NONLINEARITY AS RANK: GENERATIVE LOW-RANK ADAPTERS WITH RADIAL BASIS FUNCTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-rank adaptation (LoRA) approximates the update of a pretrained weight matrix using the product of two low-rank matrices. However, standard LoRA follows an **explicit-rank** paradigm, where increasing model capacity requires adding more rows or columns (i.e., **basis vectors**) to the low-rank matrices, leading to substantial parameter growth. In this paper, we find that these basis vectors exhibit significant parameter redundancy and can be compactly represented by lightweight nonlinear functions. Therefore, we propose **Generative Low-Rank Adapter (GenLoRA)**, which replaces explicit basis vector storage with nonlinear basis vector generation. Specifically, GenLoRA maintains a latent vector for each low-rank matrix and employs a set of lightweight radial basis functions (RBFs) to synthesize the basis vectors. Each RBF requires far fewer parameters than an explicit basis vector, enabling higher parameter efficiency in GenLoRA. Extensive experiments across multiple datasets and architectures show that GenLoRA attains higher effective LoRA ranks under smaller parameter budgets, resulting in superior fine-tuning performance. Our code is publicly available at `https://anonymous.4open.science/r/GenLoRA`.

1 INTRODUCTION

Large language models (LLMs) have demonstrated unprecedented capabilities across a wide range of tasks (Dubey et al., 2024; DeepSeek et al., 2024; Cai et al., 2024). To adapt these general-purpose models to specific downstream applications, fine-tuning is essential. However, as model sizes explode, full fine-tuning becomes prohibitive in both computation and memory. Parameter-efficient fine-tuning (PEFT) methods (Lester et al., 2021; Houlsby et al., 2019) have thus emerged as a standard paradigm, among which low-rank adaptation (LoRA) (Hu et al., 2022) is widely adopted. LoRA approximates the update of a pretrained weight $W \in \mathbb{R}^{m\times n}$ as a low-rank decomposition $\Delta W = BA$, where $B \in \mathbb{R}^{m\times r}$ and $A \in \mathbb{R}^{r\times n}$, with $r \ll \min(m, n)$.

The effectiveness of LoRA is closely tied to the rank r, as a larger rank generally increases the expressive capacity of the adaptation (Zeng & Lee, 2024). However, LoRA follows an explicit-rank paradigm: increasing r requires maintaining additional independent basis vectors, corresponding to the columns of B and the rows of A. In modern LLMs, the projection dimensions m and n are typically on the order of thousands, which makes this expansion costly. Specifically, as the rank increases, the total number of parameters grows linearly with $m + n$, leading to a parameter complexity of $\mathcal{O}(r(m + n))$. This scaling behavior motivates a natural question: ***do these high-dimensional basis vectors contain parameter redundancy that can be exploited for more efficient adaptation?***

To examine the suspected parameter redundancy, we study learnable non-linear functions for basis vector reconstruction. Specifically, given a pretrained LoRA matrix (i.e., the matrix B) containing r basis vectors , we first construct a shared prototype vector by averaging these r basis vectors. We then learn r independent non-linear functions that take this shared prototype as input and reconstruct the original r basis vectors. In this work, we focus on Radial Basis Functions (RBFs) (Buhmann, 2000), which are renowned for their powerful approximation capabilities and high parameter efficiency. Additional details of the experimental setup are provided in the Appendix K.3. As shown in Figure 1 (*Upper*), the results are compelling: the RBFs successfully recover the global shape and essential patterns of the original basis vectors. Most strikingly, this reconstruction requires very few

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.

learnable parameters, amounting to only 6.25% of the parameters needed to explicitly maintain a single basis vector. This finding affirmatively answers our research question: substantial parameter redundancy indeed exists within the explicit basis vectors, as the information spread across them can be effectively compressed into non-linear transformations. In essence, nonlinearity can serve as a parameter-efficient substitute for rank, offering a direct solution to the identified redundancy, which we term "Nonlinearity as Rank".

To address the parameter redundancy described above, we propose **Generative Low-Rank Adapters (GenLoRA)**, which generate basis vectors through nonlinearity rather than storing them as explicit parameters. Specifically, GenLoRA maintains a single learnable latent vector for each low-rank matrix (A or B) and learns r RBF-based nonlinear generators. Each generator maps the shared latent vector to one basis vector, and the resulting r generated vectors directly form the matrices A and B. This design allows the effective rank to scale while keeping the parameter complexity at $\mathcal{O}(m + n + r|\theta|)$, where $|\theta|$ denotes the number of parameters in each generator and typically satisfies $|\theta| \ll \min(m, n)$. As illustrated in Figure 1 (*Lower*), this formulation effectively removes the substantial parameter redundancy in standard LoRA and yields a better accuracy–parameter trade-off. In addition, theoretical analysis shows that GenLoRA preserves the low-rank structure and has locally bounded gradients under explicit regularity conditions.

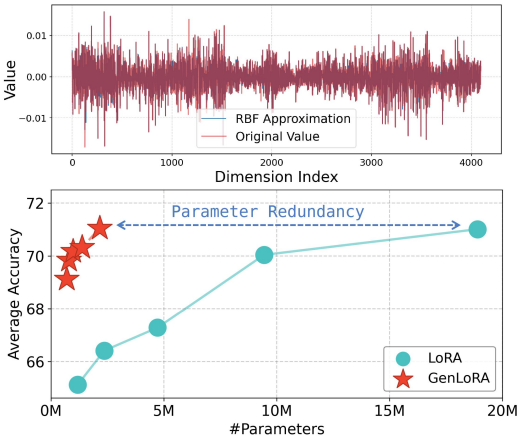


Figure 1: ***(Upper)*** The reconstruction result of the first column vector of a pretrained LoRA matrix B. The trajectories in **hue** illustrate the overlap between the Radial Basis Function (RBF) approximation and the original values. ***(Lower)*** Accuracy–parameter trade-off on mathematical reasoning tasks with LLaMA3-8B. The five points for each method correspond to ranks $r = \{2, 4, 8, 16, 32\}$.

The core contributions of this study are summarized as follows:

- We reveal significant parameter redundancy in LoRA through the basis vector reconstruction experiment. The results show that basis vectors can be effectively represented by non-linear functions, establishing nonlinearity as a parameter-efficient substitute for rank.
- We propose GenLoRA, the first generative LoRA framework that synthesizes low-rank matrices from shared latent vectors via RBF generators. This design mitigates the parameter redundancy, allowing effective rank increase with only minimal parameter cost.
- Theoretically, GenLoRA preserves the low-rank structure and has locally bounded gradients under explicit regularity conditions. Empirically, extensive experiments demonstrate that GenLoRA attains higher effective LoRA ranks under smaller parameter budgets.

2 Related Work

2.1 Low-Rank Adaptation

LoRA (Hu et al., 2022) reduces fine-tuning overhead by factorizing weight updates as $\Delta W = BA$. Subsequent work primarily explores optimization, linear structural modifications, and nonlinear extensions. For optimization, DoRA (Liu et al., 2024a) decouples magnitude and direction updates, LoRA+ (Hayou et al., 2024) adopts differential learning rates, and PiSSA (Meng et al., 2024) and LoRA-GA (Wang et al., 2024) improve initialization via SVD of pretrained weights and sampled gradients, respectively. Linear variants address rank constraints through dynamic rank allocation (AdaLoRA (Zhang et al., 2023)), token-wise projections (TopLoRA (Li et al., 2025b)), or higher-rank constructions using Hadamard products (HiRA (Huang et al., 2025)), Kronecker products (KronA (Edalati et al., 2025)), and block-diagonal stacking (MELoRA (Ren et al., 2024)). NOLA (Koohpayegani et al., 2023) and HyperLoRA (Li et al., 2025a) instead reconstruct weights through linear combinations of random or learned bases. Nonlinear extensions include LoRAN (Li et al., 2024),

SineLoRA (Ji et al., 2024), and AuroRA (Dong et al., 2025). Whereas AuroRA confines nonlinearity to the **low-dimensional activation space**, GenLoRA introduces it into a **high-dimensional parameter generation space**, alleviating the information bottleneck with minimal additional parameters. Moreover, rather than generating mixing coefficients over shared bases as in NOLA and HyperLoRA, GenLoRA directly generates basis matrices, enabling more flexible, layer-specific adaptation.

