STAGE: SCALING TASK-SPECIFIC LORA GENERATION FROM FEW-SHOT EXAMPLES

Anonymous authors
Paper under double-blind review

ABSTRACT

Parameter-efficient fine-tuning reduces the cost of adapting pretrained models, but still requires a separate optimization process for every new task. We study whether previously optimized task adapters can serve as supervision for generating a new adapter from few-shot examples, and how generation changes as this supervision grows. We introduce STAGE, a conditional Flow Matching framework that generates LoRA adapters directly from support examples without gradient-based adapter fitting at deployment. For each training task, we pair a small labeled support set with an independently optimized adapter. We represent the adapter through SVD-based factors of its effective weight update to reduce ambiguity in the original LoRA factors. For image classification, we construct up to 100,000 fitted adapters per dataset-backbone pair, and compare bank sizes of 1K, 10K, and 100K. At a fixed generation budget, support-selected accuracy increases with bank size in all six dataset-backbone settings. With only a few support images, generated adapters improve official-split accuracy over the frozen classifier by up to approximately 13.4 and 9.9 percentage points on CIFAR-100 and Tiny-ImageNet, respectively. We also evaluate a text generator on 14 language-generation datasets, with accuracy gains of 31.7 and 21.9 percentage points on e-SNLI and WikiSQL, respectively. These results show that scaling task–adapter supervision can improve few-shot adaptation without gradient-based adapter fitting at deployment.

1 INTRODUCTION

Pretrained vision and language models can be adapted to many downstream tasks, but each new task often requires additional optimization. Parameter-efficient fine-tuning (PEFT) reduces the number of updated parameters. Low-Rank Adaptation (LoRA) (Hu et al., 2022), for example, represents task-specific updates through low-rank factors while keeping the pretrained model frozen. Nevertheless, a new task still requires fitting a new adapter. When many related tasks are encountered, these separate optimizations also produce a collection of fitted adapters. Each adapter records how the model was changed for a particular task. In this work, we study whether this collection can be used to learn how to adapt from a small set of examples.

Recent work supports this perspective by treating neural-network parameters as a learnable data modality (Han et al., 2026). Generative models produce network weights and parameter subsets (Peebles et al., 2022; Wang et al., 2024; Schürholt et al., 2024; Soro et al., 2025; Wang et al., 2025; Gupta et al., 2026), while conditional methods generate adapters from task descriptions, reference images, or examples (Phang et al., 2023; Jin et al., 2024; Wu et al., 2024; Shao et al., 2025; Charakorn et al., 2025; Liang et al., 2025; Huang et al., 2026). FLoWN (Saragih et al., 2025) further demonstrates support-conditioned classifier generation through flow matching. We ask a more specific question for task-specific LoRA generation:

How does the number of independently fitted task-adapter pairs affect the quality of adapters generated from few-shot examples?

Studying this question requires a large collection of independently fitted task adapters, a consistent representation of their updates, and a way to select generated candidates without query labels. Different LoRA factor pairs can represent the same effective update, making the raw factors ambiguous training targets (Huang et al., 2026). Moreover, selecting the candidate with the highest query

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.

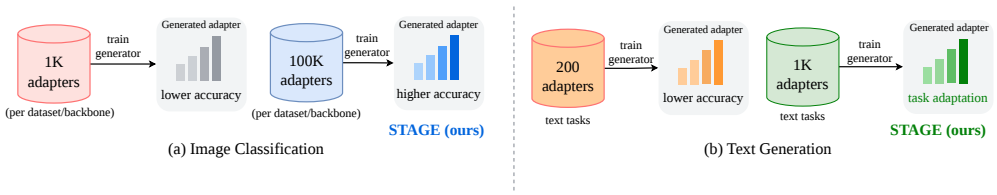

Figure 1: **Scaling task-specific adapter banks. (a) Image classification:** increasing the task-specific LoRA bank from 1K to 100K adapters per dataset–backbone pair improves the accuracy of generated adapters. **(b) Text generation:** scaling the task-specific adapter bank from 200 to 1K adapters per dataset similarly improves downstream task adaptation.

accuracy does not measure the adaptation available at deployment. To address these challenges, we introduce **S**TAGE: **S**caling **T**ask **A**dapter **G**eneration with Few-Shot **E**xamples. Our contributions are summarized as follows:

- **Few-shot conditional adapter generation.** For each sampled task, we fit a target LoRA offline and pair it with a labeled support set. We reconstruct the target's effective update and obtain SVD-based factors, so that equivalent LoRA factorizations provide a more consistent training target. STAGE learns a support-conditioned distribution over these targets using Flow Matching. For a new task, it generates candidate adapters and selects one using only the support examples, without task-specific gradient optimization.
- **Scaling task–adapter supervision.** For image classification, we construct 100,000 fitted adapters per dataset–backbone pair and compare banks of 1K, 10K, and 100K. Each comparison uses 16 generated candidates and 96 flow steps, keeping the inference budget fixed. Support-selected accuracy increases at each scale across three backbones on CIFAR-100 (Krizhevsky, 2009) and Tiny-ImageNet (Le & Yang, 2015). The improvement therefore persists when the final adapter is selected without query labels. With one support image per class, the largest official-split gains over the frozen classifier are approximately 13.4 and 9.9 percentage points, respectively.
- **Cross-domain and functional evaluation.** We evaluate a separate text generator on 14 datasets using 10 support examples per task. Support-selected generation improves over the frozen Qwen3-1.7B baseline on all 14 datasets, with accuracy gains of 31.68, 21.91, and 13.46 percentage points on e-SNLI (Camburu et al., 2018), WikiSQL (Zhong et al., 2017), and KodCode-Contest (Xu et al., 2025), respectively. A bank-size study on five commonsense datasets further shows that the effect of additional targets depends on the task. For both vision and text, we report support-selected results alongside query-label Best-of-N, separating deployable selection from candidate quality. We also compare generated and fitted effective updates and find that useful adaptation need not require matching a particular optimized target.

2 Related Work

Generative modeling of neural network weights. Recent work treats neural network parameters as a learnable data modality (Han et al., 2026). Hyper-representations (Schürholt et al., 2022) and SANE (Schürholt et al., 2024) learn weight representations that support model generation. G.pt (Peebles et al., 2022) generates parameter updates conditioned on checkpoints and target performance, while P-diff (Wang et al., 2024) and D2NWG (Soro et al., 2025) use latent diffusion, with D2NWG supporting dataset conditioning. HyperDiffusion (Erkoç et al., 2023) models implicit neural-field weights directly. RPG (Wang et al., 2025) combines recurrent parameter tokenization with diffusion for full-network and LoRA generation, whereas DeepWeightFlow (Gupta et al., 2026) applies flow matching to aligned and canonicalized network weights.

Task-conditioned and LoRA parameter generation. Hypernetworks generate network parameters from conditioning information (Ha et al., 2016). For visual few-shot learning, dynamic classifier generation (Gidaris & Komodakis, 2018), CNAPs (Requeima et al., 2019), and HyperTransformer (Zhmoginov et al., 2022) predict classifier or adaptation parameters from support examples. MetaDiff (Zhang et al., 2024) uses conditional diffusion for few-shot adaptation, while FLoWN (Saragih et al., 2025) uses conditional flow matching in a learned weight representation, including support-conditioned classifier-head generation.

For language models, HyperFormer (Karimi Mahabadi et al., 2021) generates task-specific adapters, and HyperTuning (Phang et al., 2023) generates prefixes or LoRA parameters from few-shot examples. HINT (Ivison et al., 2023) conditions parameter-efficient modules on instructions and examples, while HyperLoRA (Lv et al., 2024) studies constrained LoRA generation for cross-task generalization. Text-to-LoRA (Charakorn et al., 2025) generates adapters from task descriptions, and LoRA-Gen (Xiao et al., 2025) generates edge-model adapters using a larger cloud model. Generative Adapter (Chen et al., 2025), SHINE (Liu et al., 2026), and Doc-to-LoRA (Charakorn et al., 2026) map test-time context to adaptation parameters.

Other LoRA generators include COND P-DIFF (Jin et al., 2024), ICM-LoRA (Shao et al., 2025), and ORAL (Shahroz et al., 2025), using latent diffusion, variational autoencoding, and recurrent diffusion, respectively. DnD (Liang et al., 2025) generates LoRA updates from unlabeled task prompts through a cascaded generator. For visual personalization, HyperDreamBooth (Ruiz et al., 2024) predicts adaptation weights followed by fine-tuning, whereas DiffLoRA (Wu et al., 2024) and HyperLoRA (Li et al., 2025) generate image-conditioned LoRAs without per-subject optimization. LoRAGen (Huang et al., 2026) combines latent diffusion with effective-update supervision and a module-aware decoder.

STAGE studies how scaling task–adapter supervision improves few-shot conditional generation across vision and language. In vision, it generates classifier-head LoRAs from labeled one-shot support sets. We reconstruct each effective update $\Delta W = (\alpha/r)BA$, obtain deterministic SVD-based low-rank targets, and learn conditional rectified flow directly in this canonical adapter space without a weight autoencoder.

Weight-space structure and generalization. Parameter symmetries motivate weight alignment (Ainsworth et al., 2023), canonicalization (Gupta et al., 2026), and permutation-aware architectures (Zhou et al., 2023). Learning on LoRAs (Putterman et al., 2024) develops invariant and equivariant processing of low-rank factors, while LoRAGen (Huang et al., 2026) addresses factor ambiguity through effective-update supervision. These approaches motivate our use of canonical adapter targets and effective-update comparisons. High downstream accuracy alone does not establish weight-space novelty (Zeng et al., 2025). We therefore evaluate generalization to unseen class combinations and held-out classes alongside effective-update similarity.

3 Background and Motivation

LoRA changes a frozen weight matrix W_0 by adding a low-rank update, $\Delta W = (\alpha/r)BA$ (Hu et al., 2022), where $W_0 \in \mathbb{R}^{d\times d}$, $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times d}$, d is the hidden dimension of backbone model and $r < d$. The two smaller matrices A and B are optimized instead of W_0; r limits the rank of the update, and α sets its scale. Ordinary LoRA still requires a separate optimization run for each new task. We ask whether adapters fitted for earlier tasks can instead teach a generator to produce an update from a small labeled support set. During training, we pair each support set with an adapter fitted offline for the same task. The fitted adapter supplies the target update; the support examples tell the generator what task that update should solve. For a new task, the support set is the only task-specific information supplied to the generator. It must infer a useful weight update from a few labeled examples, without fitting a new adapter.

The fitted LoRA factors do not provide consistent target coordinates. For example, multiplying B by a nonzero number and dividing A by the same number changes both factors but leaves BA unchanged. More generally, for any invertible R, $BA = (BR)(R^{-1}A)$. Thus two adapters can make the same change to the model while having different factor values. We reconstruct their effective updates ΔW and obtain SVD-based factors with fixed ordering and sign choices as the generator targets (Huang et al., 2026). This reduces variation due to the choice of factorization and gives the generator a more consistent representation to learn. The representation of these factors depends on which layers are adapted. Section 4 give the construction.

We model the distribution of these targets using conditional Flow Matching (Lipman et al., 2023; Liu et al., 2023). During training, we sample Gaussian noise as a starting point and choose a point on the straight line toward a fitted target. Given that intermediate point, its position along the path, and the support examples, the model learns which direction leads toward the target. At evaluation, it starts from new Gaussian noise and follows the learned directions to produce candidate adapters. Because the support set is small, candidate quality must also be judged from those examples. We select a

candidate using only the labeled support set, without fitting an adapter with gradients or using query labels. The image and text generators share this training objective but use different representations and velocity networks for their adapted layers.

4 Proposed Method: STAGE

4.1 Overview

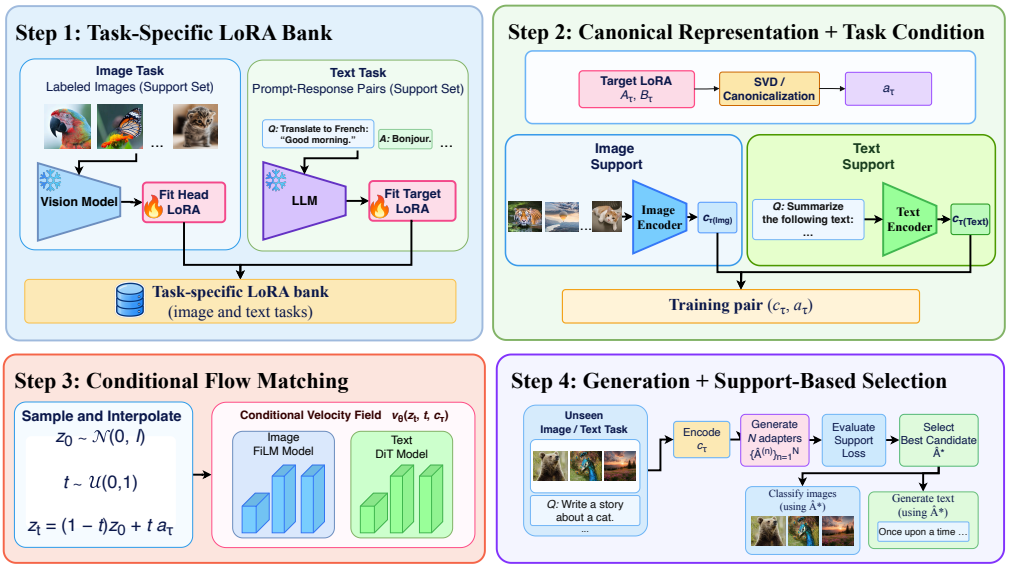

Figure 2: Overview of the four-step pipeline for STAGE.

STAGE has four steps: constructing a bank of fitted task adapters, representing their effective updates, training a support-conditioned flow generator, and generating an adapter for a new task (Figure 2). The first three stages are performed offline. We denote a task by τ, its labeled support set by $\mathcal{D}_\tau^s = \{(x_i, y_i)\}_{i=1}^{N_s}$, its adapter-fitting set by $\mathcal{D}_\tau^{\text{fit}}$, and its query set by $\mathcal{D}_\tau^q$. The target adapter is fitted on $\mathcal{D}_\tau^{\text{fit}}$; the support set conditions generation, and the query set measures performance. In the image adapter bank, these sets are disjoint. In text generation, $\mathcal{D}_\tau^s \subseteq \mathcal{D}_\tau^{\text{fit}}$, while the query set remains disjoint. The following classifier-head equations describe the image implementation. The text implementation is summarized in Table 7. Let $h_\psi(x)$ be the frozen backbone features and $f_0(x) = W_0 h_\psi(x) + b$ the frozen classifier. We adapt only its weight matrix through $\Delta W_\tau = (\alpha/r) B_\tau A_\tau$, giving $f_\tau(x) = (W_0 + \Delta W_\tau) h_\psi(x) + b$.

Step 1: Task-Specific LoRA Bank Construction. The conditional generator requires a large collection of task-dependent parameter solutions from which to learn. We therefore sample many tasks for each backbone–dataset configuration and optimize one classifier-head LoRA for every task. For task τ, the backbone, original classifier weight, and bias remain frozen, and only the LoRA factors are optimized on $\mathcal{D}_\tau^{\text{fit}}$:

$$(A_\tau^*, B_\tau^*) = \arg\min_{A,B} \frac{1}{|\mathcal{D}_\tau^{\text{fit}}|} \sum_{(x,y)\in\mathcal{D}_\tau^{\text{fit}}} \mathcal{L}_{\text{CE}} \left(\left[W_0 + \frac{\alpha}{r} BA \right] h_\psi(x) + b,\ y \right), \tag{1}$$

This produces the effective target update $\Delta W_\tau^* = (\alpha/r) B_\tau^* A_\tau^*$. This optimization is performed only during bank construction, not at deployment.

