaptation and reinforcement learning. Math (GSM8K) and coding (HumanEval) report Pass@1 (%, ↑). Multimodal understanding (ScienceQA, MMMU) reports accuracy (%, ↑). RL reports mean episodic return (↑). ✗: structurally incompatible. **Bold** and underline mark the best and second-best among generation methods. *LoRA generation remains effective across reasoning and multimodal tasks, while reinforcement learning is substantially more sensitive to parameter generation errors.*

	Method	LoRA Adaptation						Reinforcement Learning			
		GSM8K		HumanEval		ScienceQA	MMMU	LunarLander-v3		Acrobot-v1	
		Llama	Qwen3	Llama	Qwen3	Qwen3VL	Qwen3VL	PPO	DQN	PPO	DQN
	Params (M)	24.31	17.43	24.31	17.43	6.423	6.423	0.010	0.005	0.010	0.005
	Trained Model	59.07	77.98	32.32	51.22	92.23	41.78	222.3	237.4	-79.50	-78.90
T1	P-Diff	12.28 ± 0.47	73.39 ± 0.23	28.78 ± 0.80	29.51 ± 1.47	67.42 ± 0.50	36.64 ± 0.90	224.11 ± 1.04	-316.67 ± 48.77	-83.26 ± 3.99	-500.00 ± 0.00
T1	RPG	54.17 ± 1.53	78.04 ± 0.62	29.39 ± 0.80	49.51 ± 1.32	89.60 ± 0.29	40.71 ± 0.29	222.62 ± 16.55	-10.26 ± 139.11	-83.23 ± 4.85	-459.72 ± 127.38
T1	D2NWG	31.92 ± 20.92	51.86 ± 22.97	29.15 ± 0.51	16.71 ± 12.35	83.21 ± 3.52	38.22 ± 1.94	**249.69 ± 9.28**	-23.39 ± 45.42	-97.88 ± 17.38	-197.05 ± 41.88
T1	TINA	37.45 ± 0.49	76.20 ± 0.40	31.34 ± 0.33	47.80 ± 2.14	**91.65 ± 0.27**	38.58 ± 0.55	-293.33 ± 183.86	-477.93 ± 243.15	-215.09 ± 149.10	-500.00 ± 0.00
T1	LoRAGen	56.98 ± 1.73	**78.27 ± 0.82**	31.45 ± 0.68	50.36 ± 1.97	91.58 ± 2.43	40.27 ± 0.96	✗	✗	✗	✗
T2	SANE	51.39 ± 0.42	76.84 ± 0.65	30.86 ± 0.93	49.76 ± 1.27	85.88 ± 1.07	39.07 ± 0.61	223.61 ± 8.22	-225.48 ± 181.82	-86.41 ± 3.94	-372.08 ± 161.38
T2	DnD	52.63 ± 0.48	76.33 ± 0.55	30.42 ± 0.82	49.71 ± 1.14	89.10 ± 0.40	40.38 ± 0.87	✗	✗	✗	✗
T2	T2L	**57.83 ± 0.44**	77.33 ± 0.40	18.78 ± 1.25	43.78 ± 4.08	90.20 ± 0.26	**42.13 ± 0.58**	✗	✗	✗	✗
T3	NeuroGen	43.25 ± 0.41	77.31 ± 0.54	**35.13 ± 0.93**	**62.93 ± 1.00**	86.68 ± 0.14	38.84 ± 0.59	221.13 ± 15.18	**128.67 ± 30.26**	**-82.24 ± 3.66**	**-119.95 ± 42.33**
T3	LPG	32.47 ± 0.36	76.56 ± 0.47	30.16 ± 0.87	47.66 ± 1.40	84.26 ± 0.37	39.36 ± 0.63	216.75 ± 12.36	-109.10 ± 42.74	-84.82 ± 3.10	-423.90 ± 93.07

4.4 Evaluation on LoRA Adaptation Tasks

Table 4 reports LoRA generation on reasoning, coding, and multimodal tasks. At comparable parameter scales, LoRA adapters are substantially easier to generate than full model parameters, with some methods approaching trained-adapter performance. Methods built for LoRA generation lead on reasoning and multimodal tasks, with T2L at 57.83% on GSM8K with Llama against 59.07% trained and LoRAGen at 78.27% on GSM8K with Qwen3 against 77.98% trained. Their advantage comes from the training loss. T2L trains the hypernetwork with the downstream task loss, and LoRAGen supervises the full adaptation matrix $\Delta W = BA$ with direction and spectral losses, which avoids the non-uniqueness of the low-rank factors. *For training objectives, reconstructing the full adaptation matrix is more effective than reconstructing the LoRA matrices A and B separately, while downstream task loss provides an effective alternative.*

4.5 Evaluation on Reinforcement Learning Task

Table 4 reports parameter generation performance on reinforcement learning tasks. Generated PPO policies remain close to the trained baselines across both environments, whereas DQN shows much larger performance gaps and all methods fail to match the trained policies. On LunarLander-v3, NeuroGen reaches 128.67 and RPG drops to -10.26, compared with 237.4 for the trained model. In DQN, the agent takes the action with the highest Q-value, and a small error in the parameters changes which action ranks first and leads to a different policy. *Learning to generate reinforcement learning policies is highly sensitive to small parameter errors, which can alter action values or rankings and lead to substantially different action selections and returns.*

4.6 Efficiency Study

We compare end-to-end generation efficiency on Qwen3-1.7B LoRA adaptation in Figure 4. Diffusion-based methods remain competitive in accuracy across Table 4 but cost far more compute per generation. P-Diff flattens the entire weight and requires to reconstruct the full weight at once, which leads to 25.9K GFLOPs per generation and a throughput of only 0.12 gen/s. RPG generates weights token by token through its recurrent Mamba backbone, and each token further requires multi-step denoising. This sequential nature prevents parallelization and becomes increasingly expensive as the target parameter count grows, resulting in 21.9K GFLOPs and 0.05 gen/s. TINA combines attention-based processing with multi-step denoising across the full weight tensor, reaching the highest compute at 80.7K GFLOPs and the lowest throughput at 0.04 gen/s. In contrast, hypernetwork-based and LLM-conditioned methods avoid iterative denoising entirely. SANE, DnD, NeuroGen, and `LPG` use lightweight decoders such as convolutional or MLP hypernetworks that produce weights in a single forward pass.

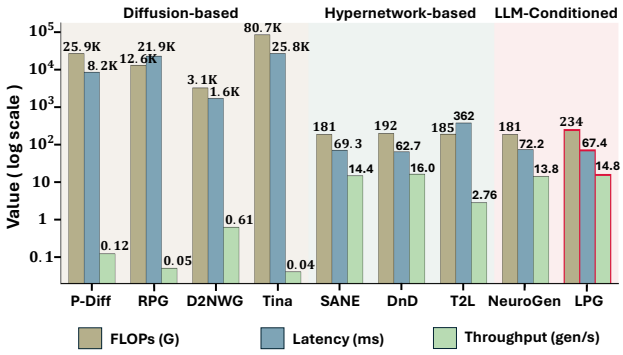


Figure 4: Efficiency comparison on Qwen3-1.7B LoRA adaptation. FLOPs (G): compute per adapter; latency (ms): wall-clock time; throughput (gen/s): adapters generated per second.

4.7 Ablation Study

We ablate three components of `LPG` across four tasks in Figure 5: per-layer tokenization versus flat-vector chunking, the validity mask that excludes padding from reconstruction loss, and the log-pivot transform for BatchNorm running variance. Removing per-layer tokenization causes the largest drops, with MobileNetV3 on CIFAR-10 falling from 93.24% to 10.00% and Oxford-IIIT Pet from 59.85% to 9.53%, confirming that preserving layer structure is critical for architectures with heterogeneous layers.