2.2 Nonlinearity Applications

Nonlinear approximation has extensively employed rational functions (Baker Jr, 1963), trigonometric functions (Edmunds et al., 2012), B-splines (De Boor, 1972), and radial basis functions (RBFs) (Buhmann, 2000). Recent adaptation methods exploit these forms: NoRA (Yin et al., 2025) learns rational activation functions, while SineLoRA (Ji et al., 2024) and FourierFT (Gao et al., 2024) employ sinusoidal modulations and Fourier-based parameterizations, respectively. KAN (Liu et al., 2024b) further shifts learning from linear weights to parameterizable activation functions, a principle adopted by AuroRA (Dong et al., 2025) through KAN-based nonlinear MLP layers. In contrast, GenLoRA uses **learnable nonlinear functions solely for weight generation**, avoiding the overhead of parameter-heavy MLP layers. It further replaces KAN-style B-splines with RBFs to balance approximation capacity and computational efficiency, enabling lightweight high-dimensional parameter generation.

3 Methodology

In this section, we introduce **Generative Low-Rank Adapters (GenLoRA)**, a novel paradigm designed to synthesize high-dimensional basis vectors from a latent vector via lightweight Radial Basis Function (RBF) generators.

3.1 Nonlinearity as Rank

We first revisit LoRA from a generative perspective. As illustrated in Figure 2(a), we characterize standard LoRA as adhering to an explicit-rank paradigm, where the low-rank structure is directly constructed from independent and trainable explicit basis vectors. Given a frozen weight matrix $W_0 \in \mathbb{R}^{m\times n}$, standard LoRA learns a low-rank update:

$$\Delta W = BA = \sum_{i=1}^{r} b_i a_i^\top, \tag{1}$$

where the matrices are composed of explicit basis vectors: $B = [b_1, \ldots, b_r] \in \mathbb{R}^{m\times r}$ and $A = [a_1, \ldots, a_r]^\top \in \mathbb{R}^{r\times n}$. Here, $b_i \in \mathbb{R}^m$ and $a_i \in \mathbb{R}^n$ denote the i-th column of B and the i-th row of A, respectively. This implies that each incremental rank unit incurs a parameter cost of $m + n$, resulting in a total complexity of $\mathcal{O}(r(m + n))$.

We propose to treat nonlinearity as a substitute for explicit basis vectors: instead of introducing a massive number of parameters for storage, we synthesize these vectors using a minimal set of parameters as illustrated in Figure 2(b). Here, "Nonlinearity as Rank refers to using compact nonlinear generators to reduce the parameter cost of constructing additional rank components. Formally, GenLoRA reparameterizes matrices B and A via latent vectors and nonlinear generators. Let $Z_B \in \mathbb{R}^{m\times 1}$ and $Z_A \in \mathbb{R}^{1\times n}$ be two learnable latent vectors. Let $F_B = \{f_1^B, \ldots, f_r^B\}$ and $F_A = \{f_1^A, \ldots, f_r^A\}$ be two sets of lightweight nonlinear generators. Each generator takes the corresponding latent vector Z_B or Z_A as input to synthesize a basis vector:

$$\begin{aligned} b_i &= f_i^B(Z_B; \theta_i^B) \in \mathbb{R}^m, \\ a_i &= f_i^A(Z_A; \theta_i^A) \in \mathbb{R}^n, \end{aligned} \tag{2}$$

where θ_i^B and θ_i^A are the parameters of the nonlinear generators, and we use θ to denote the generator parameters when no distinction is required. Stacking these vectors reconstructs the low-rank matrices:

$$B = [b_1, \ldots, b_r],\ A = [a_1, \ldots, a_r]^\top, \tag{3}$$

resulting in the GenLoRA update:

$$\Delta W_{\text{Gen}} = \sum_{i=1}^{r} f_i^B(Z_B; \theta_i^B)\, f_i^A(Z_A; \theta_i^A)^\top. \tag{4}$$

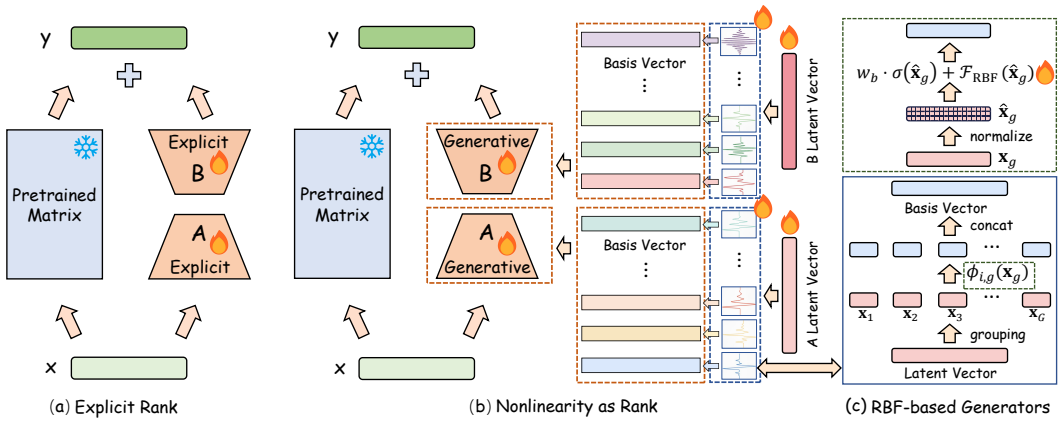


Figure 2: The overall architecture of Generative Low-Rank Adapters (GenLoRA). (a) **Explicit Rank:** Standard LoRA relies on the explicit-rank paradigm, where model capacity is constrained by the linear dimension of basis vectors. (b) **Nonlinearity as Rank:** Our proposed paradigm synthesizes basis vectors from latent vectors via generative functions, effectively reducing the parameter cost. (c) **RBF-based Generators:** The detailed internal workflow of the generator.

Under this parameterization, the number of trainable parameters is primarily determined by the sizes of Z_B, Z_A, and the nonlinear generators. Specifically, the parameter complexity is $\mathcal{O}(m+n+r|\theta|)$, where $|\theta|$ denotes the number of parameters for a pair of f_i^A and f_i^B. Since $|\theta| \ll m, n$, this complexity is substantially lower than the $\mathcal{O}(r(m+n))$ parameter cost of standard LoRA. By exploiting the expressive power of nonlinear mappings from latent vectors to basis vectors, GenLoRA significantly enhances adapter capacity, thereby realizing our core principle of Nonlinearity as Rank.

3.2 RBF-based Functional Generators

Radial basis functions (RBFs) are well known for their strong approximation capability and high parameter efficiency. Motivated by these properties, we instantiate the nonlinear generators f_i^B and f_i^A using RBF-based parameterizations.To further achieve stable and expressive nonlinear mapping, we introduce instance-wise normalization and group-wise decomposition in the generator design, as shown in Figure 2(c).

Group-wise decomposition. Let $z \in \mathbb{R}^d$ denote an input latent vector. To avoid directly high-dimensional mapping, we partition z into G low-dimensional sub-vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_G\}$, where each $\mathbf{x}_g \in \mathbb{R}^{d_g}$ such that $d = G \cdot d_g$. To ensure that the r synthesized basis vectors remain distinct and avoid rank collapse, each matrix is generated using r nonlinear generators, and each generator f_i processes its corresponding sub-vectors using G distinct transformations:

$$f_i(z) = \text{concat}\left(\phi_{i,1}(\mathbf{x}_1), \ldots, \phi_{i,G}(\mathbf{x}_G)\right) \in \mathbb{R}^d, \tag{5}$$

where $\phi_{i,g}$ represents the specific transformation corresponding to the g-th sub-vector within the i-th generator. This decomposition reduces the mapping dimensionality of the nonlinear functions, effectively mitigating mapping complexity while enhancing their expressiveness. Furthermore, the independence of these sub-mappings enables parallel processing, significantly reducing computational overhead.

Instance-wise normalization. The learned nonlinear generators are designed to operate on inputs within a predefined interval, known as the grid range. However, since the numerical distribution of the input latent vectors is typically unknown and dynamic, manually adapting the fixed RBF grid to cover all input values is impractical. This mismatch often causes some input features to fall outside the effective receptive field of the grid basis functions, reducing approximation efficiency. To resolve this, instead of adjusting the grid, we normalize the input features to match a standard distribution. For the g-th sub-vector $\mathbf{x}_g$, we compute the normalized vector $\hat{\mathbf{x}}_g$:

$$\hat{\mathbf{x}}_g = \frac{\mathbf{x}_g - \mu_g}{\sigma_g + \varepsilon}, \tag{6}$$

where μ_g and σ_g are the mean and standard deviation of $\mathbf{x}_g$ and ε is for stability. We choose the fixed grid interval $[-3, 3]$ as an empirical design, motivated by the approximately 99.7% probability mass of a standard Gaussian within this range. Normalization does not imply Gaussian-distributed latent inputs, so this heuristic does not guarantee 99.7% coverage; the actual fraction of inputs within the grid must be measured empirically.