Step 2: Canonical Adapter Representation Raw LoRA factors are not unique: different pairs (A_τ, B_τ) can represent the same effective weight update. We therefore canonicalize the trained update ΔW_τ^* rather than using its original factors. First, we compute the rank-r SVD of the trained update, $\Delta W_\tau^* = U_\tau \Sigma_\tau V_\tau^\top$. We fix the sign of each retained singular-vector pair using the largest-magnitude

entry of its left singular vector as a pivot. Using the resulting sign-adjusted vectors, we construct $\bar{B}_\tau = \widetilde{U}_{\tau,r}\left(\frac{r}{\alpha}\Sigma_{\tau,r}\right)^{1/2}$, $\bar{A}_\tau = \left(\frac{r}{\alpha}\Sigma_{\tau,r}\right)^{1/2}\widetilde{V}_{\tau,r}^\top$. The subscript r denotes the retained components. These factors preserve the trained update: $(\alpha/r)\bar{B}_\tau\bar{A}_\tau = \Delta W_\tau^*$. We concatenate and flatten the factors into $u_\tau = \mathrm{vec}(\bar{A}_\tau, \bar{B}_\tau)$, then standardize them elementwise as $a_\tau = (u_\tau - \mu)/\sigma$. The mean μ and standard deviation σ are computed only from the generator-training split. The resulting a_τ is the generator's target representation. To describe the task, we combine each support example's frozen feature $h_\psi(x)$ with a learned label embedding. An example-level MLP projects these inputs into tokens, which a Transformer support encoder E_ϕ aggregates through a learned `[CLS]` token. Its output, $c_\tau = E_\phi(\mathcal{D}_\tau^s)$, is the task condition. Generator training uses the paired condition and target (c_τ, a_τ).

Step 3: Conditional LoRA Generator Training. Given a task–adapter pair (c_τ, a_τ), STAGE learns a conditional rectified-flow model in the normalized canonical adapter space. We sample $z_0 \sim \mathcal{N}(0, I)$ and $t \sim \mathcal{U}(0, 1)$ and construct the linear path

$$z_t = (1-t)z_0 + ta_\tau, \qquad \mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{\tau,z_0,t}\left[\left\|v_\theta(z_t, t, c_\tau) - (a_\tau - z_0)\right\|_2^2\right]. \tag{2}$$

The corresponding target velocity is $a_\tau - z_0$. The support encoder E_ϕ and conditional velocity field v_θ are trained jointly. The implemented velocity model is a FiLM-conditioned residual MLP. The adapter state z_t is projected into a hidden representation, while the support condition c_τ and a sinusoidal embedding of the flow time t are projected jointly to provide the conditioning signal. Each residual block uses this signal to apply learned feature-wise scale and shift modulation, and the final layer predicts a velocity vector with the same dimensionality as the adapter representation. The generator therefore learns directly in canonical LoRA parameter space, without a weight autoencoder or latent decoder. Query examples are not used in the Flow Matching objective.

Step 4: Adapter Generation and Support-Based Selection. For a previously unseen task τ, we first encode its labeled support set to obtain $c_\tau = E_\phi(\mathcal{D}_\tau^s)$. No task-specific adapter is optimized with gradients. We draw N independent initial states $z_0^{(n)} \sim \mathcal{N}(0, I)$ and integrate the learned velocity field using Euler updates:

$$z_{\ell+1}^{(n)} = z_\ell^{(n)} + \frac{1}{L}v_\theta\left(z_\ell^{(n)}, \frac{\ell}{L}, c_\tau\right), \qquad \ell = 0, \ldots, L-1, \tag{3}$$

where L is the number of integration steps and $n \in \{1, \ldots, N\}$ indexes the candidates. Each terminal state is denormalized as $\hat{u}_\tau^{(n)} = \sigma \odot z_L^{(n)} + \mu$ and reshaped into LoRA factors $(\hat{A}_\tau^{(n)}, \hat{B}_\tau^{(n)})$. The resulting candidate update is $\widehat{\Delta W}_\tau^{(n)} = (\alpha/r)\hat{B}_\tau^{(n)}\hat{A}_\tau^{(n)}$. We evaluate each candidate using only the labeled support set. Its mean support cross-entropy is

$$\mathcal{L}_{\mathrm{sup}}^{(n)} = \frac{1}{|\mathcal{D}_\tau^s|}\sum_{(x,y)\in\mathcal{D}_\tau^s}\mathcal{L}_{\mathrm{CE}}\left([W_0 + \widehat{\Delta W}_\tau^{(n)}]h_\psi(x) + b,\, y\right).$$

We then select $n^* = \arg\min_{n\in\{1,\ldots,N\}}\mathcal{L}_{\mathrm{sup}}^{(n)}$. The selected update is attached to the frozen classifier. In the reported experiments, we generate $N = 16$ candidates using $L = 96$ Euler steps. Generation and support-based selection require no gradient-based adapter optimization at deployment. For analysis, we also report *Best-of-16*, which selects the candidate with the highest query accuracy. Because it uses query labels, this oracle selection is unavailable during deployment.

5 Experiments

We evaluate STAGE in two domains with substantially different parameter geometries and downstream tasks: episodic image classification and conditional text generation. We train the domain-specific generators separately, but both follow the same central principle: a conditional flow model generates a canonical LoRA representation conditioned on a small task-specific support set, avoiding gradient-based adapter fitting at inference. Our experiments address three questions: (1) whether generated adapters improve the corresponding frozen base model; (2) how they compare with adapted prior generation methods and gradient-based fine-tuning baselines; and (3) how adaptation performance changes as the task–adapter supervision bank grows.

5.1 IMAGE-CLASSIFICATION RESULTS

Table 1 shows that STAGE improves support-selected accuracy over the frozen baseline across all three backbones and both datasets under official evaluation. Under the official protocol, CIFAR-100 accuracy reaches **95.08%** for ViT-S/16, **94.28%** for EfficientNet-V2-S, and **93.43%** for ConvNeXt-V2 Tiny, yielding gains of 13.44, 7.37, and 12.38 percentage points, respectively. On Tiny-ImageNet, the corresponding accuracies are 90.04%, 82.21%, and 85.15%, with gains of 9.92, 2.69, and 3.98 points. Generated-mean accuracy also exceeds the frozen baseline in every setting. Support-based selection provides its largest improvement for Tiny-ImageNet ViT-S/16, raising official accuracy from **83.61%** to **90.04%**, although the query-based Best-of-16 oracle remains higher throughout. Support-selected adapters outperform the gradient reference in 10 of the 12 evaluations; Tiny-ImageNet ConvNeXt-V2 Tiny is the exception under both protocols. Under random evaluation, directly optimized target LoRAs achieve higher accuracy than support-selected adapters, but use 10 fitting images per class. STAGE instead generates adapters from one support image per class without task-specific gradient updates, demonstrating effective adaptation from limited task information. Detailed generated-adapter results are provided in Appendix D.4, Table 16.

Table 1: **Image-classification evaluation.** Mean 10-way query accuracy (%) over 100 tasks per dataset using the official CIFAR-100 test and Tiny-ImageNet validation splits. Support and query images are disjoint. Target LoRA is fitted on the same one-image-per-class support set used by the generator. Gain is support-selected minus Frozen Base in percentage points, computed before rounding. Best-of-N is a query-based oracle with $N = 16$. Row-best accuracies and gains above 10 points are bolded and highlighted in **blue** and support-selected are highlighted in **green**.

Domain	Dataset	Model	Frozen Base	Generated mean	Support-selected	Gain	Best-of-N (oracle)	Target LoRA	Full SFT
Image classification	CIFAR-100 (100 classes)	ViT-S/16	81.64	93.68	**95.08**	**+13.43**	**96.52**	95.56	91.51
		EfficientNet-V2-S	86.91	94.25	**94.28**	+7.37	**95.64**	93.78	90.34
		ConvNeXt-V2 Tiny	81.05	91.97	**93.43**	**+12.38**	**95.87**	92.96	84.00
	Tiny-ImageNet (200 classes)	ViT-S/16	80.12	83.61	**90.04**	+9.92	91.68	**94.73**	87.58
		EfficientNet-V2-S	79.52	82.17	**82.21**	+2.69	83.19	**84.02**	79.52
		ConvNeXt-V2 Tiny	81.17	85.02	**85.15**	+3.98	85.58	**88.15**	87.73

5.2 TEXT-GENERATION RESULTS

Table 4 reports results over 200 tasks per dataset spanning commonsense reasoning, mathematics, SQL, and Python generation. Conditioned on only 10 support examples, STAGE improves over the base model on all 14 datasets. The largest gains are 31.68 percentage points on e-SNLI (Camburu et al., 2018), 21.91 on WikiSQL (Zhong et al., 2017), and 13.46 on KodCode-Contest (Xu et al., 2025), while CommonGen (Lin et al., 2020) improves by 8.87 BLEU-4 (Papineni et al., 2002) points. Mathematical reasoning also improves, by 0.39 points on GSM8K (Cobbe et al., 2021), 3.44 on MATH (Hendrycks et al., 2021), and 1.17 on MetaMathQA (Yu et al., 2024). Support-selected performance exceeds the generated mean on every dataset and outperforms the directly fitted target LoRA on 12 of 14 datasets, the exceptions being CommonGen and MATH. This comparison favors the targets, which are optimized on each task's 100-example fitting set, including the 10 support examples; generated adapters can nevertheless outperform them, although not universally. Best-of-N remains above support-selected performance throughout, indicating room to improve selection without query labels. Full SFT achieves higher scores on all datasets, but it uses three epochs of full-parameter tuning on each dataset's training pool and therefore relies on substantially broader supervision.

5.3 SCALING LORA BANKS FOR TASK ADAPTATION

We evaluate how increasing the number of task-specific LoRA adapters used to train the conditional generator affects adaptation performance. Figure 3 compares banks of 1,000, 10,000, and 100,000 adapters on CIFAR-100 and Tiny-ImageNet under the official evaluation protocol. We study scaling on five commonsense datasets using 200, 1,000, 5,000, 10,000, and 20,000 targets per dataset on Qwen3-1.7B (Yang et al., 2025). We report Best-of-16 oracle performance to assess candidate quality

Table 2: **Text-generation evaluation.** Query accuracy (%) over 200 tasks per dataset (CommonGen: BLEU-4) on **Qwen3-1.7B**. Support and query rows are disjoint. Target LoRA is fitted on each task's 100-example fitting set, which contains the support examples. Full SFT is full-parameter fine-tuning on the dataset's own training pool (best in 3 epochs), decoded on the same query rows. Gain is support-selected minus Frozen Base in percentage points, computed before rounding. Best-of-N is a query-label oracle with N=16 for all datasets. Row-best accuracies and gains above 10 points are bolded and highlighted in **blue**, while support-selected results are bolded and highlighted in **green**.

Domain	Dataset	Frozen Base	Generated mean	Support-selected	Gain	Best-of-N (oracle)	Target LoRA	Full SFT
Commonsense	ART	68.71	71.48	**71.61**	+2.90	74.17	69.28	**79.62**
	PIQA	70.05	74.98	**75.14**	+5.09	77.67	72.42	**81.10**
	ECQA	65.82	68.59	**68.67**	+2.85	71.35	65.55	**72.94**
	e-SNLI	56.16	87.82	**87.84**	**+31.68**	89.98	84.64	**91.44**
	CommonGen	15.04	22.96	**23.91**	+8.87	26.09	**30.88**	25.02
Math	GSM8K	76.47	76.57	**76.86**	+0.39	80.44	76.70	**80.64**
	MATH	62.17	65.15	**65.61**	+3.44	68.99	65.81	**71.21**
	MetaMathQA	79.17	80.20	**80.34**	+1.17	**84.00**	75.19	83.61
SQL	gretel	54.10	58.70	**59.02**	+4.92	62.49	55.33	**72.54**
	Spider	38.73	46.16	**46.84**	+8.11	52.01	44.61	**65.51**
	WikiSQL	53.25	74.91	**75.16**	**+21.91**	77.82	72.85	**90.07**
Python	KodCode-Filter	22.22	27.23	**27.34**	+5.12	30.36	24.52	**39.09**
	KodCode-Leetcode	16.31	16.94	**17.02**	+0.71	20.16	14.53	**35.56**
	KodCode-Contest	25.50	38.83	**38.96**	**+13.46**	42.03	34.37	**53.99**

independently of support-based selection, keeping the inference budget fixed at 16 candidates and 96 flow steps. For image classification, increasing the bank from 1,000 to 100,000 adapters improves accuracy significantly, 95% on CIFAR-100. 91.68% on Tiny-ImageNet with ViT-S/16 Other backbones show smaller gains beyond 10,000. Detailed results are provided in Appendix E.1, Table 18. For text tasks, the largest improvements occur between 200 and 1,000 targets per dataset, particularly on e-SNLI and CommonGen; further scaling yields comparatively small changes. Together, these results support scaling offline adapter banks to improve generation quality at a fixed inference budget, with the gains and their saturation depending on the dataset and backbone.

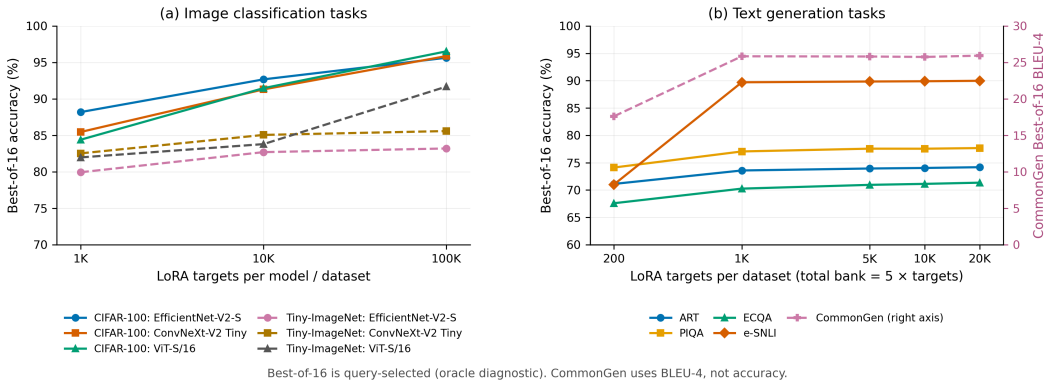

Figure 3: **Effect of Scaling LoRA banks on generated-adapter performance. Left:** Image classification tasks, evaluated on 100 official-split tasks. Counts denote training adapters per dataset–backbone pair. **Right:** 5 text datasets evaluated on 200 test tasks. Multiple-choice accuracy uses the left axis; CommonGen BLEU-4 (dashed) uses the right axis. Counts denote targets per dataset, with a total bank 5 times this size.

6 Comparison with Existing LoRA Generation Methods

Table 3: **Image-classification evaluation.** Mean 10-way query accuracy (%) on 100 official tasks per dataset–backbone pair, using disjoint one-shot support and query images. Prior methods are adaptations to the same classifier-head LoRA bank, not reproductions of their original benchmarks; RPG-LSTM is the LSTM ablation. Ours is support-selected from 16 samples. Gain is Ours minus Frozen Base (percentage points); Best-of-16 is a query-label oracle. Support LoRA uses SGD on the same support set, and Full SFT is the full-gradient reference evaluated on these tasks. Green marks Ours; blue marks a row-best result outside Ours.