The BN transform is equally important on these settings but has no effect on SST-2 and ScienceQA, whose target architectures contain no BatchNorm. The validity mask has its largest impact on full weight generation, where CIFAR-10 MobileNetV3 drops from 93.24% to 51.84% without it, but has negligible effect on LoRA adaptation where padding is minimal. More ablation study experiment results can be found in Appendix B.2. *Each component plays an effective role across target models, and all three are necessary for* `LPG` *on the parameter generation tasks.*

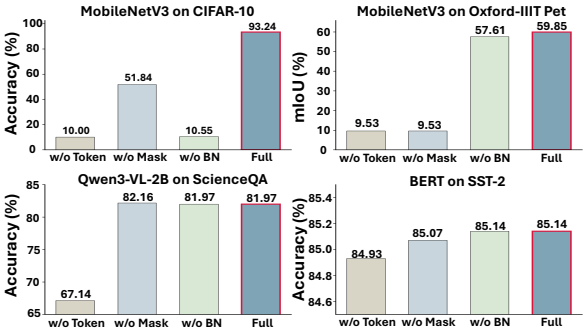


Figure 5: Ablation study of our proposed `LPG`.

5 Findings, Limitations, and Future Directions

Findings. We identify three main findings across the benchmark from three perspectives:

- *Task perspective*: Compared with image classification, text classification, and LoRA adaptation, the evaluated methods produce less reliable parameters for image segmentation and reinforcement learning, where small errors in generated parameters can substantially affect pixel-level predictions or action selection.
- *Training perspective*: Better training objectives improve LoRA generation: reconstructing the full adaptation matrix is more effective than reconstructing the matrices A and B separately, while direct optimization of the downstream task loss provides another effective training objective (§ 4.4).
- *Target model size perspective*: Performance gaps increase as target models grow larger, and architectures with mixed layer types show even larger gaps, with several examined methods reaching near-random performance (§ 4.2).

Limitations. (1) Although ParaGen-Bench covers commonly representative task domains, it does not include several specialized domains such as finance, law, and healthcare, which limits the breadth of our conclusions. (2) `LPG` is simple to implement and is not tied to a specific task or dataset. In principle, the framework remains applicable to larger models. However, full-parameter generation for billion-scale models incurs substantial computational and memory costs due to the sheer size of the target models. Given these resource requirements, we have not evaluated billion-scale models in this work at this moment. (3) Our prompt set already includes diverse controlled variations, but robustness to noisy, ambiguous, or adversarial instructions remains underexplored.

Future Directions. Based on the findings and limitations, we suggest to explore following directions: (1) Future exploration should include more domain-specific tasks, e.g. finance, law, and healthcare. Zooming into domain-specific tasks would further reveal how well parameter generation generalizes to specialized knowledge and domain-specific requirements. (2) Future exploration should develop more effective and efficient training strategies for parameter generation. Advancing the training process would further improve the consistency of downstream performance and reduce computational cost. (3) Future exploration should strengthen the robustness of conditional parameter generation against adversarial and poisoning attacks. Improving robustness would further enhance the reliability of generated parameters under corrupted or malicious conditions.

6 Conclusion

In this paper, we introduce ParaGen-Bench, the first comprehensive benchmark for neural network parameter generation, comprising 44,000 trained checkpoints across 41 architectures, 29 datasets, and seven task domains. Through a unified evaluation of ten representative parameter generation methods, we identify consistent performance gaps on large target models and architectures with diverse layer types, strong sensitivity to parameter errors in image segmentation and reinforcement learning, and distinct effects of supervision objectives in LoRA generation. We further propose `LPG`, a simple yet effective two-stage baseline combining structure-aware weight tokenization with prompt-conditioned latent generation, which achieves competitive results across diverse settings and serves as a reference point for future work on ParaGen-Bench. Finally, we summarize the main observations, limitations of this field, and discuss promising directions for future research.

AI USE STATEMENT

We used Anthropic Claude as an assistive tool during the writing process to support manuscript drafting and editing, including improving grammar, wording, and LaTeX formatting. All AI-assisted text was reviewed by the authors, who made the final scientific and methodological decisions and take full responsibility for the correctness and content of the paper.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide detailed descriptions of the benchmark construction, data processing pipeline, evaluation protocols, and experimental settings in the appendix. Our anonymized implementation, including preprocessing, training, and evaluation scripts, is available at `https://anonymous.4open.science/r/l2w-630C`. Due to the size of the benchmark, the complete dataset is hosted in an anonymous public repository at `https://huggingface.co/datasets/AnonymousAuthorEE33/ParaGen-Bench`. All reported experiments can be reproduced using the released code, configurations, and data resources.

REFERENCES

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2357–2367, 2019.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Gabriel J Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern recognition letters*, 30(2):88–97, 2009.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-loRA: Instant transformer adaption. In *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=zWskCdu3QA`.

Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Noel Codella, Veronica Rotemberg, Philipp Tschandl, M Emre Celebi, Stephen Dusza, David Gutman, Brian Helba, Aadi Kalloo, Konstantinos Liopyris, Michael Marchetti, et al. Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic). *arXiv preprint arXiv:1902.03368*, 2019.

Dorottya Demszky, Dana Movshovitz-Attias, Jeongwoo Ko, Alan Cowen, Gaurav Nemade, and Sujith Ravi. Goemotions: A dataset of fine-grained emotions. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 4040–4054, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

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 *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

Damian Falk, Léo Meynent, Florence Pfammatter, Konstantin Schürholt, and Damian Borth. A model zoo of vision transformers. In *Workshop on Neural Network Weights as a New Data Modality*, 2025. URL https://openreview.net/forum?id=HyTVpctexy.

Rickard Brüel Gabrielsson, Jiacheng Zhu, Onkar Bhardwaj, Leshem Choshen, Kristjan Greenewald, Mikhail Yurochkin, and Justin Solomon. Compress then serve: Serving thousands of LoRA adapters with little overhead. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 18062–18095. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/gabrielsson25a.html.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkpACe1lx.

Xiaolong Han, Zehong Wang, Bo Zhao, Binchi Zhang, Jundong Li, Damian Borth, Rose Yu, Haggai Maron, Yanfang Ye, Lu Yin, and Ferrante Neri. A survey of weight space learning: Understanding, representation, and generation, 2026. URL https://arxiv.org/abs/2603.10090.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTav3: Improving deBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=sE7-XhLxHA.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021a. URL https://openreview.net/forum?id=sD93GOzH3i5.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021b.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1314–1324, 2019.

Hao Huang, Jingtao Ding, Mengqi Liao, Xin Wang, Jinyang Ban, Yuan Yuan, Huaiyu Wan, and Yong Li. LoRAGen: Structure-aware weight space learning for loRA generation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=mrafO7aTYj.

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.

Rana Muhammad Shahroz Khan, Dongwen Tang, Pingzhi Li, Kai Wang, and Tianlong Chen. Oral: Prompting your large-scale loras via conditional recurrent diffusion. *arXiv preprint arXiv:2503.24354*, 2025.

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

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

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Xin Li and Dan Roth. Learning question classifiers. In *Coling 2002: The 19th international conference on computational linguistics*, 2002.

Zexi Li, Lingzhi Gao, and Chao Wu. Text-to-model: Text-conditioned neural network diffusion for train-once-for-all personalization. In *Workshop on Neural Network Weights as a New Data Modality*, 2025. URL https://openreview.net/forum?id=Sg4OTjUPgB.

Zhiyuan Liang, Dongwen Tang, Yuhao Zhou, Xuanlei Zhao, Mingjia Shi, Wangbo Zhao, Zekai Li, Peihao Wang, Konstantin Schürholt, Damian Borth, Michael M. Bronstein, Yang You, Zhangyang Wang, and Kai Wang. Drag-and-drop LLMs: Zero-shot prompt-to-weights. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=fTkBZLxBzV.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 158–167, 2017.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 11966–11976. IEEE, 2022.