Finally, the generator synthesizes values by combining a base non-linear activation with an RBF nonlinear expansion. Using the normalized vector $\hat{\mathbf{x}}_g$ from Eq. 6, we define the transformation $\phi_{i,g}$ applied to the sub-vector $\mathbf{x}_g$ as:

$$\phi_{i,g}(\mathbf{x}_g) = w_{b,i,g} \cdot \sigma(\hat{\mathbf{x}}_g) + \mathcal{F}_{\text{RBF},i,g}(\hat{\mathbf{x}}_g), \tag{7}$$

where $\sigma(\cdot)$ is a standard activation function (e.g., SiLU), $w_{b,i,g}$ is a learnable scalar scaling weight specific to the i-th generator and the g-th group, and $\mathcal{F}_{\text{RBF},i,g}(\cdot)$ represents the expressive RBF-based nonlinear term which will be detailed in Sec. 3.3.

3.3 Gaussian Radial Basis Functions

For notational simplicity, we omit the generator index i and group index g below. Given the normalized input distribution, we define a grid of K uniform centers $\{\mu_k\}_{k=1}^{K}$ over a fixed interval. The element-wise response of the k-th Gaussian basis function to the normalized input sub-vector $\hat{\mathbf{x}}_g$ is:

$$\varphi_k(\hat{\mathbf{x}}_g) = \exp\left(-\left(\frac{\hat{\mathbf{x}}_g - \mu_k}{h}\right)^2\right), \tag{8}$$

where h is the bandwidth determined by the grid spacing, and the operations are applied element-wise.

The final nonlinear output is a linear combination of these basis responses. Let $W_{\text{rbf}} = \{w_1, \ldots, w_K\} \in \mathbb{R}^K$ denote the learnable weight vector associated with the current group. The function $\mathcal{F}_{\text{RBF}}(\hat{\mathbf{x}}_g)$ is defined as:

$$\mathcal{F}_{\text{RBF}}(\hat{\mathbf{x}}_g) = \sum_{k=1}^{K} w_k \cdot \varphi_k(\hat{\mathbf{x}}_g). \tag{9}$$

In implementation, Eqs. (8) and (9) are vectorized via broadcasting, leveraging GPU parallelism for efficient inference.

3.4 Theoretical Analysis

We now analyze the theoretical properties of GenLoRA, focusing on the algebraic structure of the weight updates and optimization stability. These analyses establish preservation of the low-rank structure and conditional local gradient boundedness.

Proposition 3.1 (Rank Boundedness). *Let $\Delta W_{Gen} \in \mathbb{R}^{m\times n}$ be the weight update generated by GenLoRA with r pairs of generator functions $\{f_i^B\}_{i=1}^r$ and $\{f_i^A\}_{i=1}^r$, where $r \ll \min(m, n)$. The rank of the update matrix satisfies:*

$$\operatorname{rank}(\Delta W_{Gen}) \le r. \tag{10}$$

Proposition 3.1 guarantees that GenLoRA preserves the low-rank structure: each generated update has rank at most r, even though its factors depend nonlinearly on the latent inputs. The proof is provided in Appendix A. A refined rank bound for the fixed-grid RBF parameterization is provided in Appendix A.1.

Proposition 3.2 (Local Gradient Boundedness). *Fix $\varepsilon > 0$ and $h > 0$. On a compact neighborhood of the latent vectors Z and generator parameters Θ, assume that $\|\partial\mathcal{L}/\partial\Delta W_{\text{Gen}}\|_F$ is bounded. Then there exist finite constants $C_Z, C_\Theta > 0$ such that*

$$\left\|\frac{\partial\mathcal{L}}{\partial Z}\right\|_2 \le C_Z, \qquad \left\|\frac{\partial\mathcal{L}}{\partial\Theta}\right\|_2 \le C_\Theta. \tag{11}$$

Under these local regularity conditions, stabilized normalization and bounded Gaussian RBF derivatives yield a locally bounded generator Jacobian. This includes zero-variance groups and does not require a positive lower bound on their standard deviation. The proof and scope of the result are provided in Appendix B.

Table 1: The accuracy on Mathematical Reasoning tasks (Math10K) with various pretrained models.

	Method	#Params	AddSub	MultiArith	SingleEq	SVAMP	gsm8k	AQuA	Avg
LLaMA-3-8B	LoRA ($r = 8$)	4.72M	82.28	85.83	89.76	66.40	56.18	23.23	67.28
	MELoRA ($r = 8$)	4.72M	83.80	84.00	86.61	62.90	47.84	22.83	64.66
	HiRA ($r = 8$)	4.72M	85.82	88.16	91.34	68.90	54.66	27.55	69.41
	DoRA ($r = 8$)	4.92M	83.04	87.67	91.14	66.60	56.41	21.65	67.75
	AuroRA ($r = 2$)	1.18M	80.00	83.67	88.19	64.40	50.27	25.59	65.35
	AuroRA ($r = 8$)	4.72M	80.25	87.50	89.57	67.30	56.86	20.47	66.99
	GenLoRA ($r = 8, g = 16$)	0.98M	88.35	90.67	93.70	70.80	53.90	23.62	<u>70.17</u>
	GenLoRA ($r = 32, g = 16$)	2.16M	87.09	92.71	94.49	70.80	56.18	25.59	**71.14**
Gemma-7B	LoRA ($r = 8$)	4.82M	86.58	89.33	89.57	72.70	56.71	30.71	70.93
	MELoRA ($r = 8$)	4.82M	84.81	89.00	89.96	71.40	55.12	29.53	69.97
	HiRA ($r = 8$)	4.82M	85.85	88.67	90.55	72.10	55.04	26.38	69.76
	DoRA ($r = 8$)	5.16M	86.58	90.33	90.16	76.00	57.09	26.38	<u>71.09</u>
	AuroRA ($r = 2$)	1.20M	85.82	87.33	88.58	73.60	56.86	27.95	70.03
	AuroRA ($r = 8$)	4.82M	86.84	89.83	90.75	73.60	57.70	26.38	70.85
	GenLoRA ($r = 8, g = 16$)	0.95M	84.56	91.33	90.94	71.00	56.86	28.74	70.57
	GenLoRA ($r = 32, g = 16$)	1.98M	88.04	92.36	91.98	72.80	57.07	29.16	**71.90**
Qwen2.5-14B	LoRA ($r = 8$)	8.65M	91.14	96.17	92.52	86.10	75.51	33.46	79.15
	MELoRA ($r = 8$)	8.65M	92.91	94.00	91.73	83.60	74.75	31.89	78.15
	HiRA ($r = 8$)	8.65M	92.41	94.67	91.14	84.40	75.21	35.43	78.88
	DoRA ($r = 8$)	8.99M	93.16	96.00	92.52	85.50	75.28	33.86	79.39
	AuroRA ($r = 2$)	2.16M	93.41	95.00	91.73	84.10	73.24	33.07	78.43
	AuroRA ($r = 8$)	8.65M	93.16	96.50	92.52	85.00	75.74	34.65	79.59
	GenLoRA ($r = 8, g = 8$)	1.38M	92.15	96.00	92.13	85.90	76.19	35.83	<u>79.70</u>
	GenLoRA ($r = 32, g = 8$)	2.26M	91.65	96.17	93.31	87.80	74.68	38.58	**80.36**

4 EXPERIMENTS

In this section, we evaluate GenLoRA on three benchmarks across various architectures, followed by ablation studies verifying its scalability and visualizing singular values.

4.1 EXPERIMENTAL SETTINGS

First, we evaluate the natural language generation (NLG) capabilities of GenLoRA on the mathematical reasoning (Math10K)(Hu et al., 2023) and commonsense reasoning (Commonsense170K) benchmarks(Hu et al., 2023). Both benchmarks comprise a training corpus and multiple test sub-tasks. For each benchmark, we fine-tune the models on the training data and subsequently assess performance across all sub-tasks. To demonstrate the versatility of GenLoRA, we conduct experiments utilizing various model architectures and scales, including Gemma-7B(Team et al., 2024), LLaMA-3-8B (Dubey et al., 2024), and Qwen2.5-14B (Yang et al., 2024). Next, we assess the code generation capability of GenLoRA using LLaMA-3-8B (Dubey et al., 2024).We fine-tune the model on the Magicoder-Evol-Instruct-110k dataset(Wei et al., 2023), a curated and decontaminated subset of WizardCoder (Luo et al., 2023) comprising high-quality instruction–response pairs for programming tasks. Subsequently, we assess the performance on the Humaneval+ test with 50 sampled completions per problem. We report Pass@1, Pass@5, and Pass@10 accuracy following the standard protocol via the BigCode Evaluation Harness (Allal et al., 2022). Further experimental details are provided in Appendix K.