			Adapted comparison methods				STAGE (Ours)				
Dataset	**Backbone**	**Frozen Base**	**p-diff**	**RPG LSTM**	**DeepWeight Flow**	**DnD**	**Support-selected**	**Gain**	**Best-of-N (oracle)**	**Target LoRA**	**Full SFT**
CIFAR-100	ViT-S/16	81.64	65.99	73.47	75.81	**95.84**	95.08	+13.43	96.52	95.47	91.51
	EfficientNet-V2-S	86.91	87.35	87.25	87.62	90.79	**94.28**	+7.37	95.64	93.95	90.34
	ConvNeXt-V2 Tiny	81.05	58.71	31.87	39.20	**95.11**	93.43	+12.38	95.87	94.70	83.99
Tiny-ImageNet	ViT-S/16	80.12	56.92	54.66	78.33	93.92	90.04	+9.92	91.68	**94.34**	87.58
	EfficientNet-V2-S	79.52	79.33	61.64	77.75	81.33	82.21	+2.69	83.19	**84.04**	79.52
	ConvNeXt-V2 Tiny	81.17	81.42	81.18	81.41	84.65	85.15	+3.98	85.58	**91.17**	87.73

Table 4: **Text-generation evaluation.** Query accuracy (%) on 200 tasks per dataset (BLEU-4 for CommonGen) using Qwen3-1.7B and disjoint support and query rows. DnD and LoRAGen use the same LoRA bank and 10-example support condition as Ours; LoRAGen is support-selected from 6 samples. Ours is support-selected from 16 samples. Gain is Ours minus Frozen Base (percentage points); Best-of-16 is a query-label oracle. Target LoRA is fitted on each task's 100 examples, and Full SFT is full-parameter fine-tuning on the dataset training pool. Green marks Ours; blue marks row-best results outside Ours and highlighted gains.

			Baselines		STAGE (Ours)					
Domain	**Dataset**	**Frozen Base**	**DnD**	**LoRAGen**	**Generated mean**	**Support-selected**	**Gain**	**Best-of-N (oracle)**	**Target LoRA**	**Full SFT**
Commonsense	ART	68.71	67.97	64.37	71.48	**71.61**	+2.90	74.17	69.28	**79.62**
	PIQA	70.05	73.43	70.67	74.98	**75.14**	+5.09	77.67	72.42	**81.10**
	ECQA	65.82	65.60	65.08	68.59	**68.67**	+2.85	71.35	65.55	**72.94**
	e-SNLI	56.16	55.14	82.54	87.82	**87.84**	**+31.68**	89.98	84.64	**91.44**
	CommonGen	15.04	15.11	16.97	22.96	**23.91**	+8.87	26.09	**30.88**	25.02
Math	GSM8K	76.47	77.08	26.62	76.57	**76.86**	+0.39	80.44	76.70	**80.64**
	MATH	62.17	62.25	11.58	65.15	**65.61**	+3.44	68.99	65.81	**71.21**
	MetaMathQA	79.17	86.56	48.40	80.20	**80.34**	+1.17	**84.00**	75.19	83.61
SQL	gretel	54.10	55.24	47.40	58.70	**59.02**	+4.92	62.49	55.33	**72.54**
	Spider	38.73	44.96	42.90	46.16	**46.84**	+8.11	52.01	44.61	**65.51**
	WikiSQL	53.25	58.22	47.80	74.91	**75.16**	**+21.91**	77.82	72.85	**90.07**
Python	KodCode-Filter	22.22	23.07	16.95	27.23	**27.34**	+5.12	30.36	24.52	**39.09**
	KodCode-Leetcode	16.31	17.75	8.46	16.94	**17.02**	+0.71	20.16	14.53	**35.56**
	KodCode-Contest	25.50	25.93	26.43	38.83	**38.96**	**+13.46**	42.03	34.37	**53.99**

6.1 Weight-Space and Functional Analysis

We analyze task sensitivity and alignment with fitted targets to understand the relationship between generated updates and adaptation performance. All comparisons use effective weight updates, $\Delta W = (\alpha/r)BA$, rather than individual LoRA factors. For task sensitivity, we generate two adapters independently for each sampled task and compare their pairwise cosine similarities. Figure 4 shows three CIFAR-100 tasks across three backbones. Updates generated for the same task are generally more similar than updates generated for different tasks, suggesting that independent samples retain structure associated with their conditioning task. Figure 5 presents the corresponding analysis for text generation. To examine whether alignment with a fitted target predicts adaptation quality, we use a separate diagnostic evaluation of 100 randomly sampled image-classification tasks per dataset–backbone pair. Complete paired statistics are provided in Appendix G.3. Figures 8 and 9.

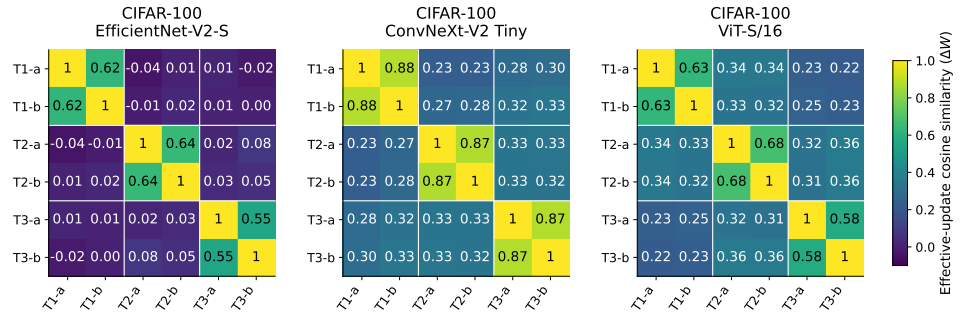

Figure 4: **Task sensitivity of generated image-classification adapters.** Pairwise cosine similarity between effective LoRA updates $\Delta W = (\alpha/r)BA$ for (T1–T3) 10-way CIFAR-100 tasks. Here, a and b denote independently generated adapters. The off-diagonal entries within each 2×2 diagonal block compare adapters for the same task; entries outside these blocks compare different tasks.

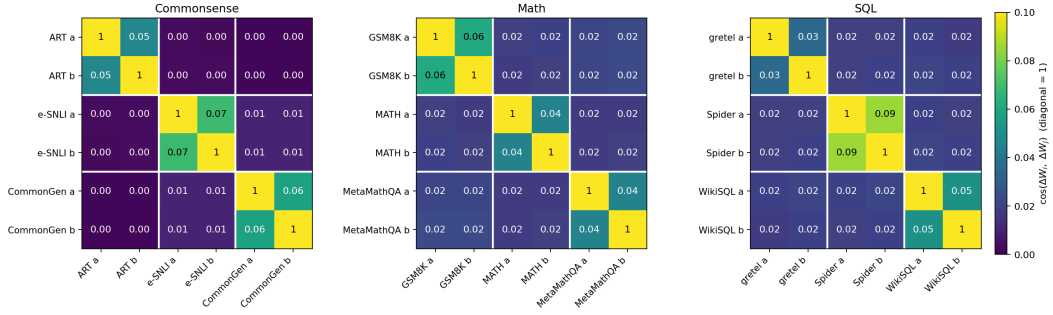

Figure 5: **Task sensitivity of generated text-generation adapters.** Pairwise cosine similarity of effective LoRA updates $\Delta W = (\alpha/r)BA$ generated by STAGE. Within-task and cross-task comparisons characterize how the generated updates depend on the conditioning task.

STAGE eliminates deployment-time gradient updates, although its default 96-step generation is not universally cheaper than adapter fitting. Reducing sampling to four steps cuts flow-sampling FLOPs by $24\times$ with at most a 0.44-point accuracy drop; full cost details are in Appendix F.

7 Discussion and Conclusion

We introduced STAGE, a framework that learns to generate task-specific LoRA adapters from a small set of support examples. By combining SVD-based canonicalization with a conditional flow model, STAGE enables few-shot adaptation to new tasks without gradient-based adapter fitting at deployment. Experiments on image classification and text generation show substantial improvements over frozen models, and in several settings the generated adapters also outperform directly fitted LoRA targets. Scaling task–adapter banks from 1K to 100K improves support-selected performance across all six dataset–backbone settings, whereas the gains from scaling the text adapter bank are more task-dependent. These findings suggest that fitted adapters can serve as reusable training data. Larger banks improve generation quality at a fixed inference budget, although the benefits vary with the task and backbone. Our results further indicates that useful adaptation does not need to reproduce the fitted target, highlighting the importance of evaluating generated adapters by their downstream performance The remaining gap between support-selected and oracle performance motivates better selection from few examples. Future work could extend adaptation to deeper vision layers, reduce the costs of bank construction and sampling, and evaluate transfer beyond the training task distribution. Overall, our results point to scaling task–adapter supervision as a promising direction for learning adaptation directly in weight space.

AI USE STATEMENT

In this work, we used generative AI tools for drafting the paper in LaTex, assists in coding to write simple helper functions. We have not used generative AI tools for writing the methods, proposing or refining research hypotheses, creating or modifying datasets, designing experiments, interpreting experimental results, or writing substantive sections of the paper and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for assisting with coding and debugging of helper scripts, preparing LaTeX drafts, formatting tables and, refining figures. We have reviewed all AI-assisted work. OpenAI ChatGPT and Codex were used as assistive tools; all AI-generated or modified helper code was inspected and tested by the authors, and all AI-assisted LaTeX, tables, figures, and manuscript edits were reviewed and revised by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the information necessary to reproduce our experiments throughout the main paper and appendices, including task and data construction, data splits, adapter-bank construction, canonicalization, generator training, sampling and support-based selection, hyperparameters, random seeds, and evaluation protocols. The supplementary material includes anonymized implementation and evaluation code, experiment configurations, and scripts corresponding to the four stages of STAGE. The task-specific LoRA banks will be released through a public data repository upon acceptance; scripts and configurations for reconstructing the banks are included in the supplementary material.

REFERENCES

Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha S. Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *International Conference on Learning Representations*, 2023.

Oana-Maria Camburu, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom. e-SNLI: Natural language inference with natural language explanations. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/4c7a167bb329bd92580a99ce422d6fa6-Abstract.html.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-LoRA: Instant transformer adaption. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 7485–7514. PMLR, 2025. URL https://proceedings.mlr.press/v267/charakorn25a.html.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-LoRA: Learning to instantly internalize contexts, 2026. URL https://arxiv.org/abs/2602.15902.

Tong Chen, Hao Fang, Patrick Xia, Xiaodong Liu, Ben Van Durme, Luke Zettlemoyer, Jianfeng Gao, and Hao Cheng. Generative adapter: Contextualizing language models in parameters with a single forward pass. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/447708674d7f30492ca5338b36b07d0c-Abstract-Conference.html.

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

Ziya Erkoç, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. HyperDiffusion: Generating implicit neural fields with weight-space diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 14300–14310, 2023. URL https://openaccess.thecvf.com/content/ICCV2023/html/Erkoc_HyperDiffusion_Generating_Implicit_Neural_Fields_with_Weight-Space_Diffusion_ICCV_2023_paper.html.

Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4367–4375, 2018. URL https://openaccess.thecvf.com/content_cvpr_2018/html/Gidaris_Dynamic_Few-Shot_Visual_CVPR_2018_paper.html.

Saumya Gupta, Scott Biggs, Moritz Laber, Zohair Shafi, Robin Walters, and Ayan Paul. Deepweightflow: Re-basined flow matching for generating neural network weights. In *International Conference on Learning Representations (ICLR)*, 2026.

David Ha, Andrew Dai, and Quoc V. Le. HyperNetworks, 2016. URL https://arxiv.org/abs/1609.09106.

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. *arXiv preprint arXiv:2603.10090*, 2026.

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*, 2021. URL https://arxiv.org/abs/2103.03874.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

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 *International Conference on Learning Representations*, 2026.

Hamish Ivison, Akshita Bhagia, Yizhong Wang, Hannaneh Hajishirzi, and Matthew Peters. HINT: Hypernetwork instruction tuning for efficient zero- and few-shot generalisation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11272–11288. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.631. URL https://aclanthology.org/2023.acl-long.631/.

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.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 565–576. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-long.47. URL https://aclanthology.org/2021.acl-long.47/.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. CS231N Course Project, Stanford University, 2015.

Mengtian Li, Jinshu Chen, Wanquan Feng, Bingchuan Li, Fei Dai, Songtao Zhao, and Qian He. HyperLoRA: Parameter-efficient adaptive generation for portrait synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13114–13123, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Li_HyperLoRA_Parameter-Efficient_Adaptive_Generation_for_Portrait_Synthesis_CVPR_2025_paper.html.

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. *arXiv preprint arXiv:2506.16406*, 2025.

Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. CommonGen: A constrained text generation challenge for generative commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1823–1840. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.findings-emnlp.165. URL https://aclanthology.org/2020.findings-emnlp.165/.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*, 2023.

Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. SHINE: A scalable in-context hypernetwork for mapping context to LoRA in a single pass, 2026. URL https://arxiv.org/abs/2602.06358.

Chuancheng Lv, Lei Li, Shitou Zhang, Gang Chen, Fanchao Qi, Ningyu Zhang, and Hai-Tao Zheng. HyperLoRA: Efficient cross-task generalization via constrained low-rank adapters generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 16376–16393. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-emnlp.956. URL https://aclanthology.org/2024.findings-emnlp.956/.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Pierre Isabelle, Eugene Charniak, and Dekang Lin (eds.), *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040/.

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 *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 27854–27875. PMLR, 2023. URL https://proceedings.mlr.press/v202/phang23a.html.

Theo Putterman, Derek Lim, Yoav Gelberg, Stefanie Jegelka, and Haggai Maron. Learning on LoRAs: GL-equivariant processing of low-rank weight spaces for large finetuned models, 2024. URL https://arxiv.org/abs/2410.04207.

James Requeima, Jonathan Gordon, John Bronskill, Sebastian Nowozin, and Richard E. Turner. Fast and flexible multi-task classification using conditional neural adaptive processes. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/1138d90ef0a0848a542e57d1595f58ea-Abstract.html.

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Wei Wei, Tingbo Hou, Yael Pritch, Neal Wadhwa, Michael Rubinstein, and Kfir Aberman. HyperDreamBooth: HyperNetworks for fast personalization of text-to-image models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6527–6536, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Ruiz_HyperDreamBooth_HyperNetworks_for_Fast_Personalization_of_Text-to-Image_Models_CVPR_2024_paper.html.

Daniel Saragih, Deyu Cao, Tejas Balaji, and Ashwin Santhosh. Flow to learn: Flow matching on neural network parameters. In *ICLR Workshop on Neural Network Weights as a New Data Modality*, 2025.

Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-representations as generative models: Sampling unseen neural network weights. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27906–27920, 2022.

Konstantin Schürholt, Michael W. Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43947–43966. PMLR, 2024. URL https://proceedings.mlr.press/v235/schurholt24a.html.

Rana Shahroz, Dongwen Tang, Pingzhi Li, Kai Wang, and Tianlong Chen. ORAL: Prompting your large-scale LoRAs via conditional recurrent diffusion. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 1357–1370. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.71.

Yihua Shao, Minxi Yan, Yang Liu, Siyu Chen, Wenjie Chen, Xinwei Long, Ziyang Yan, Lei Li, Chenyu Zhang, Nicu Sebe, Hao Tang, Yan Wang, Hao Zhao, Mengzhu Wang, and Jingcai Guo. In-context meta LoRA generation. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 6138–6146, 2025. doi: 10.24963/ijcai.2025/683.