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

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in neural information processing systems*, 35:2507–2521, 2022.

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.

Ahmed Masry, Jia Qing Tan, Shafiq Joty, Enamul Hoque, et al. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the association for computational linguistics: ACL 2022*, pp. 2263–2279, 2022.

Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The role of context for object detection and semantic segmentation in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 4. Granada, 2011.

Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

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.

Jason Phang, Yi Mao, Pengcheng He, and Weizhu Chen. Hypertuning: Toward adapting large language models without back-propagation. In *International Conference on Machine Learning*, pp. 27854–27875. pmlr, 2023.

Danfeng Qin, Chas Leichner, Manolis Delakis, Marco Fornoni, Shixin Luo, Fan Yang, Weijun Wang, Colby Banbury, Chengxi Ye, Berkin Akin, et al. Mobilenetv4: Universal models for the mobile ecosystem. In *European conference on computer vision*, pp. 78–96. Springer, 2024.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Neale Ratzlaff and Li Fuxin. HyperGAN: A generative model for diverse, performant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5361–5369. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/ratzlaff19a.html.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 10674–10685. ieee, 2022.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

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 *Thirty-Sixth Conference on Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, September 2022.

Konstantin Schürholt, Michael W. Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43947–43966. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/schurholt24a.html.

Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. A-okvqa: A benchmark for visual question answering using world knowledge. In *European conference on computer vision*, pp. 146–162. Springer, 2022.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Bedionita Soro, Bruno Andreis, Song Chong, and Sung Ju Hwang. Instruction-guided autoregressive neural network parameter generation. In *Workshop on Neural Network Weights as a New Data Modality*, 2025a. URL https://openreview.net/forum?id=QutFK34ea1.

Bedionita Soro, Bruno Andreis, Hayeon Lee, Wonyong Jeong, Song Chong, Frank Hutter, and Sung Ju Hwang. Diffusion-based neural network weights generation. In *International Conference on Learning Representations*, volume 2025, pp. 99644–99675, 2025b.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Mark Towers, Ariel Kwiatkowski, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Kallinteris Andreas, Markus Krimmel, Arjun KG, Rodrigo De Lazcano Perez-Vicente, J K Terry, Andrea Pierré, Sander V Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026. URL https://openreview.net/forum?id=qPMLvJxtPK.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models, 2019.

Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020.

Jiaqi Wang, Yusen Zhang, and Xi Li. Neurogen: Neural network parameter generation via large language models. *arXiv preprint arXiv:2505.12470*, 2025a.

Kai Wang, Dongwen Tang, Boya Zeng, Yida Yin, Zhaopan Xu, Yukun Zhou, Zelin Zang, Trevor Darrell, Zhuang Liu, and Yang You. Neural network diffusion. *arXiv preprint arXiv:2402.13144*, 2024.

Kai Wang, Dongwen Tang, Wangbo Zhao, Konstantin Schürholt, Zhangyang Wang, and Yang You. Scaling up parameter generation: A recurrent diffusion approach. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=65llKR17s4.

Zhangyang Wang, Peihao Wang, and Kai Wang. Position: Weight space should be a first-class generative AI modality. In *Forty-third International Conference on Machine Learning Position Paper Track*, 2026. URL https://openreview.net/forum?id=vu0ousWLde.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yicheng Xiao, Lin Song, Rui Yang, Cheng Cheng, Yixiao Ge, Xiu Li, and Ying Shan. LoRA-gen: Specializing large language model via online loRA generation. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=oZM5g4IvmS.

Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in neural information processing systems*, 34:12077–12090, 2021.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6980–7008, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9556–9567, 2024.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

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

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, et al. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*, 2025.

A Implementation Details

We conduct all experiments on four NVIDIA H100 NVL GPUs with 96 GB memory each, using PyTorch 2.10.0 and Python 3.12.13.

A.1 Baselines Evaluated

A.1.1 Diffusion-based Methods

P-Diff (Wang et al., 2024). P-Diff is an unconditional generator. It flattens a selected subset of trained parameters into 1D vectors and trains an autoencoder to reconstruct them, then trains a standard latent diffusion model (Ho et al., 2020; Rombach et al., 2022) to synthesize these latent representations from random noise. At inference it takes Gaussian noise as input and outputs the parameter subset through the frozen decoder, which is loaded back into the target network.

RPG (Wang et al., 2025b). RPG is an unconditional generator that scales to full network parameters. It partitions the parameters into non-overlapping tokens with layer-wise normalization, permutation states, and position embeddings, and a recurrent model produces per-token prototypes that capture inter-token relationships. These prototypes serve as conditions for a 1D diffusion process, which takes random noise as input and outputs the complete parameter set.

D2NWG (Soro et al., 2025b). D2NWG is a dataset-conditioned generator. A VAE (Kingma & Welling, 2013) encodes model weights into latents, while a Set Transformer encodes the target dataset into a dataset embedding from per-sample features. A latent diffusion model is then trained on the weight latents conditioned on this embedding, taking data samples from the target task as input and outputting weights through the VAE decoder.

TINA (Li et al., 2025). TINA is a prompt-conditioned generator. A CLIP text encoder embeds the natural-language task description, and a diffusion transformer (DiT) conditioned on this embedding denoises random noise into model parameters. It takes the task prompt as input and outputs the parameters of the target network, with a classification sequence padding strategy that allows varying numbers of output classes.

LoRAGen (Huang et al., 2026). LoRAGen is a prompt-conditioned latent diffusion method for LoRA generation. It first trains a LoRA autoencoder to encode adapters into a latent space, with supervision applied to the full adaptation matrix $\Delta W = BA$ to avoid the non-uniqueness of the low-rank factors A and B. A module-aware Mixture-of-Experts decoder models the heterogeneous weight distributions across different network modules. In the second stage, a diffusion transformer conditioned on a natural-language task description generates the latent representation from noise, which is decoded into task-specific LoRA parameters.

A.1.2 Hypernetwork-based Methods

SANE (Schürholt et al., 2024). SANE is an unconditional generator built on a sequential autoencoder for neural embeddings. It reshapes model weights into 2D matrices and slices them into fixed-size tokens, each augmented with a positional embedding encoding the sequence, layer, and within-layer index, and trains an encoder-decoder with a combination of masked reconstruction and contrastive losses. Unlike diffusion methods, generation does not start from noise: SANE samples token embeddings from the learned representation space via kernel density estimation over existing model embeddings, and decodes them sequentially into the target network.

HyperTransformer (Zhmoginov et al., 2022). HyperTransformer is a data-conditioned generator originally designed for few-shot learning. A transformer weight generator takes a support set of labeled samples as input, where each token concatenates an image embedding, an activation embedding from the previous layer, and the sample label, and predicts the weights of the target CNN layer by layer in a single forward pass. The whole generator is trained end-to-end by backpropagating the query-set loss of the generated network.

T2L (Charakorn et al., 2025). T2L is a hypernetwork-based method for LoRA generation. A frozen text encoder embeds the task description, and the concatenation of this embedding with learnable module-type and layer-index embeddings is mapped through an MLP to the low-rank matrices A and B of every target layer in a single forward pass. It is trained either by reconstructing pre-trained adapters or by supervised fine-tuning on the downstream task.

DnD (Liang et al., 2025). DnD is a hypernetwork-based method that maps unlabeled task prompts directly to LoRA weights. A frozen text encoder distills a batch of prompts into condition embeddings, and a cascaded hyper-convolutional decoder expands them into the full set of LoRA matrices for every layer. It is trained on randomly paired prompt batches and LoRA checkpoints with a weighted MSE loss against the tokenized target weights.