4.2 OVERALL PERFORMANCE

Results on Mathematical Reasoning Tasks. As shown in Table 1, GenLoRA demonstrates robust scalability and superior performance across diverse model architectures, consistently achieving higher accuracy with significantly fewer parameters. Specifically, on LLaMA3-8B, GenLoRA ($r = 8$) outperforms LoRA ($r = 8$) by 2.89 percentage points while utilizing only about one-fifth of the parameters (0.98M vs. 4.72M). Increasing GenLoRA to $r = 32$ yields a 3.86-point gain over the same LoRA ($r = 8$) baseline while still using less than half as many trainable parameters. On Gemma-7B, GenLoRA ($r = 32, g = 16$) reaches 71.90% with 1.98M trainable parameters. On both Gemma-7B and Qwen2.5-14B, its higher-rank configuration surpasses all rank-8 baselines with fewer trainable parameters. In comparison, AuroRA, which aims to optimize the trade-off between parameter efficiency and model expressivity, falls short against GenLoRA. For instance, on the

Table 2: The accuracy on Commonsense Reasoning benchmarks with various pretrained models.

	Method	#Params	OBQA	ARC-C	Wino	PIQA	Social	ARC-E	BoolQ	Hella	Avg
LLaMA-3-8B	LoRA ($r = 8$)	4.72M	81.60	78.50	82.08	87.54	77.79	91.67	70.40	87.95	82.19
	MELoRA ($r = 8$)	4.72M	74.80	75.00	71.43	84.71	72.98	89.44	67.74	80.84	77.12
	HiRA ($r = 8$)	4.72M	84.40	80.29	85.40	89.01	79.43	91.96	65.44	92.17	83.51
	DoRA ($r = 8$)	4.92M	81.00	78.41	82.24	87.65	77.99	91.58	70.15	88.12	82.14
	AuroRA ($r = 2$)	1.18M	76.20	77.13	72.93	84.28	73.08	89.73	67.77	82.45	77.95
	AuroRA ($r = 8$)	4.72M	81.60	78.50	82.08	87.27	77.48	91.62	70.46	88.05	82.13
	GenLoRA ($r = 8, g = 16$)	0.98M	85.60	79.01	84.85	88.57	79.38	91.75	71.83	91.45	<u>84.05</u>
	GenLoRA ($r = 32, g = 16$)	2.16M	86.80	80.12	85.71	88.74	80.91	91.37	72.63	92.87	**84.89**
Gemma-7B	LoRA ($r = 8$)	4.82M	81.80	82.76	86.19	88.30	77.99	92.55	68.84	91.16	83.70
	MELoRA ($r = 8$)	4.82M	79.60	79.86	78.61	84.93	74.67	89.73	69.27	87.81	80.56
	HiRA ($r = 8$)	4.82M	85.80	83.45	85.00	88.30	77.94	93.14	69.66	92.31	<u>84.45</u>
	DoRA ($r = 8$)	5.16M	82.60	80.89	84.06	85.47	75.95	91.16	69.20	89.42	82.34
	AuroRA ($r = 2$)	1.20M	80.00	81.48	79.24	85.75	75.90	90.57	68.13	85.63	80.84
	AuroRA ($r = 8$)	4.82M	82.40	82.76	83.50	85.96	76.82	91.75	70.28	90.08	82.94
	GenLoRA ($r = 8, g = 8$)	0.77M	84.80	80.80	86.90	87.54	77.43	92.26	71.25	92.07	84.13
	GenLoRA ($r = 32, g = 8$)	1.29M	88.07	84.92	89.14	89.75	81.79	95.09	72.13	93.02	**86.74**
Qwen2.5-14B	LoRA ($r = 8$)	8.65M	93.00	92.66	84.85	91.89	81.47	97.35	74.71	95.33	88.91
	MELoRA ($r = 8$)	8.65M	90.40	91.30	77.19	90.75	79.79	97.10	72.54	92.91	86.50
	HiRA ($r = 8$)	8.65M	94.20	93.26	89.98	93.09	83.88	97.98	75.87	96.50	90.60
	DoRA ($r = 8$)	8.99M	93.20	92.75	84.85	92.00	81.73	97.35	74.71	95.37	88.99
	AuroRA ($r = 2$)	2.16M	90.60	91.21	78.22	90.86	80.19	97.05	73.49	93.81	86.93
	AuroRA ($r = 8$)	8.65M	92.60	92.66	84.85	91.89	81.58	97.39	74.62	95.33	88.86
	GenLoRA ($r = 8, g = 8$)	1.38M	94.80	93.60	91.08	92.98	84.24	97.77	76.39	96.25	<u>90.89</u>
	GenLoRA ($r = 32, g = 8$)	2.26M	94.80	94.20	90.77	93.31	84.44	98.11	76.79	96.36	**91.10**

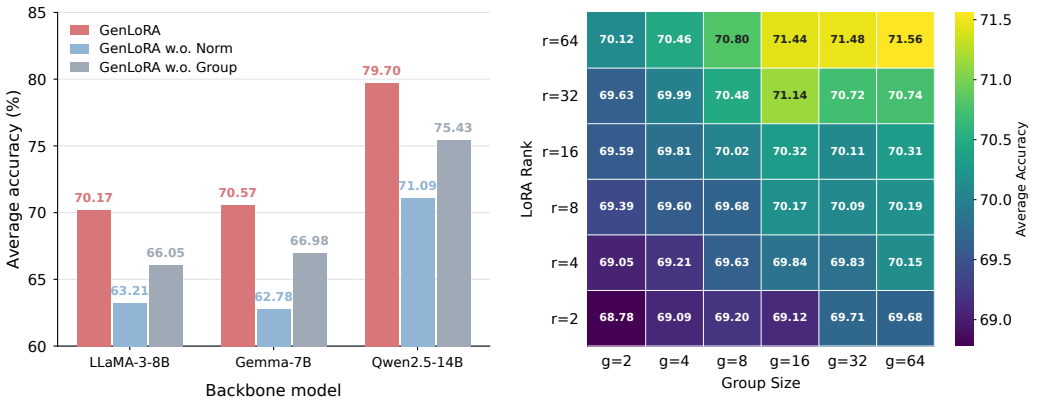
Table 3: The performance on Code Generation tasks (HumanEval+) with LLaMA-3-8B fine-tuned on Magicoder-Evol-Instruct-110k.

	Method	#Params	Train Time	Relative Time	Pass@1	Pass@5	Pass@10
LLaMA-3-8B	LoRA ($r = 8$)	4.72M	11.6h	1.00×	20.79	35.48	41.31
	MELoRA ($r = 8$)	4.72M	10.3h	0.88×	17.41	26.99	31.53
	HiRA ($r = 8$)	4.72M	10.6h	0.91×	25.33	35.36	39.72
	DoRA ($r = 8$)	4.92M	11.9h	1.03×	17.01	31.37	37.36
	AuroRA ($r = 2$)	1.18M	11.1h	0.96×	21.4	33.12	38.28
	AuroRA ($r = 8$)	4.72M	11.8h	1.02×	17.15	32.11	38.32
	GenLoRA ($r = 8, g = 16$)	0.98M	11.7h	1.01×	<u>25.42</u>	<u>36.36</u>	<u>41.48</u>
	GenLoRA ($r = 32, g = 16$)	2.16M	12.2h	1.05×	**28.30**	**41.92**	**47.77**

LLaMA3 benchmark, GenLoRA ($r = 8$) not only surpasses AuroRA ($r = 8$) by over 3% in accuracy but does so using approximately 80% fewer parameters. This clearly demonstrates that GenLoRA provides a far superior solution to the efficiency-performance trade-off.

Results on Commonsense Reasoning Tasks. As shown in Table 2, GenLoRA achieves the highest average accuracy among the evaluated methods in each backbone. At $r = 8$, its gains over LoRA are 1.86, 0.43, and 1.98 percentage points on LLaMA-3-8B, Gemma-7B, and Qwen2.5-14B, respectively, averaging 1.42 points across the three models. Increasing the rank to 32 yields 84.89%, 86.74%, and 91.10%, corresponding to gains of 2.70, 3.04, and 2.19 points over rank-8 LoRA. The Gemma-7B configuration uses $g = 8$ and 1.29M trainable parameters. All three rank-32 configurations retain less than half the trainable parameter count of their corresponding rank-8 LoRA baselines. Comparisons at matched $r = 32$ in Appendix G.5 further show that GenLoRA maintains competitive reasoning accuracy with **8.7×–15.3× fewer trainable parameters** than LoRA.