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*, 2025.

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 *Advances in Neural Information Processing Systems*, volume 38, 2025.

Yujia Wu, Yiming Shi, Jiwei Wei, Chengwei Sun, Yang Yang, and Heng Tao Shen. DiffLoRA: Generating personalized low-rank adaptation weights with diffusion models. *arXiv preprint arXiv:2408.06740*, 2024.

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 *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68459–68471. PMLR, 2025. URL https://proceedings.mlr.press/v267/xiao25e.html.

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. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.365. URL https://aclanthology.org/2025.findings-acl.365/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. doi: 10.48550/arXiv.2505.09388. URL https://arxiv.org/abs/2505.09388.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. MetaMath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=N8N0hgNDRt.

Boya Zeng, Yida Yin, Zhiqiu Xu, and Zhuang Liu. Generative modeling of weights: Generalization or memorization? *arXiv preprint arXiv:2506.07998*, 2025.

Baoquan Zhang, Chuyao Luo, Demin Yu, Xutao Li, Huiwei Lin, Yunming Ye, and Bowen Zhang. MetaDiff: Meta-learning with conditional diffusion for few-shot learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 16687–16695, 2024. doi: 10.1609/aaai.v38i15.29608. URL https://ojs.aaai.org/index.php/AAAI/article/view/29608.

Andrey Zhmoginov, Mark Sandler, and Maksym Vladymyrov. HyperTransformer: Model generation for supervised and semi-supervised few-shot learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 27075–27098. PMLR, 2022. URL https://proceedings.mlr.press/v162/zhmoginov22a.html.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

Allan Zhou, Kaien Yang, Kaylee Burns, Adriano Cardace, Yiding Jiang, Samuel Sokota, J. Zico Kolter, and Chelsea Finn. Permutation equivariant neural functionals. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2302.14040.

APPENDIX

A Notation **16**

B Hyperparameters of STAGE Models **17**

B.1 Image-classification configuration. 18

B.2 Text-generation configuration. 19

C Implementation Details of the STAGE Framework **20**

C.1 Pipeline Overview 20

C.2 Generator Architectures 20

C.3 Flow Training and Checkpoint Selection 20

C.4 Sampling, Selection, and Evaluation 21

D Step-by-Step Results of STAGE **22**

D.1 Frozen Base Models and Full-Gradient References 22

D.2 Step 1: Task-Specific LoRA Bank 26

D.3 Step 2 and 3: Canonical Targets, Task Conditions, and Generator Training 29

D.4 Step 4: Generated Adapters and Support-Based Selection 29

E Additional Results & Experiments **31**

E.1 LoRA Bank-Size Ablations 31

F Computational Cost of Adaptation **32**

G Additional Similarity Diagnostics **34**

G.1 Extended Generated-Adapter Similarity Matrices 34

G.2 Five-Task Diagnostic 35

G.3 Paired 100-Task Similarity–Accuracy Evaluation 35

A NOTATION

Table 5 summarizes the notation used to describe STAGE.

Table 5: Notation for task construction, canonical LoRA representation, conditional flow matching, and adapter generation. Matrix dimensions refer to a single adapted weight matrix. Classifier equations describe the image implementation.

Symbol	Description
Tasks and data	
$\tau,\ (x, y)$	Task index and an input–output example, respectively.
$\mathcal{D}_\tau^s,\ N_s$	Labeled support set and its size; used for conditioning and candidate selection.
$\mathcal{D}_\tau^{\mathrm{fit}}$	Target-adapter fitting set; disjoint from support in the image bank and containing support for text.
$\mathcal{D}_\tau^q$	Query evaluation set, disjoint from support and fitting.
Base model and LoRA updates	
$h_\psi(x),\ \psi$	Backbone features and frozen backbone parameters.
$d_{\mathrm{in}},\ d_{\mathrm{out}}$	Input and output dimensions of the adapted weight matrix.
$W_0,\ b$	Frozen classifier weight in $\mathbb{R}^{d_{\mathrm{out}} \times d_{\mathrm{in}}}$ and bias in $\mathbb{R}^{d_{\mathrm{out}}}$.
$f_0(x),\ f_\tau(x)$	Frozen and adapted classifier logits: $W_0 h_\psi(x) + b$ and $(W_0 + \Delta W_\tau)h_\psi(x) + b$.
$r,\ \alpha$	LoRA rank and scaling parameter, respectively.
$A_\tau,\ B_\tau$	LoRA factors in $\mathbb{R}^{r \times d_{\mathrm{in}}}$ and $\mathbb{R}^{d_{\mathrm{out}} \times r}$, respectively.
ΔW_τ	Effective update, $\Delta W_\tau = (\alpha/r) B_\tau A_\tau$.
$A_\tau^*,\ B_\tau^*,\ \Delta W_\tau^*$	Fitted target factors and their effective update, obtained using $\mathcal{D}_\tau^{\mathrm{fit}}$.
Canonical representation and conditioning	
$U_\tau,\ \Sigma_\tau,\ V_\tau$	SVD components: $\Delta W_\tau^* = U_\tau \Sigma_\tau V_\tau^\top$.
$\widetilde{U}_{\tau,r},\ \Sigma_{\tau,r},\ \widetilde{V}_{\tau,r}$	Retained SVD components, with paired sign adjustment applied to the singular vectors; r denotes the retained components.
$\bar{A}_\tau,\ \bar{B}_\tau$	SVD-based canonical factors satisfying $(\alpha/r)\bar{B}_\tau \bar{A}_\tau = \Delta W_\tau^*$.
$\mathrm{vec}(\cdot,\cdot),\ u_\tau$	Factor flattening and concatenation in a fixed order; $u_\tau = \mathrm{vec}(\bar{A}_\tau, \bar{B}_\tau)$.
$\mu,\ \sigma$	Elementwise mean and standard deviation of u_τ, computed from the generator-training split only.
a_τ	Standardized target vector, $a_\tau = (u_\tau - \mu)/\sigma$, using elementwise division.
$E_\phi,\ c_\tau$	Support encoder with trainable parameters ϕ and task condition $c_\tau = E_\phi(\mathcal{D}_\tau^s)$.
Flow training and adapter generation	
$z_0,\ t$	Initial noise $z_0 \sim \mathcal{N}(0, I)$ matching the dimension of a_τ, and flow time $t \sim \mathcal{U}(0, 1)$.
z_t	Interpolated training state, $z_t = (1 - t)z_0 + t\, a_\tau$.
$v_\theta(z_t, t, c_\tau)$	Conditional velocity network with trainable parameters θ.
$\mathcal{L}_{\mathrm{FM}}$	Mean squared velocity error: $\mathbb{E}_{\tau, z_0, t}[\|\|v_\theta(z_t, t, c_\tau) - (a_\tau - z_0)\|\|_2^2]$.
$N,\ n$	Number of generated candidates and candidate index, $n \in \{1, \ldots, N\}$.
$L,\ \ell$	Number of Euler steps and update index, $\ell = 0, \ldots, L - 1$; step size is $1/L$.
$z_\ell^{(n)}$	Candidate n's integration state at time ℓ/L; $z_L^{(n)}$ is its terminal state.
$\hat{u}_\tau^{(n)},\ \odot$	Denormalized vector $\hat{u}_\tau^{(n)} = \sigma \odot z_L^{(n)} + \mu$, where $\odot$ denotes elementwise multiplication.
$\hat{A}_\tau^{(n)},\ \hat{B}_\tau^{(n)}$	Generated factors unpacked from $\hat{u}_\tau^{(n)}$, with the same dimensions as A_τ, B_τ.
$\widehat{\Delta W}_\tau^{(n)}$	Candidate update, $(\alpha/r)\hat{B}_\tau^{(n)} \hat{A}_\tau^{(n)}$.
$\mathcal{L}_{\mathrm{CE}},\ \mathcal{L}_{\mathrm{sup}}^{(n)}$	Cross-entropy loss and candidate n's mean cross-entropy on the support set, respectively.
n^*	Support-selected index, $n^* = \arg\min_{n \in \{1,\ldots,N\}} \mathcal{L}_{\mathrm{sup}}^{(n)}$.

B HYPERPARAMETERS OF STAGE MODELS

Tables 6 and 7 summarize the target-adapter, generator-training, and evaluation configurations for image classification and text generation, respectively. Image generators are trained separately for each backbone–dataset pair. The text configuration corresponds to the shared generator trained on $140{,}000$ adapters across 14 datasets. Bank-size ablations vary the target counts as described in Appendix D.2.

B.1 IMAGE-CLASSIFICATION CONFIGURATION.

Table 6: **Image-classification configuration.** Target construction, generator architecture, training, and evaluation settings. Backbone-specific values are ordered as ViT-S/16, EfficientNet-V2-S, and ConvNeXt-V2 Tiny.

Setting	Value
Task and target-adapter configuration	
Datasets	CIFAR-100 and Tiny-ImageNet
Backbones	ViT-S/16, EfficientNet-V2-S, ConvNeXt-V2 Tiny
Target-bank size	100,000 tasks per backbone–dataset pair
Generator train / validation	90,000 / 10,000 tasks
Task arity	10 classes
Support set	1 labeled image/class
Target-LoRA fitting set	10 additional images/class
Query set	20 images/class, disjoint from support and fitting
Adapted module	Existing classifier-head weight
Classifier output dimension	100 for CIFAR-100; 200 for Tiny-ImageNet
LoRA rank / scaling / dropout	$r = 32,\ \alpha = 64,\ 0$
Target representation	Normalized deterministic canonical SVD factors
Generator architecture	
Support input	Cached frozen features with task-local label embeddings
Support Transformer	Hidden width 1024; 6 layers; 16 heads
Task-condition dimension	1024
Label / time embedding dimension	128 / 128
Velocity architecture	FiLM-modulated residual network
CIFAR-100 velocity width	5120 / 4096 / 4096
CIFAR-100 velocity depth	12 / 10 / 10
CIFAR-100 velocity dropout	0.15 / 0.10 / 0.20
Tiny-ImageNet velocity width	4096 for all backbones
Tiny-ImageNet velocity depth	10 for all backbones
Tiny-ImageNet velocity dropout	0.10 for all backbones
Generator training and checkpoint selection	
Optimizer / learning rate	Adafactor / 10^{-3}
CIFAR-100 weight decay	0.01 / 0.01 / 0.02
Tiny-ImageNet weight decay	0.01 for all backbones
Batch size / gradient accumulation	64 / 1
Gradient clipping	1.0
Warm-up	5 epochs
Learning-rate schedule	Cosine decay; minimum LR ratio 0.05
Maximum training budget	120 epochs
Early-stopping patience	15 epochs
Minimum validation improvement	10^{-4}
Checkpoint selection	Lowest validation flow loss
Sampling and official evaluation	
Evaluation tasks	100 per backbone–dataset pair
Evaluation pool	CIFAR-100 test; Tiny-ImageNet validation
Support / query separation	Disjoint image indices within each task
Generated adapters/task	16
Flow integration	96 Euler steps
Deployable candidate selection	Lowest labeled-support loss
Reported query metrics	Frozen, generated mean, support-selected, Best-of-16
Oracle diagnostic	Best-of-16 selected using query accuracy
Evaluation seed	9147

B.2 TEXT-GENERATION CONFIGURATION.

Table 7: **Text-generation configuration.** Target construction, generator architecture, training, and evaluation settings for the shared 14-dataset generator. Candidate counts follow the reported domain-specific compute configuration; commonsense bank-size ablations use 16 candidates.

Setting	Value
Task and target-adapter configuration	
Base model	Qwen3-1.7B (frozen)
Domains	Commonsense, Math, SQL, and Python
Target-bank size	140,000 tasks: 14 datasets $\times$ 10,000
Generator train / validation	138,600 / 1,400 tasks
Task composition	Rows from a single dataset
Support set	10 rows/task, drawn from the fitting set
Target-LoRA fitting set	100 rows/task
Query set	100 rows/task, disjoint from fitting rows
Adapted modules	`gate_proj`, `up_proj`, and `down_proj` in all 28 layers
LoRA rank / scaling / dropout	$r = 8,\ \alpha = 16,\ 0$
Target-LoRA optimizer	AdamW
Target-LoRA learning rate	5×10^{-4}
Target-LoRA batch size / steps	8 / 100
Target-LoRA loss	Response-token negative log-likelihood
Target representation	Deterministic canonical SVD factors
Generator architecture	
Condition dimension	4096
Support Transformer	Hidden width 1024; 4 layers; 16 heads
Time embedding dimension	1536
Velocity architecture	DiT with adaLN-single
Velocity blocks / width / heads	24 / 1536 / 24
Condition dropout	0.1
Generator training and checkpoint selection	
Optimizer / learning rate	AdamW / 10^{-4}
Batch size / gradient accumulation	128 / 1
Gradient clipping	1.0
Warm-up	1,000 steps
Learning-rate schedule	Cosine decay; minimum LR ratio 0.1
Maximum training budget	21,900 steps
Validation interval	500 steps
Checkpoint selection	Validation flow loss
Observed stopping point	Step 14,000 (approximately 12.9 epochs); validation loss plateaued
Sampling and evaluation	
Evaluation tasks	200 per dataset
Evaluation pool	Official test split of each dataset
Support / query separation	Support is a subset of fitting rows; query rows are disjoint from fitting rows
Generated adapters/task	16 for Commonsense and SQL; 6 for Math and Python
Flow integration	96 Euler steps
Deployable candidate selection	Highest support score
Query metrics	Accuracy for multiple-choice tasks; BLEU-4 for CommonGen; task-specific correctness, execution accuracy, or pass rate for Math, SQL, and Python
Oracle diagnostic	Best-of-N selected using query performance
Evaluation seed	0

C Implementation Details of the STAGE Framework

This appendix describes support conditioning, generator architectures, flow training, and evaluation for STAGE. Target-bank construction and canonicalization are described in Appendix D.2, and full hyperparameters are listed in Appendix B.

C.1 Pipeline Overview

STAGE fits task-specific target adapters, converts them into canonical representations, and trains a conditional flow generator on the resulting support–adapter pairs. Image generators are trained separately for each dataset–backbone pair. For text, a single generator is trained on 140,000 target tasks from 14 datasets (10,000 per dataset), with 138,600 used for training and 1,400 for validation.

Image tasks use one support image, ten separate fitting images, and twenty query images per class. Text tasks use 100 fitting rows, ten of which serve as support, and 100 disjoint query rows. Query examples are never used to fit target adapters or to train the generator. At deployment, the generator produces candidate adapters from the support condition and independent Gaussian noise; one candidate is selected using support examples only and evaluated on the query set, with no task-specific gradient updates.

C.2 Generator Architectures

Image classification. We freeze the backbone, classifier weight, and classifier bias of the checkpoint used to build the target bank, and cache all image features. Each task has ten classes with one support image per class. A six-layer support Transformer (hidden dimension 1024, 16 heads) takes the ten feature vectors together with learned 128-dimensional task-local label embeddings and produces a 1024-dimensional task representation c_τ. These local labels only describe the support set; the classifier keeps its dataset-wide output space (100 logits for CIFAR-100, 200 for Tiny-ImageNet).

The velocity network is a FiLM-modulated residual network that operates on the normalized canonical adapter vector, conditioned on c_τ and a 128-dimensional flow-time embedding. Its width, depth, dropout, and weight decay vary by dataset–backbone pair. After integration, the output is denormalized with training-partition statistics and decoded into a low-rank update $\Delta W = (\alpha/r)BA$, which is applied only to the classifier-head weight.