A.1.3 LLM-conditioned Methods

NeuroGen (Wang et al., 2025a). NeuroGen is an LLM-conditioned method with a two-stage pipeline. Learnable special tokens are fed into a LoRA-tuned LLM together with the instruction, and an MLP projection maps the output hidden states to the target parameters. Stage one aligns the generated weights with trained checkpoints, and stage two tunes on task-specific instructions and data samples. It takes an instruction and data samples as input and outputs the full parameter set in a single pass.

`LPG` (Ours). `LPG` is a prompt-conditioned method with a two-stage pipeline. Stage one trains TokenVAE, a weight autoencoder over per-layer tokens with structured positional encoding, masked reconstruction, and BatchNorm reparameterization. Stage two trains an MLP projector that maps prompt embeddings from a frozen LLM into the learned latent space. It takes a task prompt as input and outputs the target parameters through the TokenVAE decoder.

A.2 Details of Selected Dataset and Architectures

Table 5 summarizes the target architectures, datasets, generation targets, and evaluation metrics across ParaGen-Bench. The benchmark covers seven task domains with 41 target architectures and 29 datasets, and every architecture–dataset combination within a domain forms one setting, giving 285 settings in total, where the 12 subsets of KodCode-V1 count as separate datasets. The benchmark spans both full-parameter generation and LoRA generation. Image classification, text classification, image segmentation, and reinforcement learning use full model parameters as generation targets, while mathematical reasoning, code generation, and multimodal understanding use LoRA adapters applied to pretrained language and vision-language models. Architectures with the suffix *mini* are lightweight variants implemented from scratch for this benchmark, while the remaining models are drawn from widely used model families and open-source implementations.

A.2.1 Datasets

Image Classification. We include MNIST (LeCun et al., 1998), CIFAR-10 (Krizhevsky et al., 2009), SVHN (Netzer et al., 2011), Fashion-MNIST (Xiao et al., 2017), and TinyImageNet (Russakovsky et al., 2015).

Text Classification. We include SST-2 (Socher et al., 2013), AG News (Zhang et al., 2015), IMDB (Maas et al., 2011), TREC (Li & Roth, 2002), and GoEmotions (Demszky et al., 2020).

Image Segmentation. We include Oxford-IIIT Pet (Parkhi et al., 2012), ISIC2018 (Codella et al., 2019), CamVid (Brostow et al., 2009), Pascal Context (Mottaghi et al., 2014), and VOC2012 (Everingham et al., 2015).

Mathematical Reasoning. We include GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021b), AQuA-RAT (Ling et al., 2017), and MathQA (Amini et al., 2019).

Code Generation. We include MBPP (Austin et al., 2021), APPS (Hendrycks et al., 2021a), KodCode-V1 (Xu et al., 2025), and additionally evaluate generation methods on HumanEval (Chen et al., 2021).

Multimodal Understanding. We include ScienceQA (Lu et al., 2022), A-OKVQA (Schwenk et al., 2022), ChartQA (Masry et al., 2022), and MMMU (Yue et al., 2024).

Reinforcement Learning. We include CartPole-v1, Acrobot-v1, and LunarLander-v3 from Gymnasium (Towers et al., 2026).

A.2.2 ARCHITECTURES

Image Classification. We include MLP-mini, CNN-mini, ResNet-mini, ViT-mini, ResNet-18 (He et al., 2016), MobileNetV3-Large (Howard et al., 2019), ConvNeXt-Pico (Liu et al., 2022), ViT-Small (Dosovitskiy et al., 2021), ConvNeXt-Tiny (Liu et al., 2022), and MobileNetV4-Hybrid-Large-075 (Qin et al., 2024).

Text Classification. We include MLP-mini, RNN, LSTM, Transformer-mini, BERT-tiny (Turc et al., 2019), BERT-Base (Devlin et al., 2019), DistilBERT (Sanh et al., 2019), RoBERTa-Base (Liu et al., 2019), GPT-2 Small (Radford et al., 2019), T5-Small (Raffel et al., 2020), DeBERTa-v3-Small (He et al., 2023), ELECTRA-Small (Clark et al., 2020), DeBERTa-v3-XSmall (He et al., 2023), and DistilRoBERTa-Base.

Image Segmentation. We include UNet-mini, SegNet-mini, ViT-mini, MobileNetV3-Small (Howard et al., 2019), DeepLabV3-ResNet50 (Chen et al., 2017), FCN-ResNet50 (Long et al., 2015), and SegFormer-B2 (Xie et al., 2021).

Mathematical Reasoning and Code Generation. We use Llama-3.2-1B (Grattafiori et al., 2024), Llama-3.2-3B (Grattafiori et al., 2024), Qwen3-1.7B-Base (Yang et al., 2025), Qwen3-4B-Base (Yang et al., 2025), Gemma-2-2B (Team et al., 2024), and Gemma-3-1B (Team et al., 2025) as target models for LoRA generation.

Multimodal Understanding. We use Qwen3-VL-2B-Instruct (Bai et al., 2025), Qwen3-VL-4B-Instruct (Bai et al., 2025), InternVL3-2B (Zhu et al., 2025), and Gemma-3-4B-IT (Team et al., 2025).

Reinforcement Learning. We use MLP-based PPO and DQN agents as full-parameter generation targets.

Table 5: Coverage and statistics of PARAGEN-BENCH across seven task domains. Architectures with the suffix *mini* are lightweight variants implemented from scratch for this benchmark. Full denotes full-weight generation and LoRA denotes low-rank adapter generation.

Domain	Architectures	Datasets	Target	Metric	# Settings	# Ckpts	# Prompts
Image classification	MLP-mini, CNN-mini, ResNet-mini, ViT-mini, ResNet-18, MobileNetV3-Large, ConvNeXt-Pico, ViT-Small, ConvNeXt-Tiny, MobileNetV4-Hybrid-Large-075 (10)	MNIST, CIFAR-10, SVHN, Fashion-MNIST, TinyImageNet (5)	Full	Accuracy	50	10,000	2,500
Text classification	MLP-mini, RNN, LSTM, Transformer-mini, BERT-tiny, BERT-Base, DistilBERT, RoBERTa-Base, GPT-2 Small, T5-Small, DeBERTa-v3-Small, ELECTRA-Small, DeBERTa-v3-XSmall, DistilRoBERTa-Base (14)	SST-2, AG-NEWS, IMDB, TREC, GoEmotions (5)	Full	Accuracy, micro-F1, macro-F1	70	14,000	3,500
Image segmentation	UNet-mini, SegNet-mini, ViT-mini, MobileNetV3-Small, DeepLabV3-ResNet50, FCN-ResNet50, SegFormer-B2 (7)	Oxford-IIIT Pet, ISIC2018, CamVid, Pascal Context, VOC2012 (5)	Full	mIoU, Dice	35	7,000	1,750
Mathematical reasoning	Llama-3.2-1B, Llama-3.2-3B, Qwen3-1.7B-Base, Qwen3-4B-Base, Gemma-2-2B, Gemma-3-1B (6)	GSM8K, MATH, AQuA-RAT, MathQA (4)	LoRA	Pass@1, Accuracy	24	2,400	1,200
Code generation	Llama-3.2-1B, Llama-3.2-3B, Qwen3-1.7B-Base, Qwen3-4B-Base, Gemma-2-2B, Gemma-3-1B (6)	MBPP, APPS, KodCode-V1 (12 subsets) (3)	LoRA	Pass@1	84	8,400	4,200
Multimodal understanding	Qwen3-VL-2B-Instruct, Qwen3-VL-4B-Instruct, InternVL3-2B, Gemma-3-4B-IT (4)	ScienceQA, A-OKVQA, ChartQA, MMMU (4)	LoRA	Accuracy	16	1,600	800
Reinforcement learning	MLP-based PPO and DQN agents (2)	CartPole-v1, Acrobot-v1, LunarLander-v3 (3)	Full	Episodic return	6	600	300
Total	**41 architectures**	**29 datasets**	–	–	**285**	**44,000**	**14,250**

A.3 BENCHMARK CONSTRUCTION

Full-parameter targets. For image classification, text classification, image segmentation, and reinforcement learning, the generation target is the full model state required for inference. We first train each target network using a task- and architecture-specific recipe to obtain a converged reference checkpoint. For classification and segmentation, we initialize a new AdamW optimizer from this checkpoint and continue training for 200 optimization steps using the same batch size as in the initial

training stage and a cosine learning-rate schedule that decays to 1×10^{-5}. We save the complete model state after every optimization step, yielding 200 checkpoints per architecture–dataset condition. For reinforcement learning, we continue training the converged agent for 100 policy updates with a reduced learning rate and save the agent after every update, yielding 100 checkpoints per condition.