Results on Code Generation Tasks. At matched rank, GenLoRA improves code-generation accuracy while using fewer trainable parameters. At $r = 8$, Table 3 shows that GenLoRA reaches 25.42% Pass@1 versus 20.79% for LoRA, using 0.98M versus 4.72M parameters. At matched $r = 32$, the comparison in Appendix G.4 shows gains of 1.78, 1.73, and 2.08 percentage points in Pass@1, Pass@5, and Pass@10, respectively, with 2.16M versus 18.87M parameters. Separately, increasing GenLoRA to $r = 32$ yields 28.30% Pass@1, a 7.51-point gain over LoRA ($r = 8$) while retaining less than half its parameter budget. This cross-rank comparison illustrates the benefit of expanding rank at low parameter cost; the matched-rank comparison isolates performance at the same rank.



(a) Component ablations

(b) Rank and group size

Figure 3: Component and capacity ablations on Math10K. (a) Within each backbone, all three variants use the same trainable parameter budget; the vertical axis starts at 60%. (b) Accuracy of GenLoRA across ranks and group sizes on LLaMA-3-8B.

Efficiency Analysis. Beyond performance gains, we assess the computational overhead in Table 3. Remarkably, GenLoRA ($r = 8$) achieves a training time of 11.7h, corresponding to a relative time of $1.01\times$, comparable to the efficiency of standard LoRA. We attribute this low latency primarily to the utilization of Radial Basis Functions. Compared to B-spline functions, which require complex knot management and can be less efficient for vectorized processing, the core RBF computation relies on pure tensor operations such as vectorized element-wise exponentiations and matrix multiplications. This structure is inherently suited for massive GPU parallelism, enabling GenLoRA to introduce powerful expressive capabilities with minimal computational overhead.

4.3 Ablation Studies

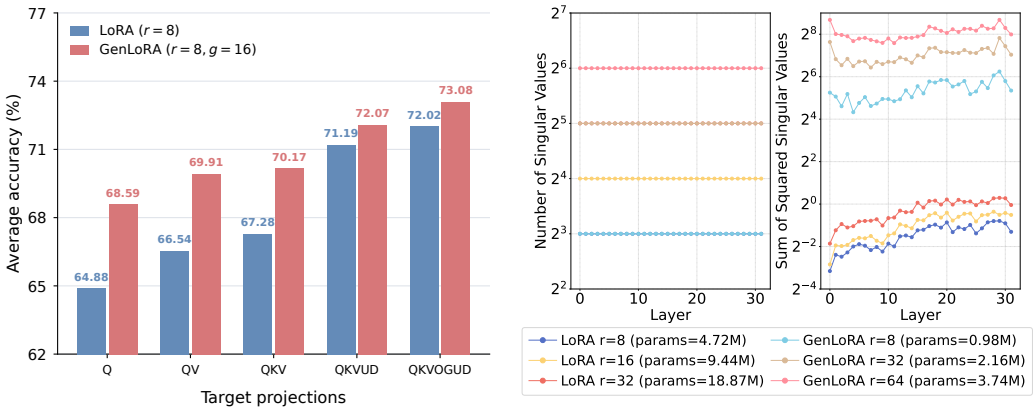
We evaluate instance-wise normalization and group-wise decomposition on Math10K using LLaMA-3-8B, Gemma-7B, and Qwen2.5-14B. Within each model, the full model and both ablated variants use the same trainable parameter budget, including the variant without grouping. As shown in Figure 3(a), removing normalization reduces average accuracy by 6.96, 7.79, and 8.61 percentage points, respectively. Removing grouping reduces accuracy by 4.12, 3.59, and 4.27 points. Both components therefore improve performance consistently across the three backbones under matched parameter budgets.

For LLaMA-3-8B, all three variants use approximately 0.98M trainable parameters. The ungrouped variant ($G = 1$, $r = 128$) matches the budget of the grouped configuration ($G = 16$, $r = 8$) by increasing the number of generators. However, under our fixed-grid RBF parameterization, the shared normalized inputs and the K RBF responses plus one base-activation response imply $\mathrm{rank}(\Delta W) \leq K + 1 = 16$ for each ungrouped update. Its lower accuracy shows that increasing the number of generators does not recover grouped generation's performance in this matched-budget setting. This compares parameterizations at the same trainable parameter budget, rather than updates with actual ranks 128 and 8. Appendix A.1 provides the feature-space factorization, rank bound, and implications for this ablation.

4.4 Scalability Analysis

We evaluate the scalability of GenLoRA on mathematical reasoning tasks using LLaMA3-8B from three perspectives: the LoRA rank, the group size, and the tuning granularity.

Impact of Rank and Group Size. We investigate the interaction between rank (r) and group size (g) by varying both within $\{2, 4, 8, 16, 32, 64\}$, as visualized in Figure 3(b). Generally, increasing r and g enhances model capacity. However, we observe that the performance gains tend to stabilize



(a) Tuning granularity

(b) Singular value analysis

Figure 4: Adaptation scope and singular value analysis on LLaMA-3-8B. (a) Math10K accuracy across target-projection configurations; the vertical axis starts at 62%. (b) Number and squared sum of singular values for query-projection updates. The singular-value counts of LoRA and GenLoRA at $r \in \{8, 32\}$ overlap.

around $g = 16$; further increasing the group size does not yield significant improvements. We believe that this is because the fitting capability of the non-linear functions has approached its limit.

Impact of Tuning Granularity. Finally, we evaluate the scalability of GenLoRA under different tuning granularities. We introduce five distinct settings: Q, QV, QKV, QKVUD, and QKVOGUD, where Q, K, and V denote the query, key, and value projection weights, and O, G, U, and D represent the output, gate, up, and down projection weights, respectively. As shown in Figure 4(a), GenLoRA consistently outperforms LoRA across all configurations. Most notably, in the most restricted setting (Q), GenLoRA surpasses LoRA by 3.71 percentage points, highlighting its superior efficiency in feature extraction. With comprehensive tuning (QKVOGUD), GenLoRA scales effectively to a peak accuracy of 73.08%, maintaining its robust lead over LoRA.

4.5 Singular Value Analysis

Figure 4(b) shows that both LoRA and GenLoRA attain numerical ranks matching their configured ranks. In particular, GenLoRA ($r = 64$) attains a measured numerical rank of 64 using 3.74M parameters, an eightfold increase in measured numerical rank over LoRA ($r = 8$, 4.72M). The spectral energy satisfies $\sum_i \sigma_i^2 = \|\Delta W\|_F^2$ and therefore measures the squared magnitude of the weight update. GenLoRA exhibits larger spectral energy than LoRA across the evaluated layers; in particular, GenLoRA ($r = 8$) produces larger updates than LoRA ($r = 32$) despite using fewer trainable parameters. Larger energy alone does not establish greater expressiveness or better generalization.

5 Conclusion

We introduce GenLoRA, which reduces the parameter cost of constructing low-rank basis vectors by generating them from shared latent vectors with compact nonlinear RBF generators. At the same rank, GenLoRA achieves competitive adaptation performance with substantially fewer trainable parameters. In reasoning experiments at matched $r = 32$, it uses approximately 8.7×–15.3× fewer parameters than LoRA; in code generation at the same rank, it improves Pass@1 by 1.78 percentage points. Increasing rank while remaining below the parameter budget of lower-rank LoRA provides a complementary route to further performance gains. These results support generative basis-vector parameterization as an effective approach to parameter-efficient adaptation.

REFERENCES

Loubna Ben Allal, Niklas Muennighoff, Logesh Kumar Umapathi, Ben Lipkin, and Leandro Von Werra. A framework for the evaluation of code generation models, 2022.

George A Baker Jr. The theory and application of the padé approximant method. Technical report, Los Alamos Scientific Lab., Univ. of California, N. Mex., 1963.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Martin Dietrich Buhmann. Radial basis functions. *Acta numerica*, 9:1–38, 2000.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, et al. Internlm2 technical report. *arXiv preprint arXiv:2403.17297*, 2024.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

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

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

Carl De Boor. On calculating with b-splines. *Journal of Approximation theory*, 6(1):50–62, 1972.

AI DeepSeek, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Haonan Dong, Wenhao Zhu, Guojie Song, and Liang Wang. Aurora: Breaking low-rank bottleneck of lora with nonlinear mapping. *arXiv preprint arXiv:2505.18738*, 2025.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. Krona: Parameter-efficient tuning with kronecker adapter. In *Enhancing LLM Performance: Efficacy, Fine-Tuning, and Inference Techniques*, pp. 49–65. Springer, 2025.

David E Edmunds, Petr Gurka, and Jan Lang. Properties of generalized trigonometric functions. *Journal of Approximation Theory*, 164(1):47–56, 2012.

Ziqi Gao, Qichao Wang, Aochuan Chen, Zijing Liu, Bingzhe Wu, Liang Chen, and Jia Li. Parameter-efficient fine-tuning with discrete fourier transform. *arXiv preprint arXiv:2405.03003*, 2024.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 523–533, 2014.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 5254–5276, 2023.