Text generation. Text adapters modify `gate_proj`, `up_proj`, and `down_proj` in all 28 layers of a frozen Qwen3-1.7B; all other weights remain fixed. To make bank construction efficient, several adapters (four, or eight for short commonsense examples) are fitted concurrently on a shared backbone, with each batch row routed through its own adapter and per-adapter losses kept separate. Task-derived seeds make fitting reproducible and resumable across shards.

For conditioning, each support row is formatted as "Question: x / Answer: y", and the frozen model's hidden states after the answer marker and at the sequence end are concatenated into a cached 4096-dimensional feature. Unlike the image setting, the ten support rows are drawn from the adapter's fitting set, while query rows remain disjoint. A four-layer support Transformer (hidden dimension 1024, 16 heads) encodes these features. Canonical adapter factors are arranged as structured tokens for a DiT velocity network with 24 blocks, width 1536, and 24 heads, conditioned via adaLN-single. Generated representations are decoded into low-rank factors and loaded into the frozen model.

C.3 Flow Training and Checkpoint Selection

The generator is trained with a conditional flow objective to predict velocities between Gaussian noise and canonical target adapters. Only the support encoder and velocity network are updated.

Image generators. Each generator uses 90,000 training and 10,000 validation tasks, and is trained with Adafactor (learning rate 10^{-3}, batch size 64, five warm-up epochs, cosine decay, gradient clipping at 1.0) for up to 120 epochs, with early-stopping patience of 15 epochs and minimum improvement 10^{-4}. We evaluate the checkpoint with the lowest validation flow loss, not necessarily the last epoch. Table 8 reports canonical-vector dimensions, selected epochs, and flow losses for all

Table 8: Flow-generator training summary. "Train at best" and "Best validation" are measured at the same validation-selected epoch.

Target pool	Backbone	Vector dim.	Best epoch	Train at best	Best val.	Final epoch	Final train / val.
CIFAR-100 all-100	ViT-S/16	15,488	32	1.6284	1.7007	47	1.5726 / 1.7151
CIFAR-100 all-100	EfficientNet-V2-S	44,160	117	1.4582	1.5172	120	1.4578 / 1.5216
CIFAR-100 all-100	ConvNeXt-V2 Tiny	27,776	118	1.2779	1.3301	120	1.2781 / 1.3316
Tiny all-200	ViT-S/16	18,688	50	1.6434	1.7024	65	1.6169 / 1.7054
Tiny all-200	EfficientNet-V2-S	47,360	112	1.1765	1.1913	120	1.1764 / 1.1929
Tiny all-200	ConvNeXt-V2 Tiny	30,976	110	1.3029	1.3848	120	1.3010 / 1.3868
Tiny seen-180	ViT-S/16	18,688	51	1.6384	1.6995	66	1.6104 / 1.7057
Tiny seen-180	EfficientNet-V2-S	47,360	110	1.1772	1.1893	120	1.1772 / 1.1895
Tiny seen-180	ConvNeXt-V2 Tiny	30,976	118	1.2953	1.3689	120	1.2949 / 1.3704

Table 9: Flow-generator training summary for the text experiments. One generator is trained on all 140,000 target tasks; rows split its loss by dataset. "Train at best", "Best val." and "Best test" are measured at the validation-selected epoch, on 100 training tasks, the 100 held-out validation tasks and the 200 test-task target LoRAs of each dataset. Training was stopped once validation loss was flat, so the final epoch coincides with the best one.

Target pool	Backbone	Vector dim.	Best epoch	Train at best	Best val.	Best test	Final epoch	Final train / val. / test
ART	Qwen3-1.7B	5,505,024	12.9	1.1077	1.1081	1.1483	12.9	1.1077 / 1.1081 / 1.1483
PIQA	Qwen3-1.7B	5,505,024	12.9	1.1644	1.1619	1.1703	12.9	1.1644 / 1.1619 / 1.1703
ECQA	Qwen3-1.7B	5,505,024	12.9	1.2433	1.2490	1.2462	12.9	1.2433 / 1.2490 / 1.2462
e-SNLI	Qwen3-1.7B	5,505,024	12.9	1.4761	1.4763	1.4894	12.9	1.4761 / 1.4763 / 1.4894
CommonGen	Qwen3-1.7B	5,505,024	12.9	1.4536	1.4544	1.5126	12.9	1.4536 / 1.4544 / 1.5126
GSM8K	Qwen3-1.7B	5,505,024	12.9	1.5978	1.5947	1.6006	12.9	1.5978 / 1.5947 / 1.6006
MATH	Qwen3-1.7B	5,505,024	12.9	1.7149	1.7133	1.7180	12.9	1.7149 / 1.7133 / 1.7180
MetaMathQA	Qwen3-1.7B	5,505,024	12.9	1.5157	1.5164	1.5096	12.9	1.5157 / 1.5164 / 1.5096
gretel	Qwen3-1.7B	5,505,024	12.9	1.3724	1.3656	1.3725	12.9	1.3724 / 1.3656 / 1.3725
Spider	Qwen3-1.7B	5,505,024	12.9	1.3787	1.3812	1.3628	12.9	1.3787 / 1.3812 / 1.3628
WikiSQL	Qwen3-1.7B	5,505,024	12.9	1.2864	1.2871	1.2857	12.9	1.2864 / 1.2871 / 1.2857
KodCode-Filter	Qwen3-1.7B	5,505,024	12.9	1.5874	1.5910	1.5893	12.9	1.5874 / 1.5910 / 1.5893
KodCode-Leetcode	Qwen3-1.7B	5,505,024	12.9	1.6243	1.6275	1.6277	12.9	1.6243 / 1.6275 / 1.6277
KodCode-Contest	Qwen3-1.7B	5,505,024	12.9	1.5817	1.5846	1.5820	12.9	1.5817 / 1.5846 / 1.5820
All 14 datasets	Qwen3-1.7B	5,505,024	12.9	1.4370	1.4377	1.4447	12.9	1.4370 / 1.4377 / 1.4447

image generators, including the separate Tiny-ImageNet seen-180 generators. Most runs select a late epoch close to the 120-epoch budget, while the ViT-S/16 generators stop early; in all cases the train–validation gap at the selected epoch is small.

Text generator. The shared text generator is trained with AdamW (learning rate 10^{-4}, batch size 128, 1,000 warm-up steps, cosine decay, gradient clipping at 1.0, condition dropout 0.1), with validation every 500 steps. Training was stopped at step 14,000 (about 12.9 epochs), before the 21,900-step budget, once validation loss had plateaued; the final checkpoint is therefore also the validation-selected one. Table 9 breaks down its flow loss by dataset, using 100 training tasks, 100 held-out validation tasks, and 200 test-task target adapters per dataset. These are parameter-space diagnostics, not downstream accuracy, and test-target losses are not used for checkpoint selection. Train, validation, and test losses are close for every dataset, indicating little overfitting to the training tasks, while loss levels vary considerably across datasets.

C.4 Sampling, Selection, and Evaluation

Sampling integrates the learned velocity field from Gaussian noise with 96 forward-Euler steps. Image evaluation generates 16 candidates per task. For text, we use 16 candidates for Commonsense and SQL, six for Math and Python, and 16 for the Commonsense bank-size ablations.

We report three metrics for generated adapters. *Generated mean* averages query performance over all candidates. *Support-selected*, our deployable result, picks the candidate with the lowest labeled-support loss (images) or highest support score (text). *Best-of-N* reports the best query performance among N candidates and serves only as an oracle diagnostic; query labels are never used for support-selected results.

Image accuracy is computed on query examples from the ten sampled classes using the dataset-wide output head, so it is not directly comparable to conventional full-dataset Acc.@1. Text evaluation uses each dataset's standard metric: multiple-choice accuracy, BLEU-4 for CommonGen, execution accuracy for SQL, and pass rate for Python. Evaluation seeds are fixed at 9147 for images and 0 for text.

D STEP-BY-STEP RESULTS OF STAGE

This appendix reports results for each step of the STAGE pipeline (Figure 2). Appendix D.1 first describes the frozen base models and the full-gradient references used throughout. Appendix D.2 covers Step 1: task construction, adapter fitting, and target quality. Appendix D.3 summarizes Steps 2 and 3: canonicalization, task conditioning, and generator training. Appendix D.4 reports Step 4: generated-adapter accuracy, behavior on held-out classes, and the effect of bank size. Unless noted otherwise, image accuracies are 10-way episodic query accuracies computed with the original dataset-wide classifier head. Text scores use each dataset's standard metric: accuracy, BLEU-4 for CommonGen, execution accuracy for SQL, and pass rate for Python.

D.1 FROZEN BASE MODELS AND FULL-GRADIENT REFERENCES

Every stage operates on a frozen base model. Separately, the *Full SFT* comparisons in Stage 4 use full-gradient models trained on the same data.

Image classifiers. We train EfficientNet-V2-S, ConvNeXt-V2 Tiny, and ViT-S/16 with full gradients on CIFAR-100 and Tiny-ImageNet. Table 10 reports their full-class accuracy over 100 and 200 classes, respectively. These numbers describe the classifiers themselves and are not directly comparable to the 10-way episodic accuracies used elsewhere. Figure 6 shows the corresponding learning curves. ViT-S/16 reaches the highest official accuracy on both datasets. These full-gradient classifiers also serve as the Full SFT references in Stage 4.

Table 10: Full-gradient baseline performance on CIFAR-100 and Tiny-ImageNet. Acc.@1 denotes full-class top-1 accuracy (%) over 100 and 200 classes, respectively. Base reports the initial evaluation before training. Validation includes internal validation or development results. Official test uses the CIFAR-100 test split and Tiny-ImageNet validation split. The highest official accuracy within each dataset is bolded and highlighted. Missing or omitted results are marked –.

Dataset	Backbone	Base		Training		Validation		Official test	
		Loss	Acc.@1	Loss	Acc.@1	Loss	Acc.@1	Loss	Acc.@1
CIFAR-100	ViT-S/16 AugReg	4.9939	1.50	0.8549	98.25	0.3701	91.82	0.3804	**91.92**
	EfficientNet-V2-S†	–	–	1.0643	99.35	1.3732	90.66	1.3732	90.66
	ConvNeXt-V2 Tiny‡	–	–	2.1423	–	–	–	0.6163	87.66
Tiny-ImageNet	ViT-S/16 AugReg	5.3125	0.50	0.6531	83.87	–	–	0.4263	**88.66**
	EfficientNet-V2-S	6.1534	0.55	0.8575	99.90	1.0883	78.60	1.1098	78.29
	ConvNeXt-V2 Tiny	5.2986	0.55	1.9716	82.09	0.9899	80.85	0.9665	81.45

Checkpoints. For CIFAR-100: ViT-S/16, final epoch 50/50; EfficientNet-V2-S, best epoch 85/104; ConvNeXt-V2 Tiny, best epoch 276/300. For Tiny-ImageNet: ViT-S/16, final/best step 10k/10k; EfficientNet-V2-S, final step 20k/20k; ConvNeXt-V2 Tiny, final epoch 300/300.
† EfficientNet-V2-S used the CIFAR-100 test split for checkpoint selection without an independent validation split; its validation and official-test entries therefore coincide. ‡ ConvNeXt-V2 Tiny training Acc.@1 on CIFAR-100 is omitted because Mixup/CutMix produces soft training labels. For Tiny-ImageNet, ViT-S/16 did not report an independent development result, while the other two backbones used internal development splits before official validation-set evaluation.

The frozen classifier used to build the LoRA bank is not necessarily the best full-gradient checkpoint. For some pairs, we freeze an earlier checkpoint so that task-specific adaptation still has meaningful headroom (Table 11). For example, the CIFAR-100 ViT-S/16 bank uses the epoch-1 checkpoint (82.5% validation accuracy) rather than the final model. The selected classifier is frozen before feature extraction and target fitting.

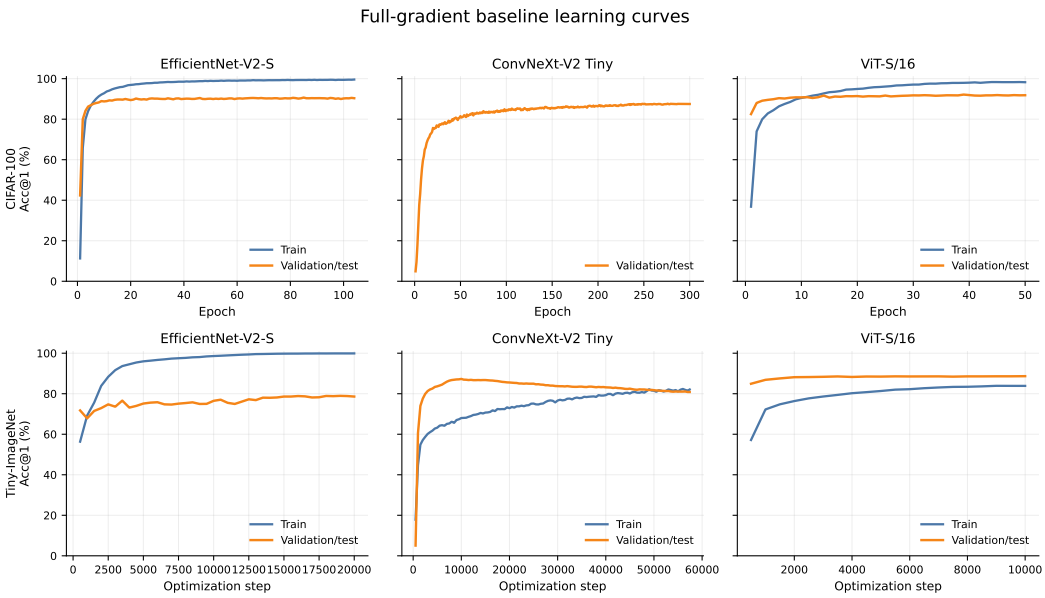

Figure 6: **Full-gradient baseline learning curves.** Training and validation/evaluation Acc.@1 for EfficientNet-V2-S, ConvNeXt-V2 Tiny, and ViT-S/16 on CIFAR-100 (top) and Tiny-ImageNet (bottom). The horizontal axis follows the training schedule used by the corresponding implementation.

Table 11: Frozen classifier checkpoints used to construct the 100k task-specific LoRA target libraries.

Dataset	Backbone	Adapter-source checkpoint
CIFAR-100	ViT-S/16	`epoch_001_valacc_82.5000.pt`
CIFAR-100	EfficientNet-V2-S	`ckpt-6`
CIFAR-100	ConvNeXt-V2 Tiny	`step_04000_top1_80.8000.pt`
Tiny-ImageNet	ViT-S/16	`step_00100_top1_81.2800.pt`
Tiny-ImageNet	EfficientNet-V2-S	`best_top1_78.9500_step_19000.pt`
Tiny-ImageNet	ConvNeXt-V2 Tiny	`last_testacc_81.4500.pt`

Language model. For text, the frozen base model is Qwen3-1.7B without task-specific fine-tuning. For the Full SFT reference, we fine-tune all of its parameters for three epochs on each dataset's own training pool. Table 12 reports loss and task metric before and after fine-tuning, and Figure 7 shows the learning curves. Full SFT improves official test performance on most datasets but lowers it on GSM8K and MATH.