LoRA targets. For mathematical reasoning, code generation, and multimodal understanding, we use LoRA adapters as the parameter-generation targets while keeping the pretrained backbone frozen. For a pretrained linear transformation $\mathbf{W}_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, LoRA parameterizes the adapted weight as

$$\mathbf{W} = \mathbf{W}_0 + \frac{\alpha}{r}\mathbf{B}\mathbf{A},$$

where $\mathbf{A} \in \mathbb{R}^{r \times d_{\text{in}}}$ and $\mathbf{B} \in \mathbb{R}^{d_{\text{out}} \times r}$ are the trainable parameters, $r = 16$ is the LoRA rank, and $\alpha = 32$ is the scaling factor. For language models, we apply LoRA to `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`. For vision-language models, the vision encoder is kept frozen, and LoRA is applied only to `q_proj`, `k_proj`, `v_proj`, and `o_proj` of the language model.

Inspired by the local checkpoint collection strategy of DnD (Liang et al., 2025), we use a validation-guided two-stage protocol. In Stage 1, the adapter is trained with a learning rate of 1×10^{-4}, an effective batch size of 16, and a maximum sequence length of 768. We evaluate checkpoints at 40%, 70%, and 100% of the scheduled training budget on the validation split and select the collection anchor according to validation performance. In Stage 2, we resume training from the selected anchor with a newly initialized optimizer and collect 100 consecutive LoRA checkpoints after a short stabilization period, saving one checkpoint per optimization step.

Prompt construction. For each architecture–dataset pair, we generate natural-language prompts with Qwen3-VL-32B-Instruct (Bai et al., 2025). Generation starts from a base template populated with the model and dataset metadata, which contains the target architecture, task type, and dataset properties. For LoRA tasks, the metadata also includes the task definition and a few examples from the training split. Prompts are generated from five perspectives, specification-oriented, task-centric, data-centric, name-masked, and user-request descriptions, which together cover the three axes of architecture specificity, task granularity, and user intent in Section 3.2. The name-masked perspective omits the architecture and dataset names and describes the target by its function and data. Generated prompts pass two filters. A consistency check with Qwen3-VL-32B-Instruct compares each prompt against its metadata and removes unsupported or contradictory content. Semantic filtering removes near-duplicate and underspecified prompts, and we manually inspect a random sample of the retained prompts. For each pair we keep 50 validated prompts and split them into 30 training, 10 validation, and 10 test prompts. The prompt split is independent of the task-data split and evaluates generalization to unseen descriptions of the same pair.

Quality control and metadata. All collected checkpoints are evaluated using the corresponding validation metric. Failed or numerically unstable checkpoints are discarded, and the collection stage continues until each pool reaches its target size. For each checkpoint, we record its architecture, dataset or task condition, training configuration, optimization step, validation performance, and parameter statistics. For LoRA targets, we additionally record the base-model identifier, rank, scaling factor, and target modules.

Checkpoint pool statistics. Table 6 reports the validation performance of representative checkpoint pools. Every pool stays in a narrow band around a working solution. The standard deviation is at most 1.07 points across all listed pools, and the range between Q10 and Q90 is 0.30 mIoU for VOC2012 SegFormer, 1.27 points for SST-2 BERT-tiny, and 2.44 points for GSM8K on Qwen3-1.7B-Base. The trained model in the main tables is the best validation checkpoint from the first training stage, and the pool collected after it varies around that point, so the pool maximum stays close to the trained model. These statistics show that each pool provides many working checkpoints of the same target with local variation from optimization, and no pool contains early or failed training states.

A.4 Generator Training Configuration

P-Diff. We follow the released P-Diff training setup and use a 1D-convolutional VAE with encoder channels $[64, 128, 256, 256, 32]$ and a 768-dimensional latent representation when the released con-

Table 6: Statistics of representative checkpoint pools. N is the number of evaluated checkpoints. The trained model in Tables 2 to 4 is the best validation checkpoint from the first training stage, and the collected pool varies around it.

Task	Dataset / Architecture	Metric	N	Mean	Std.	Q10	Q50	Q90	Min	Max
Text cls.	SST-2 / MLP	Acc	200	81.04	0.46	80.49	81.02	81.65	79.59	82.11
	SST-2 / BERT-tiny	Acc	200	82.58	0.48	81.88	82.68	83.15	81.08	83.72
Image seg.	Oxford-Pet / MNV3	mIoU	200	69.72	0.42	69.06	69.88	70.12	68.06	70.27
	VOC2012 / SegFormer	mIoU	200	67.71	0.11	67.56	67.71	67.86	67.38	67.96
Math (LoRA)	GSM8K / Qwen3-1.7B-Base	Acc	100	77.30	0.99	76.19	77.15	78.63	75.63	79.27
	MATH / Qwen3-1.7B-Base	Acc	100	66.28	0.86	65.24	66.30	67.26	64.80	68.40
	GSM8K / Llama-3.2-3B	Acc	100	57.69	1.07	56.34	57.59	59.30	55.20	60.44
Code (LoRA)	KodCode Data-Structure / Qwen3-1.7B-Base	pass@1	100	66.04	0.75	65.40	66.00	67.24	64.60	67.60
	KodCode LeetCode / Qwen3-1.7B-Base	pass@1	100	32.04	0.91	30.96	32.00	33.12	30.60	34.60

figuration is directly applicable. For larger target parameter vectors, we use deeper and narrower encoder configurations to control the resulting generator size. We train with AdamW using learning rates of 10^{-4} for the diffusion model and 2×10^{-5} for the VAE (5×10^{-6} for segmentation targets). The default training budget is 2K VAE steps followed by 8K diffusion steps; for LoRA targets, we use 3K and 20K steps, respectively. Batch sizes range from 1 to 8 across target configurations. Target-specific settings are reported in Table 7.

Table 7: P-Diff configurations used in our experiments. P denotes the number of generated scalars. Steps report autoencoder and diffusion training steps.

Target	P	Encoder channels / latent	Generator	Batch	Steps
CIFAR-10 / CNN-mini	0.003M	$[64, 128, 256, 256, 32]$ / 768	249M	8	2K + 8K
Oxford-Pet / ViT-mini	1.245M	$[64, 128, 256, 256, 32]$ / 768	1.93B	4	2K + 8K
VOC2012 / SegFormer	27.36M	$[16, 32, 64, 128, 256, 256, 256, 8]$ / 2048	3.54B	1	2K + 8K

RPG. We follow the released RPG training setup and use a Mamba backbone with a 1D-convolutional diffusion head. We train with AdamW, cosine learning-rate decay, and no weight decay. The backbone width, state size, batch size, learning rate, and training budget are configured for each target; the LoRA setting additionally uses four Mamba layers and a wider diffusion head. target-specific configurations are reported in Table 8.