Hao Huang, Jingtao Ding, Mengqi Liao, Xin Wang, Jinyang Ban, Yuan Yuan, Huaiyu Wan, and Yong Li. Loragen: Structure-aware weight space learning for lora generation. In *The Fourteenth International Conference on Learning Representations*, 2026.

Qiushi Huang, Tom Ko, Zhan Zhuang, Lilian Tang, and Yu Zhang. Hira: Parameter-efficient hadamard high-rank adaptation for large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Yiping Ji, Hemanth Saratchandran, Cameron Gordon, Zeyu Zhang, and Simon Lucey. Efficient learning with sine-activated low-rank matrices. *arXiv preprint arXiv:2403.19243*, 2024.

Jiale Kang and Qingyu Yin. Miss: Revisiting the trade-off in lora with an efficient shard-sharing structure. In *International Conference on Learning Representations*, volume 2026, pp. 140327–140341, 2026.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. Parsing algebraic word problems into equations. *Trans. Assoc. Comput. Linguistics*, 3:585–597, 2015. doi: 10.1162/TACL_A_00160. URL https://doi.org/10.1162/tacl_a_00160.

Soroush Abbasi Koohpayegani, KL Navaneet, Parsa Nooralinejad, Soheil Kolouri, and Hamed Pirsiavash. Nola: Compressing lora using linear combination of random basis. *arXiv preprint arXiv:2310.02556*, 2023.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

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 Computer Vision and Pattern Recognition Conference*, pp. 13114–13123, 2025a.

Shiwei Li, Xiandi Luo, Haozhao Wang, Xing Tang, Ziqiang Cui, Dugang Liu, Yuhua Li, Xiuqiang He, and Ruixuan Li. Beyond higher rank: Token-wise input-output projections for efficient low-rank adaptation. *arXiv preprint arXiv:2510.23123*, 2025b.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. 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. 4582–4597, 2021.

Yinqiao Li, Linqi Song, and Hanxu Hou. Loran: Improved low-rank adaptation by a non-linear transformation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 3134–3143, 2024.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*, 2017.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024a.

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024b.

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

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37: 121038–121072, 2024.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Jooyoung Park and Irwin W Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pp. 2080–2094. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.NAACL-MAIN.168. URL https://doi.org/10.18653/v1/2021.naacl-main.168.

Pengjie Ren, Chengshun Shi, Shiguang Wu, Mengqi Zhang, Zhaochun Ren, Maarten de Rijke, Zhumin Chen, and Jiahuan Pei. Melora: Mini-ensemble low-rank adapters for parameter-efficient fine-tuning. *arXiv preprint arXiv:2402.17263*, 2024.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. *CoRR*, abs/1608.01413, 2016. URL http://arxiv.org/abs/1608.01413.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

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

Chinmay Savadikar, Xi Song, and Tianfu Wu. Wegeft: Weight-generative fine-tuning for multi-faceted efficient adaptation of large models. *arXiv preprint arXiv:2312.00700*, 2023.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Shaowen Wang, Linxi Yu, and Jian Li. Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems*, 37:54905–54931, 2024.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with oss-instruct. *arXiv preprint arXiv:2312.02120*, 2023.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *CoRR*, abs/2412.15115, 2024. doi: 10.48550/ARXIV.2412.15115. URL `https://doi.org/10.48550/arXiv.2412.15115`.

Bo Yin, Xingyi Yang, and Xinchao Wang. Don't forget the nonlinearity: Unlocking activation functions in efficient fine-tuning. *arXiv preprint arXiv:2509.13240*, 2025.

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

Yuchen Zeng and Kangwook Lee. The expressive power of low-rank adaptation. In *International Conference on Learning Representations (ICLR)*, 2024.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023.

Yibo Zhong, Haoxiang Jiang, Lincan Li, Ryumei Nakada, Tianci Liu, Linjun Zhang, Huaxiu Yao, and Haoyu Wang. Peanut: Parameter-efficient adaptation with weight-aware neural tweakers. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 2054–2065, 2026.

A Proof of Rank Boundedness

In this appendix, we provide the theoretical analysis for Proposition 3.1. We formally demonstrate that the algebraic structure of the GenLoRA update, constructed via generator functions, strictly adheres to the low-rank constraint.

Proposition 3.1 (Restated). *Let $\Delta W_{Gen} \in \mathbb{R}^{m\times n}$ be the weight update generated by GenLoRA with r pairs of generator functions $\{f_i^B\}_{i=1}^r$ and $\{f_i^A\}_{i=1}^r$, where $r \ll \min(m,n)$. The rank of the update matrix satisfies:*

$$\operatorname{rank}(\Delta W_{Gen}) \leq r. \tag{12}$$

First, we establish the algebraic form of the update. Given latent vectors Z_B, Z_A and generator parameters Θ, the instantiated basis vectors for $i \in \{1,\dots,r\}$ are $b_i = f_i^B(Z_B) \in \mathbb{R}^m$ and $a_i = f_i^A(Z_A) \in \mathbb{R}^n$. The GenLoRA update is defined as the summation of outer products: $\Delta W_{\text{Gen}} = \sum_{i=1}^r b_i a_i^\top$.

Lemma A.1. *The summation of outer products $\sum_{i=1}^r b_i a_i^\top$ is algebraically equivalent to the matrix product BA, where $B = [b_1,\dots,b_r]$ and $A = [a_1,\dots,a_r]^\top$.*

Proof. We verify this equality using element-wise index notation. Let $(M)_{jk}$ denote the entry of a matrix M at the j-th row and k-th column. Compute the element $(BA)_{jk}$:

$$(BA)_{jk} = \sum_{p=1}^{r} (B)_{jp}(A)_{pk}. \tag{13}$$

By construction, the p-th column of B is b_p, so $(B)_{jp} = (b_p)_j$. The p-th row of A is $a_p^\top$, so $(A)_{pk} = (a_p)_k$. Substituting these terms:

$$(BA)_{jk} = \sum_{p=1}^{r} (b_p)_j (a_p)_k. \tag{14}$$

Now, compute the element of the summation term. The (j,k)-th entry of the outer product $b_i a_i^\top$ is $(b_i)_j (a_i)_k$. Summing over $i = 1$ to r:

$$\left(\sum_{i=1}^{r} b_i a_i^\top \right)_{jk} = \sum_{i=1}^{r} (b_i)_j (a_i)_k. \tag{15}$$

Since the element-wise expressions are identical for all j, k, it follows that $\Delta W_{\text{Gen}} = BA$. □

Lemma A.2. *The rank of the factorized matrix product BA is bounded by r.*

Proof. Recall that the rank of a matrix corresponds to the dimension of its column space. The column space of the product BA, denoted $\mathcal{C}(BA)$, is defined as:

$$\mathcal{C}(BA) = \{B(Av) \mid v \in \mathbb{R}^n\}. \tag{16}$$

Since Av is a vector in $\mathbb{R}^r$, let $u = Av$. Then $\mathcal{C}(BA) \subseteq \{Bu \mid u \in \mathbb{R}^r\} = \mathcal{C}(B)$. Consequently, the dimension of the column space of BA cannot exceed the dimension of the column space of B:

$$\operatorname{rank}(BA) \leq \operatorname{rank}(B). \tag{17}$$

The matrix $B \in \mathbb{R}^{m \times r}$ has exactly r columns. The rank of a matrix is bounded by the number of its columns, so $\operatorname{rank}(B) \leq \min(m, r)$. Given the low-rank setting where $r < m$, we have $\operatorname{rank}(B) \leq r$. Therefore, $\operatorname{rank}(\Delta W_{\text{Gen}}) = \operatorname{rank}(BA) \leq r$. □

Conclusion. From Lemma A.1, the update ΔW_{Gen} is equivalent to the matrix product BA. From Lemma A.2, the rank of this product is at most r. Thus, Proposition 3.1 holds.

Scope of the rank guarantee. Proposition 3.1 is a structural result: nonlinear basis synthesis preserves the factorized update and its compatibility with weight merging. It does not by itself establish improved approximation or accuracy. Parameter efficiency follows instead from the incremental cost of lightweight generators, as discussed in the method and scalability analysis. The RBF approximation result in Appendix D concerns a sufficiently rich function family; it does not guarantee arbitrary basis vectors with a fixed finite grid or fixed parameter budget. Likewise, $\sum_j \sigma_j^2 = \|\Delta W\|_F^2$ measures update magnitude, so increased spectral energy alone does not prove greater expressiveness or better generalization.

A.1 Feature-space structure and rank bound of grouped generators

This section refines Proposition 3.1 for the fixed-grid RBF parameterization in Eqs. (5)–(9) and clarifies the interpretation of the grouping ablation.