Table 12: Full fine-tuning baseline performance. Each dataset is fine-tuned on its own training pool for 3 epochs. Loss is the average negative log-likelihood per response token. Metric is accuracy, except BLEU-4 for CommonGen, execution accuracy for SQL, and pass rate for Python. Base reports the model before fine-tuning on the official test pool; Official test reports the fine-tuned model on the same pool.

Domain	Dataset	Base		Training		Validation		Official test	
		Loss	Metric	Loss	Metric	Loss	Metric	Loss	Metric
Commonsense	ART	28.300	68.41	0.000	100.00	0.149	95.20	0.856	78.07
	PIQA	25.800	69.91	0.000	100.00	0.601	81.07	0.597	81.40
	ECQA	1.764	65.71	0.001	99.85	0.286	72.48	0.318	72.28
	e-SNLI	7.849	55.74	0.675	95.80	1.010	89.60	1.070	91.22
	CommonGen	6.685	15.04	1.233	35.49	2.026	24.56	2.302	25.08
Math	GSM8K	2.613	74.87	0.266	77.95	0.402	67.89	0.432	64.62
	MATH	1.932	36.82	0.496	41.75	0.637	38.02	0.629	35.15
	MetaMathQA	1.353	79.29	0.088	93.35	0.175	83.30	0.169	83.91
SQL	gretel	2.299	54.30	0.069	88.65	0.185	71.35	0.179	72.45
	Spider	5.158	38.74	0.032	91.40	0.084	82.16	0.347	66.48
	WikiSQL	1.962	53.51	0.004	97.50	0.023	91.20	0.024	89.83
Python	KodCode-Filter	1.170	21.93	0.170	54.85	0.298	40.20	0.299	39.09
	KodCode-Leetcode	1.177	16.25	0.186	56.20	0.302	36.51	0.301	35.56
	KodCode-Contest	1.332	25.10	0.160	70.80	0.286	51.88	0.280	53.99

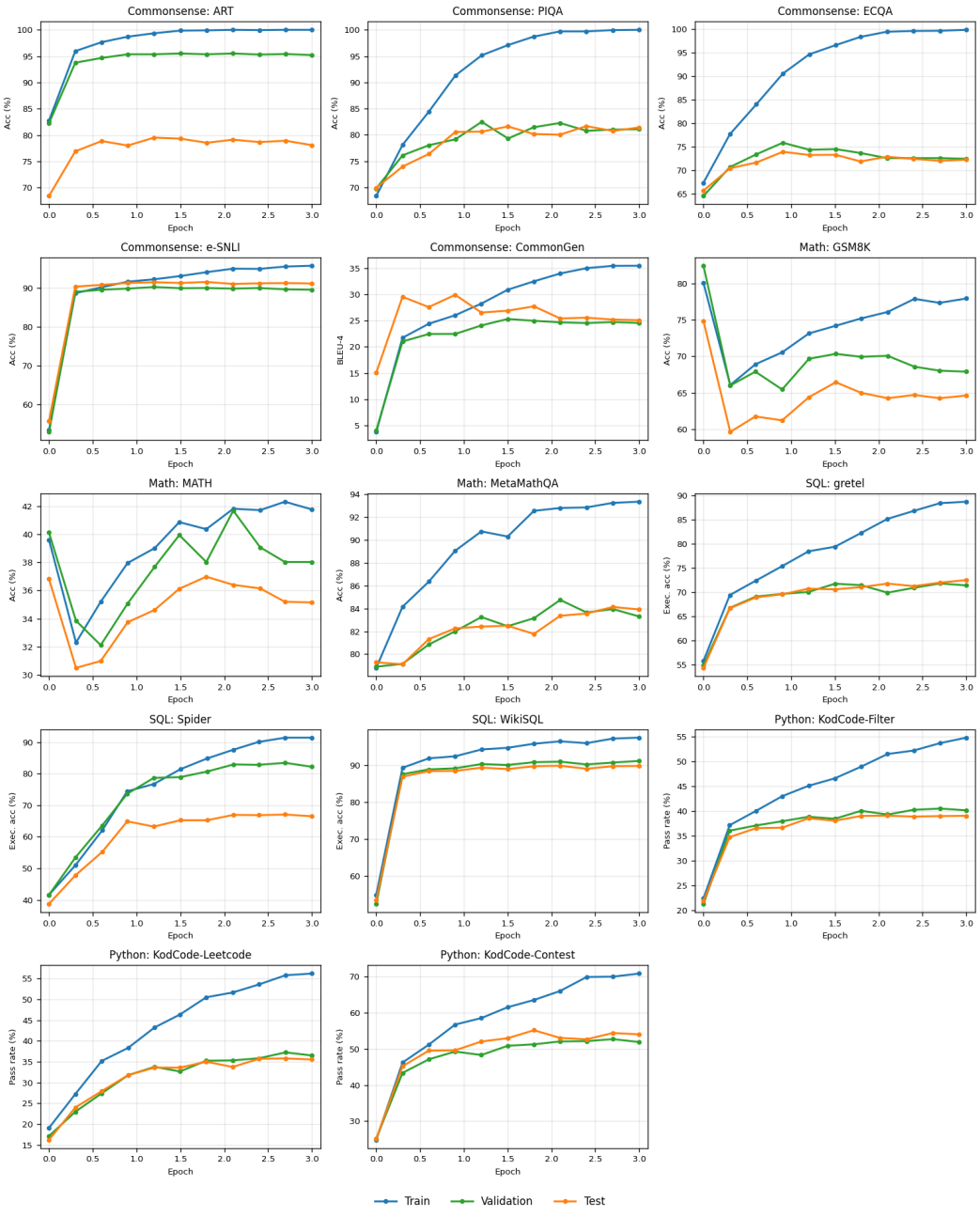

Figure 7: **Full-parameter SFT baseline learning curves.** Training, validation, and test performance of Qwen3-1.7B fine-tuned for 3 epochs on each dataset's own training pool, evaluated at every 10% of training (epoch 0 is the base model). Metric is accuracy, except BLEU-4 for CommonGen, execution accuracy for SQL, and pass rate for Python.

D.2 STEP 1: TASK-SPECIFIC LORA BANK

Step 1 fits one LoRA adapter per sampled task on the frozen base model. We describe how tasks and adapters are constructed, how the bank is built efficiently, and how well the fitted targets perform.

D.2.1 TASK AND ADAPTER CONSTRUCTION

Image tasks. Each task τ is a 10-way classification problem with three disjoint sets per class: one support image ($\mathcal{D}_\tau^s$), ten fitting images ($\mathcal{D}_\tau^{\text{fit}}$), and twenty query images ($\mathcal{D}_\tau^q$). Each task therefore has 10 support, 100 fitting, and 200 query images. The target adapter is fitted only on $\mathcal{D}_\tau^{\text{fit}}$. The support set is kept as the generator's condition, and the query set is used only to measure target quality. Support and fitting images come from the training pool. Query images may come from the official evaluation pool, but they are never used to optimize the target adapter or the generator. Disjointness is enforced within each task, so classes and images can reappear across tasks. We build $100{,}000$ tasks per backbone–dataset pair, sampling from all 100 CIFAR-100 classes and all 200 Tiny-ImageNet classes. We also build a Tiny-ImageNet seen-180 bank, in which 20 classes are excluded from both target fitting and generator training.

Each target is a rank-32 LoRA ($\alpha = 64$, no dropout) on the weight of the existing linear classifier head, with $\Delta W = (\alpha/r)BA$. The backbone, classifier weight, and classifier bias stay frozen. The classifier keeps its original output space (100 logits for CIFAR-100, 200 for Tiny-ImageNet). The 10-way task only determines which examples appear in the episode; it does not replace the head with a 10-output classifier.

Text tasks. Each text task is drawn from one of 14 datasets in four domains (commonsense, math, SQL, and Python). It consists of 100 fitting rows (prompt–response pairs), a support set of 10 rows taken from those fitting rows, and 100 query rows disjoint from the fitting set. The target adapter is fitted on all 100 fitting rows, the generator sees only the 10 support rows, and query rows are used only for evaluation. Bank tasks come from each dataset's training pool: 90% of the official training split, with the remainder held out by prompt as a validation pool. This gives $10{,}000$ tasks per dataset and $140{,}000$ target adapters in total. Rows are sampled without replacement within a task but can reappear across tasks. The 200 test tasks per dataset are built the same way from the official test split, with the support rows fixed by a seed so that every method is conditioned on the same rows. No test-pool row is used to train the bank or the generator.

Each target is a rank-8 LoRA ($\alpha = 16$, no dropout) on `gate_proj`, `up_proj`, and `down_proj` in all 28 layers of Qwen3-1.7B, giving 84 adapted matrices and 5.5M trainable parameters per adapter. Attention projections, embeddings, the language-model head, and the original feed-forward weights stay frozen. A is initialized with Kaiming-uniform and B with zeros, using an independent seed for every adapter. Each target is trained for 100 AdamW steps (learning rate 5×10^{-4}, batch size 8) on its 100 fitting rows, with the loss computed on response tokens only.

D.2.2 EFFICIENT BANK CONSTRUCTION

Image. Because the classifier is frozen, we compute and cache its image features once. Each target is then fitted directly in feature space, optimizing only the rank-32 factors on the task's 100 fitting examples, without re-running the backbone. Optimization settings are listed in Appendix B.

Text. Text adapters sit inside the language model, so every fitting step requires a full forward and backward pass through Qwen3-1.7B. We therefore share one backbone across N adapters trained in the same process. Their LoRA factors are stored as a stacked $[N, r, d]$ tensor at each adapted layer, and a batch mixes rows from the N tasks, with row i passing only through its own adapter $g[i]$. The loss is the sum of the per-adapter mean cross-entropies and the optimizer is element-wise, so each adapter receives exactly the update it would receive if trained alone. With $N{=}4$ ($N{=}8$ for short commonsense rows), a single GPU fits an adapter in an effective 5–12 s, depending on row length. Rows are tokenized once, batches are length-sorted to reduce padding, and seeds are derived from task IDs, so the build is deterministic and resumable across GPUs and shards.

D.2.3 TARGET-ADAPTER QUALITY

Image targets. Table 13 reports target quality over all 100,000 tasks of each bank. The targets fit their own training examples almost perfectly (98.4–100%), so they provide reliable supervision for the generator. Their query accuracy shows how much headroom is available. Gains are large on CIFAR-100 for all backbones (7.7–14.7 pp) and for ViT-S/16 on Tiny-ImageNet (about 15 pp). They are much smaller for EfficientNet-V2-S and ConvNeXt-V2 Tiny on Tiny-ImageNet (3.5–4.1 pp), indicating a stronger dependence on the frozen representation. The seen-180 bank closely matches the all-200 bank.

Table 13: Task-specific target-LoRA results on CIFAR-100 and Tiny-ImageNet over 100,000 sampled 10-way tasks for each backbone–dataset pair. Tiny-ImageNet (all 200) samples tasks from the full 200-class pool. Tiny-ImageNet (seen-180) uses the seen-180 target pool; the remaining 20 class identities are excluded from both target-adapter and generator training.

Dataset	Backbone	Frozen query	Target-fit accuracy	Target-LoRA query
CIFAR-100	ViT-S/16	81.81%	99.92%	**96.51%**
	EfficientNet-V2-S	**87.57%**	99.87%	95.30%
	ConvNeXt-V2 Tiny	81.86%	98.42%	96.03%
Tiny-ImageNet (all 200)	ViT-S/16	80.00%	99.87%	**95.49%**
	EfficientNet-V2-S	79.25%	**99.96%**	82.72%
	ConvNeXt-V2 Tiny	**81.43%**	99.92%	85.51%
Tiny-ImageNet (seen-180)	ViT-S/16	80.27%	99.87%	**95.61%**
	EfficientNet-V2-S	79.73%	**99.96%**	83.20%
	ConvNeXt-V2 Tiny	**81.97%**	99.92%	85.93%

Text targets on test tasks. Table 14 reports target quality on the 200 test tasks per dataset. Targets fit their own rows almost perfectly, except on GSM8K (90.88%) and MATH (70.15%). Transfer to held-out query rows varies by dataset:

- *Clear gains* on e-SNLI (+28.5), WikiSQL (+19.6), CommonGen (+15.8 BLEU-4), KodCode-Contest (+8.9), and Spider (+5.9).
- *Small changes* (within 2.4 points) on ART, PIQA, ECQA, gretel, KodCode-Filter, and KodCode-Leetcode.
- *Drops* on all three math datasets: GSM8K (−24.7), MATH (−13.7), and MetaMathQA (−4.0).

The text bank therefore provides strong supervision for some datasets but weak, or even harmful, targets for math. This matches the Full SFT behavior in Table 12.

Text bank quality on fitting rows. Table 15 scores bank adapters on the rows they were trained on, with percentiles showing how consistent fitting is across adapters. The first row reports the mixed-pool commonsense bank; the remaining rows report single-dataset banks. Adapters improve strongly over the base model on every dataset, and the median adapter scores at least 94 everywhere except MATH (73). MATH is the hardest to fit, with 99.9% of adapters below 90. GSM8K and Spider also have many adapters below 95 (60.6% and 35.6%).

Table 14: Task-specific target-LoRA results on the 200 test tasks per dataset. Each task has 100 fitting rows and 100 disjoint query rows from the same dataset's test pool; the target LoRA is trained on the fitting rows. Base query is the un-adapted model on the query rows; Target-fit is the target LoRA on its own fitting rows; Target-LoRA query is the target LoRA on the query rows.

Domain	Dataset	Base query	Target-fit accuracy	Target-LoRA query
Commonsense	ART	68.71%	99.92%	**69.28%**
	PIQA	70.05%	100.00%	**72.42%**
	ECQA	**65.82%**	100.00%	65.55%
	e-SNLI	56.16%	99.06%	**84.64%**
	CommonGen	15.04	97.65	**30.88**
Math	GSM8K	**75.47%**	90.88%	50.74%
	MATH	**37.80%**	70.15%	24.14%
	MetaMathQA	**79.17%**	98.67%	75.19%
SQL	gretel	54.10%	98.75%	**55.33%**
	Spider	38.73%	94.52%	**44.61%**
	WikiSQL	53.25%	99.75%	**72.85%**
Python	KodCode-Filter	22.22%	96.50%	**24.52%**
	KodCode-Leetcode	**16.31%**	97.55%	14.53%
	KodCode-Contest	25.50%	98.90%	**34.37%**

Table 15: Bank quality on the adapters' own training rows. Each adapter is decoded on the 100 rows it was trained on; Base is the un-adapted model on the same rows; Gain is Adapter minus Base. P10/P50/P90 are percentiles of the per-adapter score; the last two columns are the fraction of adapters scoring below 90 and below 95.