Table 8: RPG configurations used in our experiments. P denotes the number of generated scalars and d the backbone width.

Target	P	Width / state size	Generator	Batch	LR	Steps
CIFAR-10 / CNN-mini	0.003M	256 / 32	1.22M	8	1×10^{-4}	30K
CIFAR-10 / ViT-mini	0.268M	2048 / 128	61.2M	4	3×10^{-5}	10K
Oxford-Pet / ViT-mini	1.245M	2048 / 128	61.2M	4	3×10^{-5}	8K
VOC2012 / SegFormer-B2	27.36M	8192 / 128	953M	8	3×10^{-5}	8K
Qwen3-1.7B-Base / LoRA	17.43M	4096 / 256 (4 layers)	451M	2	3×10^{-5}	120K

D2NWG. We follow the released three-stage D2NWG training pipeline, consisting of a weight VAE, a dataset encoder, and a conditional latent-diffusion model. We use a $4 \times 4 \times 4$ latent representation with KL weight 10^{-6} and a diffusion UNet with 128 base channels, channel multipliers $[1, 2]$, two residual blocks per level, and $T{=}1024$. For larger target parameter vectors, we use a narrower VAE with 32 base channels and linear lifting to $2 \times 16 \times 16$. We train the VAE with Adam using a learning rate of 10^{-3}, the dataset encoder with AdamW at 4×10^{-4}, and the diffusion model with AdamW at 5×10^{-5} with 1K warmup steps. Representative target-specific settings are reported in Table 9.

TINA. We re-implement TINA following the configuration described in the paper, using a text-conditioned diffusion transformer over weight tokens. Target weights are standardized per layer and partitioned into layer-local chunks, while task descriptions are encoded by a frozen CLIP ViT-B/32

Table 9: D2NWG configurations used in our experiments. P denotes the number of generated scalars.

Target	P	VAE latent	VAE	Diffusion
CIFAR-10 / CNN-mini	0.003M	$4 \times 4 \times 4$	32 base channels	128 base channels, $[1, 2]$
CIFAR-10 / ResNet-mini	0.172M	$4 \times 4 \times 4$	32 base channels	128 base channels, $[1, 2]$

text encoder. For vision and text-classification targets, we use a transformer with width 512, depth 8, and 8 attention heads; for LoRA targets, we use width 1024, depth 12, and 16 heads. We train with AdamW using a learning rate of 4×10^{-4}, 200 warmup steps followed by cosine decay, weight decay 0.01, gradient clipping at 0.1, and batch sizes of 4–32 for 300–500 epochs. We use T=1000 diffusion steps, min-SNR weighting with γ=5, and EMA with decay 0.999. Target-specific settings are reported in Table 10.

Table 10: Representative TINA configurations used in our experiments. P denotes the number of generated scalars. Generator size excludes the frozen CLIP text encoder.

Target	P	Chunk / width / depth / heads	Generator	Batch	Epochs
CIFAR-10 / CNN-mini	0.003M	128 / 512 / 8 / 8	56.7M	32	300
CIFAR-10 / ViT-mini	0.268M	512 / 512 / 8 / 8	58.1M	16	300
Qwen3-1.7B-Base / LoRA	17.43M	4096 / 1024 / 12 / 16	345.7M	4	500
Llama-3.2-3B / LoRA	24.31M	4096 / 1024 / 12 / 16	349.1M	4	305

SANE. We follow the released SANE pretraining recipe and use a GPT-2-style transformer autoencoder with width 1024, 8 heads, and a 128-d latent per token. The autoencoder does not scale directly with the full target parameter dimensionality; for larger targets, we increase the token size according to the widest layer and use 6–7 transformer layers, a window size of 128, and batch size 16. We train with AdamW using a learning rate of 10^{-4} with a one-cycle schedule, weight decay 3×10^{-9}, and a contrastive-loss weight of 0.01–0.05 for 60–80 epochs. Permutation augmentation is not used. Target-specific settings are reported in Table 11.

Table 11: Representative SANE configurations used in our experiments. P denotes the number of generated scalars. All configurations use width 1024, 8 heads, and a 128-d latent per token.

Target	P	Token size / layers	Generator	Batch	Epochs
CIFAR-10 / ViT-mini	0.268M	769 / 6	≈153M	16	80
Oxford-Pet / ViT-mini	1.245M	1152 / 6	≈154M	16	80
VOC2012 / SegFormer-B2	27.36M	4097 / 7	≈185M	16	60

HyperTransformer. We re-implement HyperTransformer in PyTorch following the original paper, using a transformer width of 64, 2 attention heads, and 3 layers. The support set contains 5 images per class for classification and 10 images for segmentation. We train with AdamW using a learning rate of 10^{-3}, weight decay 10^{-4}, and gradient clipping at 1.0, for 20K steps on classification and 10K steps on segmentation. Normalization layers in segmentation targets are not generated and are learned jointly with the generator. We evaluate HyperTransformer only on compatible small targets; larger or structurally more complex targets are marked as not scalable. Representative settings are reported in Table 12.

DnD. We follow the released DnD implementation, including its conditioning, LoRA tokenization, and cascaded hyper-convolutional decoder. Each condition contains 128 unlabeled task prompts encoded by a frozen MiniLM sentence encoder, and rank-16 LoRA parameters are partitioned into 16×256 weight tokens. We train with AdamW using a learning rate of 3×10^{-4} with cosine decay, weight decay 0.01, gradient clipping at 1.0, and batch size 4. Exact target-specific configurations are reported in Table 13.

Table 12: Representative HyperTransformer configurations used in our experiments. P denotes the number of target parameters. Both configurations use a transformer with width 64, 2 attention heads, and 3 layers.

Target	P	Support set	Generator	Query batch	Steps
CIFAR-10 / CNN-mini	0.003M	10×5	0.46M	128	20K
Oxford-Pet / UNet	0.23M	10 images	0.91M	4	10K

Table 13: Representative DnD configurations used in our experiments. P denotes the number of generated scalars. Both targets use rank-16 LoRA adapters. Generator size excludes the frozen sentence encoder.

Target	P	Tokens / decoder	Generator	Batch	Steps
Qwen3-1.7B-Base / LoRA	17.43M	$4256 \times (16 \times 256)$ / 1024-d, conv $[256, 256, 384, 384]$	1.12B	4	18K
Llama-3.2-3B / LoRA	24.31M	$5936 \times (16 \times 256)$ / 1024-d, conv $[256, 256, 384, 384]$	1.56B	4	14K

T2L. We use the official T2L implementation with its supervised fine-tuning recipe and the medium (M) hypernetwork variant, with a residual MLP trunk of hidden size 512 and depth 4, 64-dimensional layer and module embeddings, and per-module output heads with a 256-dimensional bottleneck. Task descriptions are embedded with a frozen gte-large-en-v1.5 encoder. We generate rank-16 LoRA adapters for the target linear layers to match the target LoRA pool. We train with AdamW using a learning rate of 10^{-4}, 200 warmup steps followed by cosine decay, no weight decay, gradient clipping at 1.0, and batch size 8, for 25K steps. Representative target-specific settings are reported in Table 14.

Table 14: Representative T2L configurations used in our experiments. P denotes the number of generated scalars. Both targets use rank-16 LoRA adapters on all linear layers. Generator size excludes the frozen base model and description encoder.