Assumptions and notation. Within each factor, all generators use the same grouping, normalized latent inputs, fixed RBF centers $\{\mu_k\}_{k=1}^K$, bandwidth $h > 0$, and base activation σ. The base-activation and RBF combination coefficients are separately trainable for each generator and group. The latent vectors may also be trainable. We write

$$M_B = B \in \mathbb{R}^{m \times r}, \qquad M_A = A^\top \in \mathbb{R}^{n \times r}, \tag{18}$$

and let $d_B = m$ and $d_A = n$. For each side $s \in \{A, B\}$, the latent vector is partitioned into G_s groups, with group dimensions satisfying $\sum_{g=1}^{G_s} d_g^s = d_s$. Each normalized group is viewed as a column vector $\hat{x}_g^s \in \mathbb{R}^{d_g^s}$. The following analysis concerns algebraic rank at any fixed parameter state.

Feature-space factorization. Define the Gaussian response functions

$$\phi_k(t) = \exp\left(-\frac{(t - \mu_k)^2}{h^2} \right), \qquad k = 1, \ldots, K. \tag{19}$$

For group g on side s, collect the shared responses into

$$\Psi_g^s = \begin{bmatrix} \sigma(\hat{x}_g^s) & \phi_1(\hat{x}_g^s) & \cdots & \phi_K(\hat{x}_g^s) \end{bmatrix} \in \mathbb{R}^{d_g^s \times (K+1)}, \tag{20}$$

where all functions are applied element-wise. The coefficient vector of generator i in this group is

$$c^s_{i,g} = \begin{bmatrix} w^s_{b,i,g} & w^s_{i,g,1} & \cdots & w^s_{i,g,K} \end{bmatrix}^\top \in \mathbb{R}^{K+1}. \tag{21}$$

Let

$$C^s_g = \begin{bmatrix} c^s_{1,g} & \cdots & c^s_{r,g} \end{bmatrix} \in \mathbb{R}^{(K+1)\times r}. \tag{22}$$

Eqs. (7)–(9) then give

$$(M_s)_g = \Psi^s_g C^s_g, \qquad M_s = \begin{bmatrix} \Psi^s_1 C^s_1 \\ \vdots \\ \Psi^s_{G_s} C^s_{G_s} \end{bmatrix}. \tag{23}$$

Each column of C^s_g retains the separately trainable coefficients of one generator. Thus, this factorization does not identify the different generators with a single function or tie their trainable coefficients. It expresses their outputs as different linear combinations of the same evaluated nonlinear responses within each group.

Rank bound. The rank of a vertically concatenated matrix is at most the sum of the ranks of its blocks. Therefore,

$$\operatorname{rank}(M_s) \le \min\left\{ r, \sum_{g=1}^{G_s} \operatorname{rank}(\Psi^s_g C^s_g) \right\} \tag{24}$$

$$\le \min\left\{ r, \sum_{g=1}^{G_s} \operatorname{rank}(\Psi^s_g) \right\} \tag{25}$$

$$\le \min\{r, d_s, G_s(K+1)\}. \tag{26}$$

Since $\operatorname{rank}(A) = \operatorname{rank}(A^\top)$ and $\Delta W = BA$, it follows that

$$\operatorname{rank}(\Delta W) \le \min\{r, m, n, G_A(K+1), G_B(K+1)\}. \tag{27}$$

This bound holds at every parameter state, including when the latent vectors are optimized. Learning the latent vectors changes the evaluated feature matrices Ψ^s_g, but does not change their number of columns. The result is specific to the shared-input, fixed-grid parameterization considered here, rather than a restriction on arbitrary nonlinear generators.

Implications for the grouping ablation. For the ungrouped configuration, $G_A = G_B = 1$ and $K = 15$, so

$$\operatorname{rank}(\Delta W) \le \min\{r, m, n, 16\}. \tag{28}$$

Consequently, $r = 128$ specifies the number of generated columns and rows, not an attainable update rank of 128. For fixed latent inputs, increasing r adds coefficient vectors but does not enlarge the shared feature space of either factor.

Grouping instead introduces separately parameterized feature blocks on disjoint coordinate subsets. With $G_A = G_B = 16$ and $K = 15$, the feature-count ceiling is 256 per factor, before imposing the factor dimensions and the nominal width r. In the grouped ablation configuration with $r = 8$, the update rank nevertheless remains at most 8.

The two configurations satisfy the same product $rG = 128$, where $G_A = G_B = G$. With both latent vectors trainable and no additional trainable terms, their parameter counts for adapting a weight matrix are equal:

$$P = m + n + 2rG(K+1). \tag{29}$$

Their equal parameter counts do not imply identical feature spaces or identical realizable updates.

The lower accuracy of the ungrouped variant supports the usefulness of group-wise generation in this matched-budget setting. The factorization above provides a structural distinction between the two parameterizations, but the accuracy gap alone does not isolate its contribution from changes in optimization. In particular, the result should not be interpreted as showing that the grouped configuration has a higher actual update rank.

Scope for the main configurations. For $K = 15$, $G_A, G_B \in \{8, 16\}$, and $r \leq 64$, the feature-count terms $G_A(K+1)$ and $G_B(K+1)$ are at least 128. Hence, this additional feature-count ceiling does not tighten the usual bound $\mathrm{rank}(\Delta W) \leq r$ in these configurations. This observation does not, by itself, guarantee that the learned factors attain full rank or that the parameterization can represent every rank-r update.

B Proof of Local Gradient Boundedness

We prove Proposition 3.2 under explicit local regularity conditions. Fix $\varepsilon > 0$ and Gaussian bandwidth $h > 0$, and restrict the latent vectors and generator parameters to a compact neighborhood. Assume that the upstream gradient $\|\partial\mathcal{L}/\partial\Delta W_{\mathrm{Gen}}\|_F$ is uniformly bounded there. This assumption is not implied merely by finite network outputs, and compactness here does not assert that an entire training trajectory remains bounded.

Stabilized normalization. For a group $z \in \mathbb{R}^d$, define

$$P = I - \frac{1}{d}\mathbf{1}\mathbf{1}^T, \qquad u = Pz, \qquad \sigma = \frac{\|u\|_2}{\sqrt{d}}, \qquad \hat{z} = \frac{u}{\sigma + \varepsilon}. \tag{30}$$

The normalized input satisfies $\|\hat{z}\|_2 = \sqrt{d}\,\sigma/(\sigma+\varepsilon) \leq \sqrt{d}$. For $\sigma > 0$, its Jacobian is

$$J_{\mathrm{norm}} = \frac{P}{\sigma+\varepsilon} - \frac{uu^T}{d\sigma(\sigma+\varepsilon)^2}. \tag{31}$$

Using $\|u\|_2^2 = d\sigma^2$ gives

$$\|J_{\mathrm{norm}}\|_2 \leq \frac{1}{\sigma+\varepsilon} + \frac{\sigma}{(\sigma+\varepsilon)^2} \leq \frac{5}{4\varepsilon}. \tag{32}$$

At $u = 0$, for a perturbation v, $\hat{z}(z+v) - Pv/\varepsilon = O(\|v\|_2^2)$, so the composite normalization map is differentiable with Jacobian P/ε. Thus no positive lower bound on the standard deviation is needed.

Generator and weight-update Jacobians. For $\varphi_k(t) = \exp(-(t-\mu_k)^2/h^2)$,

$$0 < \varphi_k(t) \leq 1, \qquad |\varphi_k'(t)| \leq \frac{\sqrt{2/e}}{h}. \tag{33}$$

Because normalized inputs are bounded, the SiLU residual and its derivative are bounded on the relevant domain. The generator coefficients are bounded on the compact parameter set. Consequently, the generated factors A, B and their Jacobians with respect to Z and Θ are uniformly bounded. For the matrix product $\Delta W_{\mathrm{Gen}} = BA$,

$$\mathrm{d}\Delta W_{\mathrm{Gen}} = (\mathrm{d}B)A + B(\mathrm{d}A), \qquad \|\mathrm{d}\Delta W_{\mathrm{Gen}}\|_F \leq \|A\|_2\|\mathrm{d}B\|_F + \|B\|_2\|\mathrm{d}A\|_F. \tag{34}$$

This bounds the factor-to-update Jacobian that also enters the chain rule. Combining these bounds with the assumed bound on the upstream gradient yields finite constants $C_Z, C_\Theta > 0$ such that

$$\left\|\frac{\partial\mathcal{L}}{\partial Z}\right\|_2 \leq C_Z, \qquad \left\|\frac{\partial\mathcal{L}}{\partial\Theta}\right\|_2 \leq C_\Theta, \tag{35}$$

where the parameters are vectorized when taking norms. The constants may depend on the neighborhood, model dimensions, h, and ε. The result establishes conditional local boundedness, not uniform control over arbitrary training trajectories or a guarantee of stable optimization. In particular, small ε or h can still produce large bounds.