Bank	Dataset	#Adapters	Metric	Base	Adapter	Gain	P10	P50	P90	%<90	%<95
Mixed-pool bank											
Mixed-pool	Commonsense	10,000	ROUGE-L	13.44	98.78	+85.33	97.1	99.2	100.0	0.12	2.49
Single-dataset bank											
Single-dataset	ART	4,000	Accuracy	82.86	100.00	+17.14	100.0	100.0	100.0	0.00	0.00
Single-dataset	PIQA	4,000	Accuracy	70.23	100.00	+29.77	100.0	100.0	100.0	0.00	0.00
Single-dataset	ECQA	4,000	Accuracy	66.74	100.00	+33.26	100.0	100.0	100.0	0.00	0.00
Single-dataset	e-SNLI	4,000	Accuracy	53.49	98.74	+45.25	97.0	99.0	100.0	1.35	4.47
Single-dataset	CommonGen	4,000	BLEU-4	3.83	97.55	+93.72	93.3	99.3	100.0	6.30	14.80
Single-dataset	GSM8K	4,000	Pass@1	79.86	93.38	+13.52	89.0	94.0	97.0	10.78	60.62
Single-dataset	MATH	4,000	Pass@1	40.24	72.00	+31.76	63.0	73.0	81.0	99.90	100.00
Single-dataset	MetaMathQA	4,000	Pass@1	77.42	98.47	+21.05	97.0	99.0	100.0	0.07	1.62
Single-dataset	gretel	4,000	Pass@1	55.33	98.90	+43.57	97.0	100.0	100.0	0.47	3.62
Single-dataset	Spider	4,000	Pass@1	40.16	95.12	+54.97	90.0	96.0	99.0	7.88	35.60
Single-dataset	WikiSQL	4,000	Pass@1	53.87	99.70	+45.83	99.0	100.0	100.0	0.07	0.35
Single-dataset	KodCode-Filter	4,000	Pass@1	21.42	97.09	+75.67	95.0	97.0	99.0	0.27	9.28
Single-dataset	KodCode-Leetcode	4,000	Pass@1	18.06	97.32	+79.26	95.0	98.0	100.0	0.27	8.70
Single-dataset	KodCode-Contest	4,000	Pass@1	24.84	98.78	+73.94	97.0	99.0	100.0	0.03	0.75

D.3 Step 2 and 3: Canonical Targets, Task Conditions, and Generator Training

These stages produce no separate accuracy results. Here we summarize what is computed once and cached, and point to the training diagnostics.

Canonical targets. Each fitted adapter is SVD-canonicalized once (Section 4.1) and cached with the bank. Text targets are additionally aligned to a fixed reference. Normalization statistics are computed only on the generator-training split.

Task conditions. Image support conditions reuse the frozen features cached in Stage 1. For text, the frozen base model reads each support row as "Question: x / Answer: y". We keep the hidden states after the answer marker and at the end of the sequence as a 4,096-dimensional row feature. These features are computed once for all 561k training-pool rows and for the test pools. As a result, neither generator training nor generation requires a backbone forward pass for conditioning.

Generator training. Canonical-vector dimensions, selected epochs, and flow losses are reported in Tables 8 and 9 (Appendix C). In all settings, the gap between training and validation flow loss at the selected checkpoint is small.

D.4 Step 4: Generated Adapters and Support-Based Selection

Step 4 generates 16 candidate adapters per task with 96 Euler steps and selects one using support examples only. This subsection covers four sets of results:

- the full image-classification results behind the main-paper tables;
- exact-task full-gradient references;
- a diagnostic on held-out classes;
- the effect of bank size for both modalities.

D.4.1 Evaluation Protocols and Metrics

Random tasks. Ten classes are sampled from the permitted class pool. Support and fitting examples come from the training split; query examples come from the official evaluation split.

Novel tasks. Novel tasks are sampled like random tasks, except that a set of ten classes is rejected if it appears in the 100K target-task manifest. Individual classes were seen during training, but the exact ten-class combination was not. In the compact 25-task experiments, the random and novel rows are identical. Under the fixed seed, every sampled combination was already absent from the manifest, so these rows satisfy both definitions and are not independent samples. The paired 100-task analysis in Appendix G instead samples random and novel tasks separately, with novel tasks obtained by explicit rejection.

Official tasks. Support and query examples both come from the official labeled evaluation split (test for CIFAR-100, validation for Tiny-ImageNet), with disjoint images within each task. Generation does not fit a target adapter for these tasks; the Target LoRA values reported for them are separately optimized references.

Metrics. *Frozen* is query accuracy before adaptation. *Generated mean* averages query accuracy over all 16 candidates. *Support-selected*, the deployable result, uses the candidate with the lowest support loss (images) or highest support score (text). *Best-of-16* uses query labels to pick the best candidate and is only an oracle diagnostic. *Gain* is support-selected minus frozen accuracy, in percentage points. *Target LoRA* is a separately optimized task adapter; for random tasks, it is fitted for 50 epochs. *Full SFT* evaluates the full-gradient classifier from Appendix D.1 on the same task examples.

D.4.2 Main Image-Classification Results

Table 16 reports results on 100 random and 100 official tasks for every dataset–backbone pair:

Table 16: Complete evaluation of generated LoRA adapters on CIFAR-100 and Tiny-ImageNet. Results are averaged over 100 sampled 10-way tasks for both the random and official evaluation settings. All accuracy values are reported in percent. **Gain** denotes the improvement of the support-selected generated adapter over the frozen backbone in percentage points. **Best-of-16** reports the highest query accuracy among the 16 generated adapters for each task.

Dataset	Backbone	Evaluation	Frozen	Gen. mean	Support-selected	Gain (pp)	Best-of-16	Target LoRA	Full SFT
CIFAR-100	ViT-S/16	Random	81.71	93.86	**95.15**	13.44	96.81	96.61	91.22
		Official (test)	81.64	93.68	**95.08**	13.44	96.52	95.56	91.51
	EfficientNet-V2-S	Random	87.61	94.57	**94.54**	6.93	95.85	96.27	90.66
		Official (test)	86.91	94.25	**94.28**	7.37	95.64	93.78	90.34
	ConvNeXt-V2 Tiny	Random	81.36	92.31	**94.07**	12.71	96.03	96.22	84.11
		Official (test)	81.05	91.97	**93.43**	12.38	95.87	92.96	84.00
Tiny-ImageNet	ViT-S/16	Random	79.39	84.14	**89.99**	10.60	91.64	95.24	87.78
		Official (val)	80.12	83.61	**90.04**	9.92	91.68	94.73	87.58
	EfficientNet-V2-S	Random	79.29	82.80	**82.86**	3.57	83.48	85.31	79.30
		Official (val)	79.52	82.17	**82.21**	2.69	83.19	84.02	79.52
	ConvNeXt-V2 Tiny	Random	81.47	84.99	**85.07**	3.60	85.54	86.37	87.17
		Official (val)	81.17	85.02	**85.15**	3.98	85.58	88.15	87.73

- The support-selected adapter improves over the frozen backbone in all twelve rows, with larger gains on CIFAR-100 (6.9–13.4 pp) than on Tiny-ImageNet (2.7–10.6 pp). These gains follow the target headroom in Table 13.
- Random and official results agree closely, so the gains do not depend on the split the support images come from.
- On official CIFAR-100 tasks, the support-selected adapter outperforms the separately optimized Target LoRA for EfficientNet-V2-S and ConvNeXt-V2 Tiny.
- It also outperforms Full SFT in every setting except Tiny-ImageNet with ConvNeXt-V2 Tiny.

D.4.3 Exact-Task Full-Gradient References

Table 17 evaluates the best checkpoint of each full-gradient classifier on exactly the same class sets and query images used in the corresponding generator evaluation. The official column matches the Full SFT values in Table 16. The random/novel column is paired with its own task manifest, so it should not be substituted for the 100-task random Full SFT values in that table. The two columns differ by at most 0.62 pp.

Table 17: Exact-task full-gradient reference accuracies on the same 10-way task and query indices used by the corresponding generator evaluations.

Dataset	Backbone	Random/novel best checkpoint	Official best checkpoint
CIFAR-100	ViT-S/16	91.06	91.51
CIFAR-100	EfficientNet-V2-S	90.96	90.34
CIFAR-100	ConvNeXt-V2 Tiny	83.94	84.00
Tiny-ImageNet	ViT-S/16	87.96	87.58
Tiny-ImageNet	EfficientNet-V2-S	79.24	79.52
Tiny-ImageNet	ConvNeXt-V2 Tiny	87.12	87.73

E Additional Results & Experiments

E.1 LoRA Bank-Size Ablations

This ablation varies the size of the Stage 1 bank and measures the effect on Stage 4 performance. Table 18 compares banks of 1,000, 10,000, and 100,000 image adapters on 100 official tasks. Table 19 repeats the study on the five commonsense text datasets with 200 to 20,000 adapters per dataset.

Image. Support-selected accuracy increases monotonically with bank size in all six settings. The largest jumps come from ViT-S/16 on Tiny-ImageNet (79.92 to 90.04) and ConvNeXt-V2 Tiny on CIFAR-100 (82.05 to 93.43). With only 1,000 adapters, the generator is at or below the frozen backbone for EfficientNet-V2-S and ViT-S/16 on Tiny-ImageNet. A large bank is therefore essential for the gains in Table 16.

Text. The largest improvement comes from moving from 200 to 1,000 adapters per dataset; for example, e-SNLI rises from 69.74 to 87.50. Beyond that point, gains are small (at most about one point), and CommonGen peaks at 1,000 adapters. Text performance therefore saturates at much smaller bank sizes than image performance.

E.1.1 Image-Classification Bank-Size Ablations

Table 18: Effect of target-bank size on image-classification performance. All results are mean 10-way query accuracies (%) over 100 official tasks, using the CIFAR-100 test split or the Tiny-ImageNet validation split. Each task uses one support image and 20 disjoint query images per class, with 16 generated adapters and 96 flow steps per adapter. Support-selected chooses the adapter with the lowest support loss; Best-of-16 uses query labels and is an oracle diagnostic. The highest accuracy in each row is bolded and highlighted.

Model	Targets	Frozen	Generated mean	Support-selected	Best-of-16
CIFAR-100					
EfficientNet-V2-S	1,000	86.91	87.38	87.77	**88.19**
EfficientNet-V2-S	10,000	86.91	91.25	91.64	**92.67**
EfficientNet-V2-S	100,000	86.91	94.25	94.28	**95.64**
ConvNeXt-V2 Tiny	1,000	81.05	79.89	82.05	**85.46**
ConvNeXt-V2 Tiny	10,000	81.05	80.57	89.46	**91.30**
ConvNeXt-V2 Tiny	100,000	81.05	91.97	93.43	**95.87**
ViT-S/16	1,000	81.64	79.99	83.03	**84.40**
ViT-S/16	10,000	81.64	84.58	89.84	**91.47**
ViT-S/16	100,000	81.64	93.68	95.08	**96.52**
Tiny-ImageNet					
EfficientNet-V2-S	1,000	79.52	79.36	79.37	**79.93**
EfficientNet-V2-S	10,000	79.52	81.98	81.98	**82.69**
EfficientNet-V2-S	100,000	79.52	82.17	82.21	**83.19**
ConvNeXt-V2 Tiny	1,000	81.17	82.11	82.16	**82.52**
ConvNeXt-V2 Tiny	10,000	81.17	84.52	84.58	**85.05**
ConvNeXt-V2 Tiny	100,000	81.17	85.02	85.15	**85.58**
ViT-S/16	1,000	80.12	74.90	79.92	**81.97**
ViT-S/16	10,000	80.12	75.28	81.59	**83.79**
ViT-S/16	100,000	80.12	83.61	90.04	**91.68**

E.1.2 TEXT-GENERATION BANK-SIZE ABLATIONS

Table 19: Effect of bank size on text performance (commonsense domain, single-dataset banks). All results are mean query scores over 200 test tasks per dataset (accuracy in %; BLEU-4 for CommonGen), using the official test split of each dataset. Each task uses 10 support rows and 100 disjoint query rows, with 16 generated adapters and 96 flow steps per adapter. Targets is the number of LoRAs per dataset (total bank size in parentheses); one generator is trained per bank size on all five datasets. Support-selected chooses the adapter with the highest support score; Best-of-16 uses query labels and is an oracle diagnostic. The highest score in each row is bolded and highlighted.

Dataset	Targets	Frozen	Generated mean	Support-selected	Best-of-16
ART	200 (1K)	68.71	68.91	68.81	**71.11**
ART	1K (5K)	68.71	71.04	71.12	**73.57**
ART	5K (25K)	68.71	71.38	71.33	**73.93**
ART	10K (50K)	68.71	71.43	71.58	**74.03**
ART	20K (100K)	68.71	71.47	71.66	**74.17**
PIQA	200 (1K)	70.05	71.60	72.01	**74.11**
PIQA	1K (5K)	70.05	74.21	74.10	**77.05**
PIQA	5K (25K)	70.05	74.76	74.69	**77.56**
PIQA	10K (50K)	70.05	74.87	74.80	**77.55**
PIQA	20K (100K)	70.05	74.98	75.16	**77.67**
ECQA	200 (1K)	65.82	65.50	65.47	**67.58**
ECQA	1K (5K)	65.82	67.67	67.62	**70.25**
ECQA	5K (25K)	65.82	68.30	68.42	**70.94**
ECQA	10K (50K)	65.82	68.44	68.56	**71.12**
ECQA	20K (100K)	65.82	68.59	68.67	**71.34**
e-SNLI	200 (1K)	56.16	63.32	69.74	**71.01**
e-SNLI	1K (5K)	56.16	87.53	87.50	**89.70**
e-SNLI	5K (25K)	56.16	87.75	87.81	**89.84**
e-SNLI	10K (50K)	56.16	87.77	87.84	**89.89**
e-SNLI	20K (100K)	56.16	87.82	87.88	**89.97**
CommonGen	200 (1K)	15.04	15.63	16.93	**17.61**
CommonGen	1K (5K)	15.04	22.93	24.21	**25.84**
CommonGen	5K (25K)	15.04	22.90	24.01	**25.81**
CommonGen	10K (50K)	15.04	22.86	23.79	**25.75**
CommonGen	20K (100K)	15.04	22.96	23.91	**25.93**

F COMPUTATIONAL COST OF ADAPTATION

STAGE replaces deployment-time gradient-based adapter fitting with support-conditioned generation and selection. Tables 20 and 21 distinguish per-task adaptation costs from the shared offline costs of adapter-bank construction and generator training. Generation uses 10 labeled support examples, compared with the 100 examples used to fit each target adapter. All generation configurations use $N = 16$ candidates and 96 flow steps unless otherwise specified.

To compare costs, we define the time and FLOP ratios as

$$R_T = \frac{T_{\text{fit}}}{T_{\text{STAGE}}}, \qquad R_F = \frac{F_{\text{fit}}}{F_{\text{STAGE}}},$$

where T denotes adaptation time and F denotes FLOPs. Values above one favor STAGE; values below one favor gradient-based fitting. These ratios describe the reported configurations rather than a universal advantage over PEFT.

At the default sampling budget, generation has higher reported time and FLOPs than target fitting. For images, fitting uses cached features and is amortized over 512-task batches, whereas generation is timed per task on unmatched hardware. The image time ratios are therefore descriptive cost ratios rather than controlled latency comparisons. For text, the costs below account for 16 candidates in every domain; the Math and Python generation totals are recalculated from the stated per-candidate costs.

The strongest compute reduction comes from the image sampling-step ablation. Reducing the number of flow steps from 96 to four lowers flow-sampling FLOPs by $\mathbf{24}\times$, with at most a $\mathbf{0.44}$-percentage-point decrease in support-selected accuracy. This reduction applies to sampling computation, rather

than total adaptation latency, because conditioning and candidate selection introduce additional costs. Thus, STAGE supports a favorable quality–sampling-compute trade-off while eliminating deployment-time gradient updates. Offline bank construction and generator training remain shared costs, and whole-dataset full-gradient training and SFT are reported as separate references.

Table 20: **Compute cost for image-classification adaptation.** Per-task fitting and generation costs, shared offline training costs, and the full-gradient classifier reference. Generation uses $N = 16$ candidates and 96 flow steps. $R_T = T_{\text{fit}}/T_{\text{STAGE}}$ and $R_F = F_{\text{fit}}/F_{\text{STAGE}}$; values above one favor STAGE.