Target	P	Hypernetwork	Generator	Batch	Steps
Qwen3-1.7B / LoRA	17.43M	M variant: hidden 512, depth 4, bottleneck 256	164M	8	25K
Llama-3.2-3B / LoRA	24.31M	M variant: hidden 512, depth 4, bottleneck 256	227M	8	25K

LPG. We use the LPG token VAE with an approximately 225M-parameter generator budget for large targets. Each checkpoint is standardized and partitioned layer-wise into tokens of width 2048 with structural embeddings for layer index, within-layer position, and parameter type. The VAE uses a four-stage 1D-convolutional encoder–decoder with channels $[64, 128, 256, 256]$ and kernel size 7. The latent width is adjusted according to target dimensionality to control the resulting generator size. We train with AdamW using a learning rate of 2×10^{-5}, 10 warmup epochs followed by cosine decay, no weight decay, gradient clipping at 1.0, KL weight 0.05, and batch size 2 for 200 epochs. In the second stage, prompts and optional data samples are encoded by a frozen Qwen3-VL-8B-Instruct, and a two-layer MLP projector maps the embedding to the VAE latent. Target-specific settings are reported in Table 15.

A.5 Evaluation Protocol

Common evaluation setup. For each architecture–dataset pair, all methods generate parameters for the same target model and are evaluated on the same downstream test set. We evaluate the generated parameters directly using the task-specific metric: accuracy for image and text classification, mIoU and Dice for segmentation, Pass@1 for mathematical reasoning and code generation, accuracy for multimodal understanding, and mean episodic return for reinforcement learning. The corresponding fully trained target model is reported as *Trained Model* to indicate the performance of the parameter pool from which the generator is learned.

Conditioning and data splits. Each method retains the conditioning interface defined by its original design. Prompt-conditioned methods are trained using the training prompt split and evaluated on

Table 15: Representative LPG configurations used in our experiments. P denotes the number of generated scalars. Both configurations use token width 2048 and encoder channels $[64, 128, 256, 256]$.

Target	P	Tokens / latent	Generator	Batch	Epochs
TinyImageNet / ResNet-18	11.27M	5,584 / 1280	236M	2	200
VOC2012 / SegFormer-B2	27.36M	13,570 / 512	227M	2	200

held-out prompt descriptions for the same architecture–dataset pair. Data-conditioned methods obtain their conditioning examples from the corresponding task data according to their original interface, while unconditional methods receive no conditioning input. Thus, methods share the same target models, checkpoint pools, and downstream evaluation sets, while differing only in the form of conditioning required by their design. Prompt splits are performed independently of the underlying task-data splits.

Repeated evaluation. Each generator is trained with five random seeds. For stochastic generators, we sample 10 generated parameter sets from each trained generator and report the mean and standard deviation over the resulting 50 evaluations. For deterministic generators, results are aggregated across the five independently trained seeds. All generated models are evaluated with the same downstream evaluation pipeline and decoding settings within each target task. Method-specific sampling schedules and generator training configurations are fixed across repeated evaluations and are detailed in Appendix A.4.

Model selection and reporting. Generator checkpoints are selected without using the downstream test set. When a method defines a validation criterion, we use its corresponding validation objective; otherwise, we use the fixed training budget specified for that method. The reported metric is always the downstream performance of the generated parameters rather than reconstruction error in parameter space. This separates the training objective of a generator from the functional metric used to compare methods.

Unsupported settings. We use **n.s.** (not scalable) when a method cannot be extended to a target through configuration changes alone and instead requires manual redesign of its generation architecture or target-specific generation modules. We use ✗ (structurally incompatible) when a method cannot generate the required parameter type by design, such as a LoRA-only generator applied to a full-parameter target. These labels are determined by the method interface and implementation requirements, rather than by poor downstream performance.

B Additional Experiments

B.1 Generalizability Study

B.1.1 Dataset Preparation

We construct three settings to evaluate cross-task generalization, each varying in how densely the conditioning space is covered during training.

CIFAR-10 Binary Pairs. Following TINA Li et al. (2025), we form 45 binary classification tasks from all $\binom{10}{2}$ class pairs in CIFAR-10. Target networks are trained from scratch with 50 checkpoints per task across 5 random seeds. We hold out 3 unseen pairs ({bird, deer}, {bird, horse}, {dog, horse}) and use the remaining 42 pairs as seen tasks. Each class label appears in only nine training tasks, providing sparse conditioning supervision.

Mini-ImageNet 10-way. We use stratified sampling to select 1050 10-way subsets from Mini-ImageNet, ensuring each class appears the same number of times across sampled tasks. 1000 subsets are used as seen and 50 are held out as unseen. Target networks are trained from scratch following the same protocol as Binary Pairs.

Binary Subset CIFAR-10. We follow the protocol introduced in RPG Wang et al. (2025b) and reuse its released checkpoints. The generation target is the classification head of a ViT-Tiny Dosovitskiy et al. (2021) attached to a frozen ImageNet-pretrained backbone. Each task corresponds to a binary classification defined by which CIFAR-10 classes form the positive class, e.g., {cat, dog} versus the rest, or {airplane, ship, truck, automobile} versus the rest. Excluding the all-zero and all-one cases, this yields 1022 distinct tasks, of which 1002 are seen and 20 are held out as unseen. Each CIFAR-10 class appears in roughly half of the 1002 training tasks, providing dense conditioning supervision.

Table 16: **Generalization to unseen tasks on CIFAR-10, Mini-ImageNet, and Binary Embedding CIFAR-10.** Seen reports the average accuracy over all seen tasks (%)↑. Unseen reports the accuracy on held-out tasks, averaged across $K = 10$ sampled weights over 3 random seeds. Mini-ImageNet Comb-1/2/3 denote randomly selected unseen 10-way combinations. **Bold** indicates the best result, and underline indicates the second best.

Method	CIFAR-10 Binary Pairs					Mini-ImageNet 10-way					Binary Subset CIFAR-10	
	Seen	Unseen				Seen	Unseen				Seen	Unseen
	Avg	bird/deer	bird/horse	dog/horse	Avg	Avg	Comb-1	Comb-2	Comb-3	Avg	Avg	Avg
RPG	51.45± 1.11	50.81± 2.59	50.37± 2.83	50.08± 1.91	50.42± 2.49	9.99± 1.15	10.06± 1.03	10.10± 1.07	9.47± 1.11	10.05± 1.38	95.79± 1.28	95.24± 1.28
D2NWG	**59.38**± 7.91	**52.06**± 4.13	**54.33**± 11.7	50.51± 9.55	**52.30**± 1.92	11.00± 1.30	8.24± 2.51	10.36± 2.24	10.78± 3.26	9.80± 1.11	48.49± 4.76	50.40± 11.65
TINA	50.32± 1.76	50.11± 3.39	48.04± 1.03	**50.96**± 3.36	49.70± 1.22	**35.62**± 3.35	**15.67**± 3.49	**19.96**± 3.45	**16.48**± 2.74	**20.35**± 4.48	94.27± 1.77	92.46± 2.71
NeuroGen	50.00± 0.00	50.00± 0.00	50.00± 0.00	50.00± 0.00	50.00± 0.00	10.00± 0.00	10.00± 0.00	10.00± 0.00	10.00± 0.00	10.00± 0.00	50.07± 15.23	49.95± 14.64
LPG	54.62± 7.78	50.08± 0.11	49.92± 0.09	50.05± 0.13	50.02± 0.10	9.81± 1.85	10.08± 0.04	10.13± 0.05	9.32± 0.12	9.98± 0.88	**97.51**± 1.12	**96.90**± 1.12

B.1.2 Analysis

The three settings differ primarily in conditioning supervision density. On Binary Subset CIFAR-10, each class appears in roughly half of the 1002 training tasks, so the generator can learn class-level semantics by interpolating across many label combinations. RPG Wang et al. (2025b) reaches 95.79% seen and 95.24% unseen, matching the oracle reported in its original paper, and TINA Li et al. (2025) exceeds 90%, showing that strong generalization is achievable under this setting. On Binary Pairs and Mini-ImageNet 10-way, most methods perform near chance level. The frozen backbone on Subsets absorbs much of the inductive bias and reduces the generation target to a lightweight classification head, while Pairs and 10-way require generating an entire CNN from scratch with sparse supervision (each class appears in only nine training tasks on Pairs). TINA is the only baseline that produces clearly above-chance accuracy on Mini-ImageNet 10-way, reaching 35.62% seen and 20.35% unseen against a 10% chance level, indicating partial task semantics transfer through CLIP conditioning. NeuroGen Wang et al. (2025a) produces exactly 10% with zero variance, indicating that its generated network degenerates to a constant predictor. `LPG` reaches 97.51% seen and 96.90% unseen on Binary Subsets, the highest among evaluated methods, but performs at chance level on CIFAR-10 Binary Pairs and 10-way Mini-ImageNet unseen tasks, consistent with the broader trend across baselines. *The generalization experiment shows that reliable cross-task generalization remains difficult once dense conditioning supervision and lightweight generation targets are removed, highlighting cross-task generalization as a core open challenge for the field.*

B.2 Ablation Study

Summary. Table 17 reports the experiment results of ablation study. Tokenization is the most critical component, causing collapse on CIFAR-10 MobileNetV3 (93.24% to 10.00%) and segmentation (59.85% to 9.53%) when removed. BN reparameterization is equally important for architectures with BatchNorm, dropping CIFAR-10 MobileNetV3 to 10.55%, but has no effect on models without BatchNorm such as Qwen-VL and BERT. Masked reconstruction matters most for full weight generation with heterogeneous layers, where padding tokens are prevalent, but has negligible effect on LoRA adaptation where token lengths are uniform. Each component addresses a different structural challenge in parameter generation.

B.3 Prompt Conditioning Analysis

We further evaluate whether prompt-conditioned generators respond to the semantic content of the conditioning prompt rather than merely reproducing a fixed architecture–dataset association. We train

Table 17: **Ablation study across four settings.** We ablate the main design components of TokenVAE on four representative settings: image classification (CIFAR-10 with MobileNetV3), image segmentation (MobileNetV3-Seg), multimodal LoRA adaptation (Qwen-VL), and text classification (SST-2 with BertMini). Tokenization denotes per-layer tokenization with structured positional encoding; masked reconstruction computes the MSE only over real weight elements, excluding padding; BN reparameterization applies the log-pivot transform to BatchNorm statistics. BN reparameterization is only applicable to architectures with BatchNorm running statistics, and is marked "–" otherwise. The final row is the full TokenVAE.

Component			CIFAR-10 + MobileNetV3		MobileNetV3-Seg		Qwen-VL LoRA		SST-2 + BertMini	
Tokenization	Masked	BN	Recon.↓	Acc.↑	Recon.↓	mIoU.↑	Recon.↓	Acc.↑	Recon.↓	Acc.↑
×	✓	✓	0.5378	10.00	NaN	9.53	0.5332	67.14	0.00005	84.93
✓	×	✓	0.0003	51.84	0.0001	9.53	0.5273	82.16	0.00004	85.07
✓	✓	×	0.0004	10.55	0.0001	57.61	0.5273	81.97	0.0004	85.14
✓	✓	✓	0.0004	93.24	0.0001	59.85	0.5273	81.97	0.00004	85.14

a single generator over four tasks with matched target parameter dimensionality: CNN on CIFAR-10, CNN on SVHN, MLP on MNIST, and MLP on Fashion-MNIST. At inference, we generate parameters using the prompt for each task and evaluate the resulting model on all four downstream tasks.

Table 18 reports the results for TINA and LPG. For both methods, the generated model achieves its highest performance on the task specified by the conditioning prompt in all four cases. Replacing the correct prompt with a mismatched task prompt substantially degrades performance, while random text fails to specialize the generated parameters to any of the known tasks. These results indicate that the conditioning signal affects which task-specific parameters are generated rather than acting only as an identifier for a fixed checkpoint pool.

Table 18: Prompt-conditioning analysis. Each row specifies the task used to condition parameter generation, while columns report downstream evaluation accuracy (%). Bold denotes the corresponding prompted task.

Method	Generation prompt	CIFAR-10	SVHN	MNIST	Fashion
TINA	CNN / CIFAR-10	**37.6**	12.0	6.2	12.1
	CNN / SVHN	8.3	**73.0**	7.9	7.9
	MLP / MNIST	8.9	6.2	**95.5**	6.3
	MLP / Fashion-MNIST	10.0	6.7	11.2	**80.3**
LPG	CNN / CIFAR-10	**46.4**	13.2	5.8	11.1
	CNN / SVHN	7.8	**73.8**	10.5	12.8
	MLP / MNIST	10.0	15.9	**96.8**	8.8
	MLP / Fashion-MNIST	10.0	6.7	10.5	**83.0**

We additionally summarize the effect of prompt perturbation in Table 19. *Correct* uses the prompt corresponding to the evaluation task, *Mismatch* uses a prompt from another known task, and *Random text* replaces the task description with unrelated text.

Table 19: Average accuracy (%) under different conditioning inputs across the four tasks.

Method	Correct	Mismatch	Random text
TINA	71.95	9.15	20.93
LPG	74.79	11.44	26.55

Table 20: **Generated LoRA adapters against few-shot in-context learning.** Base + ICL prompts the base model with few-shot demonstrations and no adapter. Best generation reports the best method in Table 4 for each target. Trained LoRA is the adapter obtained by fine-tuning.

Method	GSM8K		HumanEval		ScienceQA	MMMU
	Llama	Qwen3	Llama	Qwen3	Qwen3VL	Qwen3VL
Base + few-shot ICL	4.80	69.60	9.09	36.36	70.96	30.00
Best generation	57.83	78.27	35.13	62.93	91.65	42.13
Trained LoRA	59.07	77.98	32.32	51.22	92.23	41.78

Table 21: **Full parameter generation performance on saturated benchmarks: MNIST and AG-NEWS.** Top-1 accuracy (%, ↑). MNIST and AG-NEWS represent easier image and text classification benchmarks where most generation methods approach the base model performance, providing limited discriminative signal. Detailed analysis on the harder benchmarks is provided in the main text. **n/a** marks settings that require substantial per-architecture engineering. CNN, ResNet, ViT, and MNV3 in MNIST refer to their mini variants; Transformer and BERT in AG-NEWS refer to their mini variants. Trainable parameters of text models exclude embedding layers. **Bold** and underline mark the best and second-best results among generation methods.

Method	MNIST (Image Classification)				AG-NEWS (Text Classification)			
	CNN	ResNet	ViT	MNV3	MLP	RNN	Transformer	BERT
Trainable Params (M)	0.003	0.172	0.268	4.215	0.009	0.009	0.794	3.226
Base Model	94.83	99.49	98.75	96.39	91.21	89.55	92.72	92.80
DIFFUSION-BASED								
P-Diff	94.57	94.75	94.04	9.80	87.60	86.61	86.67	25.00
RPG	**94.80**	98.22	97.96	83.79	88.31	88.95	**91.82**	89.18
D2NWG	39.66	32.40	34.95	9.74	69.86	77.30	66.48	61.66
TINA	94.20	**99.34**	**98.15**	10.29	90.27	87.45	91.40	44.78
HYPERNET-BASED								
SANE	93.21	68.71	95.86	18.27	**91.11**	88.77	90.67	90.96
HyperTransformer	93.69	**99.34**	**n/a**	**n/a**	✗	✗	✗	✗
LLM-CONDITIONED								
NeuroGen	12.98	11.35	26.02	12.21	84.81	78.24	19.83	25.19
LPG	94.41	98.92	97.37	**88.38**	89.11	**89.13**	91.74	**91.32**