Backbone	Fitted LoRA			STAGE			Fit / STAGE		Full-grad.
	Fit s/task	Fit GFLOPs/task	Bank GPU-h	Gen. s/task	Gen. GFLOPs/task	Generator GPU-h	Time R_T	FLOPs R_F	GPU-h
CIFAR-100									
EfficientNet-V2-S	0.0153	2.60	0.43	0.592	6,281	5.32	0.0258	0.000414	3.72
ConvNeXt-V2 Tiny	0.0126	1.60	0.35	0.999	5,868	5.23	0.0126	0.000273	7.95
ViT-S/16	0.0144	0.85	0.40	1.642	10,169	3.01	0.0088	0.000084	0.21
Tiny-ImageNet									
EfficientNet-V2-S	0.0145	3.98	0.40	0.597	6,362	5.45	0.0243	0.000626	5.00
ConvNeXt-V2 Tiny	0.0172	2.47	0.48	1.021	5,949	5.15	0.0168	0.000415	7.61
ViT-S/16	0.0156	1.33	0.43	0.980	5,639	2.74	0.0159	0.000236	1.53

Fitting uses cached backbone features and 512-task batches; generation is timed per task. Timings were collected on NVIDIA H200 and A100 80GB GPUs without matching hardware between fitting and generation. Consequently, R_T does not represent a controlled wall-clock speedup. Bank construction and generator training are separate offline costs; the full-gradient classifier is shared across tasks. FLOPs are analytic operation counts.

Table 21: **Compute cost for text-generation adaptation.** Per-task fitting and generation costs across domains, with $N = 16$ candidates and 96 flow steps throughout. Time and FLOP ratios use the reported domain means. Values above one favor STAGE. Full-parameter SFT is a separate whole-dataset reference.

Domain (#datasets)	Fitted LoRA			STAGE			Fit / STAGE		Full SFT
	Fit s/task	Fit GFLOPs/task	Bank GPU-h	Gen. s/task	Gen. GFLOPs/task	Generator GPU-h	Time R_T	FLOPs R_F	GPU-h
Commonsense (5)	6.5 ± 1.0	$450{,}000 \pm 70{,}000$	90.5	62.4	2,400,000	27.3	0.104	0.188	0.6 ± 0.4
Math (3)	10.6 ± 0.8	$1{,}260{,}000 \pm 90{,}000$	88.2	62.4†	2,400,000†	27.3	0.170†	0.525†	1.1 ± 1.5
SQL (3)	6.3 ± 1.8	$810{,}000 \pm 220{,}000$	52.8	62.4	2,400,000	27.3	0.101	0.338	0.9 ± 0.5
Python (3)	11.1 ± 1.0	$1{,}530{,}000 \pm 220{,}000$	92.1	62.4†	2,400,000†	27.3	0.178†	0.638†	0.6 ± 0.4

† Calculated totals for 16 candidates, using the stated per-candidate costs of 3.9 seconds and 150,000 GFLOPs. The time estimates assume linear scaling and are not independent measurements of 16-candidate latency. Reported timings use NVIDIA H100 NVL GPUs. Fitting uses 100 examples per task; generation is conditioned on 10 support examples. The 27.3 GPU-h generator-training cost is shared across all domains and must not be summed over rows. Bank GPU-h values retain the supplied accounting for 10,000 adapters per dataset; they are not costs for a 1,000-adapter bank. Full SFT uses three epochs on each dataset's training pool. FLOPs are analytic operation counts.

Table 22: **Image sampling compute versus adaptation quality.** Both configurations generate 16 candidates per task. FLOP reduction is relative to 96-step STAGE and counts flow-sampling computation only. Accuracy loss is measured in percentage points.

Method	Steps	Relative sampling FLOPs	FLOP reduction	Maximum accuracy loss (pp)
STAGE	96	1	$1\times$	0
STAGE	4	$1/24$	$\mathbf{24}\times$	≤ 0.44

G Additional Similarity Diagnostics

The main paper uses effective-update similarity to study task sensitivity and the relationship between numerical target recovery and functional performance. Raw LoRA factors are not compared because they are non-identifiable.

For a generated effective update ΔW_{gen} and the directly optimized target update ΔW_{target}, cosine similarity is

$$\text{Cos} = \frac{\langle \Delta W_{\text{gen}}, \Delta W_{\text{target}} \rangle}{\|\Delta W_{\text{gen}}\|_2 \, \|\Delta W_{\text{target}}\|_2}, \tag{4}$$

and relative L_2 error is

$$\text{RelL2} = \frac{\|\Delta W_{\text{gen}} - \Delta W_{\text{target}}\|_2}{\|\Delta W_{\text{target}}\|_2}. \tag{5}$$

G.1 Extended Generated-Adapter Similarity Matrices

Figures 8 and 9 extend the generated-adapter comparisons in the main paper. The image matrices show greater similarity between independent samples from the same task than between different tasks. The text matrices exhibit clearer dataset-level structure, while differences between tasks within the same dataset are less pronounced.

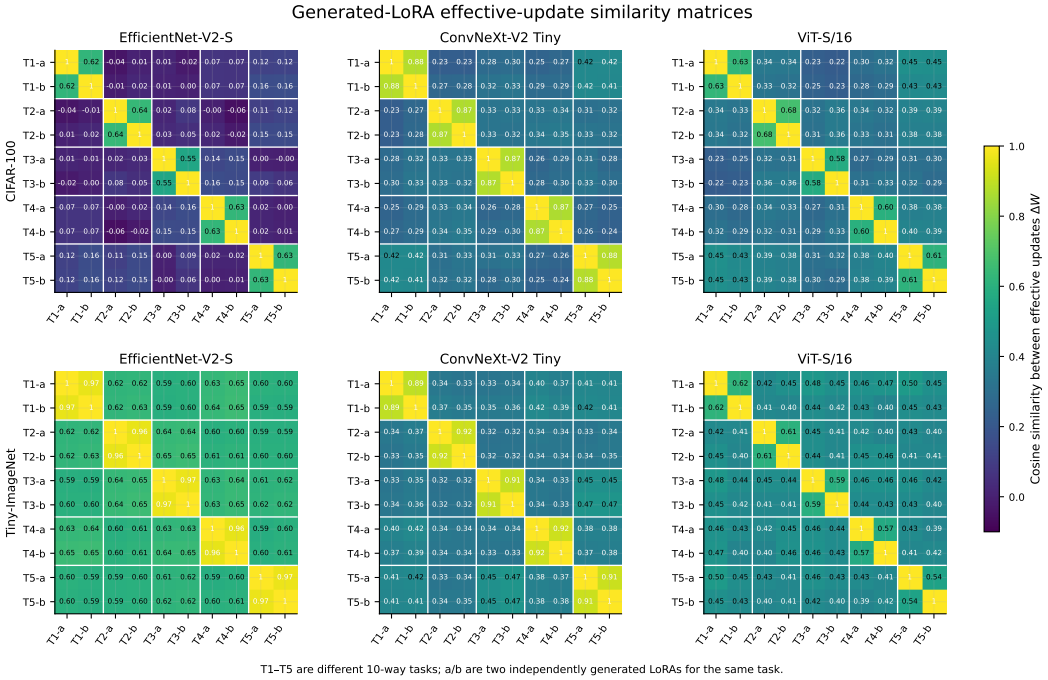

Figure 8: **Extended similarity analysis of generated image adapters.** Pairwise cosine similarity between effective LoRA updates $\Delta W = (\alpha/r)BA$ for five 10-way tasks (T1–T5), with two independently generated adapters (a, b) per task. Rows correspond to CIFAR-100 and Tiny-ImageNet; columns correspond to EfficientNet-V2-S, ConvNeXt-V2 Tiny, and ViT-S/16. Off-diagonal entries within each 2×2 diagonal block compare samples from the same task; entries outside these blocks compare different tasks.

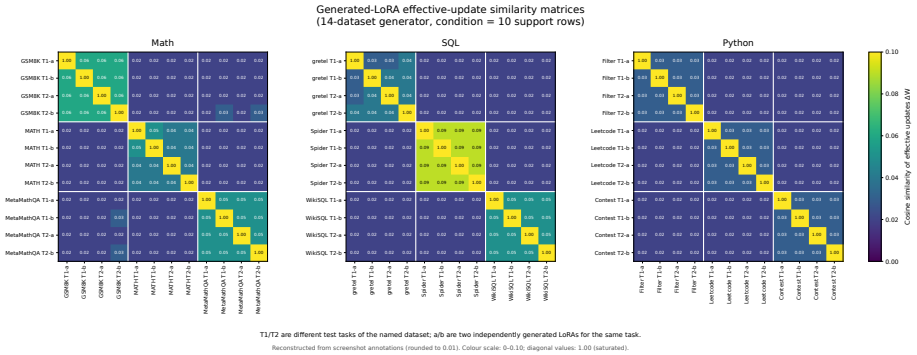

Figure 9: **Extended similarity analysis of generated text adapters.** Pairwise cosine similarity between effective LoRA updates across nine mathematics, SQL, and Python datasets. Each dataset contributes two test tasks (T1, T2), each conditioned on 10 support examples, with two independently generated adapters (a, b) per task. White boundaries separate datasets. Within each dataset block, entries compare both same-task and different-task samples. The color scale spans 0–0.1 to reveal off-diagonal variation; diagonal self-similarities equal 1.

G.2 Five-Task Diagnostic

The compact diagnostic uses five tasks and two independently generated LoRAs per task. Same-task cosine averages the five within-task pairs, whereas different-task cosine compares generated adapters conditioned on different tasks.

Table 23: Effective-update similarity in the five-task diagnostic. Higher cosine is better, while lower relative L_2 indicates closer agreement with the directly optimized target.

Dataset / Backbone	Target cosine ↑	Relative L_2 ↓	Same-task cosine ↑	Different-task cosine
CIFAR-100 / ViT-S/16	0.594	0.896	0.617	0.334
CIFAR-100 / EfficientNet-V2-S	0.565	0.967	0.613	0.057
CIFAR-100 / ConvNeXt-V2 Tiny	0.866	0.520	0.872	0.308
Tiny-ImageNet / ViT-S/16	0.585	0.904	0.585	0.433
Tiny-ImageNet / EfficientNet-V2-S	**0.962**	**0.276**	0.965	0.615
Tiny-ImageNet / ConvNeXt-V2 Tiny	0.897	0.459	0.908	0.367

Across all six model–dataset pairs, same-task cosine exceeds different-task cosine. This supports the interpretation that independently generated adapters respond to the conditioning task rather than collapsing to a single task-independent update.

G.3 Paired 100-Task Similarity–Accuracy Evaluation

For every model and protocol, we evaluate 100 tasks with 16 independently generated adapters per task. Each generated adapter is paired with the directly optimized target for the same task and evaluated on the same query examples, yielding

$$100 \times 16 = 1{,}600$$

paired comparisons per model and protocol.

The paired analysis shows that numerical target recovery and functional recovery are related but are not equivalent. For example, CIFAR-100 ViT-S/16 achieves only moderate generated-to-target cosine similarity (0.577 for random tasks and 0.574 for novel tasks) while producing the largest mean accuracy gains, 12.15 and 12.05 percentage points, respectively. We therefore treat similarity to a particular supervised target as a diagnostic rather than as the definition of successful adapter generation.

Table 24: Paired generated-to-target evaluation over 100 tasks per setting. Each task has 16 generated candidates, each compared with the LoRA fitted on the same task (target cosine ↑, relative L_2 ↓), giving 1,600 generated-to-target comparisons per model and setting. Query performance is the mean 10-way query accuracy (%) over the 100 tasks. Mean gain is Generated mean minus Frozen. Novel tasks use ten-class combinations absent from the 100K target bank.

			LoRA similarity		Query performance					
Dataset	**Backbone**	**Evaluation**	**Target cosine ↑**	**Relative L_2 ↓**	**Frozen**	**Generated mean**	**Mean gain**	**Support-selected**	**Best of 16**	**Direct target**
CIFAR-100	ViT-S/16	Random	0.577	0.910	82.01	94.16	+12.15	95.40	96.94	96.81
		Novel	0.574	0.914	81.66	93.71	+12.05	94.96	96.59	96.43
	EfficientNet-V2-S	Random	0.498	1.004	88.14	94.66	+6.52	94.86	96.02	95.85
		Novel	0.504	0.982	87.63	94.47	+6.84	94.60	95.76	95.72
	ConvNeXt-V2 Tiny	Random	0.851	0.548	82.35	92.41	+10.06	94.30	96.16	96.13
		Novel	0.847	0.557	81.87	91.83	+9.96	93.31	95.73	95.90
Tiny-ImageNet	ViT-S/16	Random	0.595	0.894	80.36	84.61	+4.25	89.93	92.28	95.65
		Novel	0.599	0.887	80.69	84.91	+4.22	90.37	92.36	95.78
	EfficientNet-V2-S	Random	0.457	1.443	79.29	82.05	+2.76	82.07	83.06	83.06
		Novel	0.450	1.465	79.50	82.45	+2.95	82.49	83.43	83.08
	ConvNeXt-V2 Tiny	Random	0.646	0.870	81.45	85.01	+3.56	85.06	85.50	86.25
		Novel	0.646	0.870	81.70	85.28	+3.58	85.42	85.83	86.45

Table 25: Paired generated-to-target evaluation over 200 test tasks per dataset. Each task has N generated candidates, each compared with the LoRA fitted on the same task (target cosine ↑, relative L_2 ↓ in ΔW space). Query performance is the mean over the 200 tasks' query rows (accuracy in %; BLEU-4 for CommonGen; execution accuracy for SQL; pass rate for Python). Mean gain is Generated mean minus Frozen. †Math and SQL performance columns are pending re-decoding with the corrected output budget.

			LoRA similarity		Query performance					
Domain	**Dataset**	N	**Target cosine ↑**	**Relative L_2 ↓**	**Frozen**	**Generated mean**	**Mean gain**	**Support-selected**	**Best of N**	**Direct target**
Commonsense	ART	16	0.052	1.486	68.71	71.48	+2.77	71.61	74.17	69.28
	PIQA	16	0.058	1.537	70.05	74.98	+4.93	75.14	77.67	72.42
	ECQA	16	0.016	1.593	65.82	68.59	+2.77	68.67	71.35	65.55
	e-SNLI	16	0.068	1.351	56.16	87.82	+31.66	87.84	89.98	84.64
	CommonGen	16	0.032	1.353	15.04	22.96	+7.92	23.91	26.09	30.88
Math†	GSM8K	6	0.058	1.363	75.47	60.48	−14.99	60.80	65.31	50.74
	MATH	6	0.041	1.362	37.80	32.70	−5.10	33.17	37.25	24.14
	MetaMathQA	6	0.052	1.473	79.17	80.20	+1.03	80.34	84.00	75.19
SQL†	gretel	16	0.041	1.516	54.10	58.70	+4.60	59.02	62.49	55.33
	Spider	16	0.082	1.503	38.73	46.16	+7.43	46.84	52.01	44.61
	WikiSQL	16	0.065	1.575	53.25	74.91	+21.66	75.16	77.82	72.85
Python	KodCode-Filter	6	0.030	1.456	22.22	27.23	+5.01	27.34	30.36	24.52
	KodCode-Leetcode	6	0.032	1.455	16.31	16.94	+0.63	17.02	20.16	14.53
	KodCode-Contest	6	0.033	1.485	25.50	38.83	+13.33	38.96	42.03	34.37