C Algorithmic Complexity Analysis

In this section, we provide a theoretical comparison between Radial Basis Functions (RBFs) and B-Spline from an algorithmic perspective. We analyze why RBFs constitute a more efficient choice for the high-dimensional generation tasks in GenLoRA.

C.1 Computational Operations

RBFs: Single-Step Global Mapping The RBF generator models nonlinearity via a unified, single-step calculation that is inherently parallelizable. For a given input x and a grid resolution N (denoting the number of basis functions), the algorithm executes three fundamental operations: distance calculation ($(x - \mu_k)^2$), Gaussian activation ($\exp(\cdot)$), and weighted aggregation. Crucially, these operations are strictly independent across the grid dimension N, meaning there are no data dependencies between the k-th and $(k+1)$-th basis functions. Consequently, the entire process can be fused and expressed as a single vectorized matrix operation. In terms of sequential complexity, this represents an $\mathcal{O}(1)$ depth algorithm, regardless of the grid size, making it exceptionally efficient on modern SIMD architectures like GPUs.

B-Spline: Recursive Dependency B-Splines are defined by the Cox-de Boor recursion algorithm, which inherently requires an iterative computation process. A spline of order k (degree $k-1$) is computed by recursively evaluating basis functions from order 0 up to k. This process begins with an initialization step to locate the input x within the knot vector intervals. Subsequently, the algorithm enters a recursive loop of depth k, where at each depth $d \in [1, k]$, the value of a basis function is derived via linear interpolation of two basis functions from the previous depth $d-1$. While the total operation count scales as $\mathcal{O}(k^2)$, the primary computational bottleneck is the requirement for $\mathcal{O}(k)$ sequential iterations to resolve these dependencies. Unlike RBFs, B-Splines impose strict sequential constraints: the basis values at step d cannot be computed until step $d-1$ is fully resolved, which significantly limits the potential for operator fusion and parallel execution.

C.2 Space and Gradient Complexity

In deep learning, the training memory cost is dominated by the requirement to store intermediate activations for gradient computation. For RBF generators, the computational graph is shallow, necessitating the backpropagation engine to trace through only a single exponential layer. Consequently, the space complexity remains linear with respect to the grid resolution N, i.e., $\mathcal{O}(N)$. In contrast, B-Spline generators construct an inherently deep computational graph due to their recursive definition. Since the algorithm iterates k times, the automatic differentiation engine is forced to cache intermediate basis tensors for every step of the recursion to correctly apply the chain rule. This results in a space complexity of $\mathcal{O}(N \cdot k)$. For high-dimensional inputs, this multiplicative factor k significantly amplifies the memory footprint during training, making B-Splines less memory-efficient than RBFs.

C.3 Summary

Table 4 summarizes the algorithmic differences. While B-Splines offer compact support, their recursive nature and higher memory complexity for gradient computation make them less suitable for the high-throughput requirements of GenLoRA. RBFs provide a mathematically simpler and parallel-friendly alternative that aligns better with the efficiency demands of large-scale neural network training.

Table 4: Algorithmic comparison between RBF and B-Spline generators.

Aspect	RBF (Ours)	B-Spline
Structure	Single-step Mapping	k-step Recursion
Dependencies	Independent (Parallel)	Sequential (Iterative)
Gradient Memory	$\mathcal{O}(N)$	$\mathcal{O}(N \cdot k)$

D Approximation Capability of RBF Generators

In this section, we provide the theoretical justification for the strong fitting capability of our RBF-based generators, drawing upon the fundamental mathematical properties of radial basis functions analyzed by Buhmann (2000).

D.1 Universal Approximation Theorem

The theoretical foundation of our RBF generator is grounded in the analysis provided by Park & Sandberg (1991). We first cite the original Theorem 2 regarding the approximation capabilities of RBF networks:

Theorem D.1 (Theorem 2 in Park & Sandberg (1991)). *Let $K : \mathbb{R}^r \to \mathbb{R}$ be an integrable bounded function such that K is continuous almost everywhere and $\int_{\mathbb{R}^r} K(x)dx \neq 0$. Then the family S_K is dense in $C(\mathbb{R}^r)$ with respect to the metric d defined by*

$$d(f,g) = \sum_{n=1}^{\infty} 2^{-n} \frac{\|(f-g)\cdot 1_{[-n,n]^r}\|_\infty}{1+\|(f-g)\cdot 1_{[-n,n]^r}\|_\infty}. \tag{36}$$

As further explained by the authors, the statement in Theorem 2 is equivalent to the statement that S_K is uniformly dense on compacta in $C(\mathbb{R}^r)$. That is, for any continuous function f, any $\epsilon > 0$, and any compact subset $C \subset \mathbb{R}^r$, there exists a $q \in S_K$ such that $\|(q-f)\cdot 1_C\|_\infty < \epsilon$.

D.2 Proof of Fitting Capability in GenLoRA

We now connect the above theoretical result to the element-wise generator design in GenLoRA.

Setup. Since GenLoRA applies RBF generators in an element-wise manner for computational efficiency and parameter reduction, we model the target weight generation process at the scalar level. Specifically, we define a continuous 1D mapping $w^* : \mathbb{R} \to \mathbb{R}$ from a single scalar latent component $z^{(i)}$ to its corresponding target weight element. We emphasize that the latent components $z^{(i)}$ are learnable parameters and are not explicitly constrained to lie in a compact domain. Accordingly, our analysis focuses on the approximation behavior of the 1D generator when restricted to any bounded interval of the latent space, which is the standard setting for applying universal approximation results.

The generator in GenLoRA employs 1D Gaussian radial basis functions of the form $\phi(x) = \exp(-(x-\mu)^2/h^2)$, which are integrable, bounded, continuous, and have a non-zero integral over $\mathbb{R}$. Therefore, the kernel ϕ satisfies the assumptions required in Theorem D.1 for the univariate case ($r = 1$).

Approximation Result. Let $\mathcal{Z}_R \subset \mathbb{R}$ denote an arbitrary compact interval of the scalar latent space, for example, $\mathcal{Z}_R = [-R, R]$. Consider any continuous scalar-valued target function $w^*(z^{(i)})$ on this compact interval $\mathcal{Z}_R$. Under the assumption that the target weight elements can be independently modeled from their corresponding scalar latent inputs, by the uniform density property of Theorem D.1, for any $\epsilon > 0$, there exists a 1D RBF generator $g(z^{(i)}) \in S_K$ with sufficiently many basis functions such that

$$\sup_{z^{(i)} \in \mathcal{Z}_R} |g(z^{(i)}) - w^*(z^{(i)})| < \epsilon. \tag{37}$$

Conclusion. This result establishes the expressive power of the proposed RBF-based generator in a local sense: for any bounded interval of the latent space encountered during training, the element-wise generator can approximate the target 1D weight mapping to arbitrary precision given sufficient capacity. Combined with the locally bounded gradient analysis in Appendix B, this characterizes its element-wise approximation capability and conditional local regularity, without guaranteeing global optimization stability.

E Algorithm Workflow

The complete training and inference workflow of GenLoRA is presented in Algorithm 1.

Zero-Latency Inference via Weight Merging. A critical advantage of GenLoRA is its compatibility with standard weight merging techniques. Unlike adapter methods that apply non-linearities directly to the input activation x (e.g., $h = W_0x + B\sigma(Ax)$), GenLoRA restricts non-linearity solely to the parameter generation process. Once training is complete, the latent vectors Z and generator

Algorithm 1 Algorithm Workflow of GenLoRA

Require: Pretrained weight $W_0 \in \mathbb{R}^{m\times n}$; latents Z_B, Z_A; generator parameters Θ_B, Θ_A; rank r; group counts G_B, G_A; training data $\{(x_i, y_i)\}_{i=1}^{N}$.
Ensure: Fine-tuned model weights W.

Function GENERATOR(Z, Θ, r, G):
  Partition Z into column vectors $\mathbf{x}_g \in \mathbb{R}^{d_g}$, $g = 1, \ldots, G$;
  for $g = 1, \ldots, G$ **do**
    $\hat{\mathbf{x}}_g \leftarrow \text{InstanceNorm}(\mathbf{x}_g)$; {Eq. (6)}
    Allocate $V_g \in \mathbb{R}^{d_g \times r}$;
    for $i = 1, \ldots, r$ **do**
      $V_g[:, i] \leftarrow \phi_{i,g}(\mathbf{x}_g)$; {Eqs. (7)–(9)}
      {Independent coefficients $w_{b,i,g}$ and $\{w_{i,g,k}\}_{k=1}^{K}$; shared $\hat{\mathbf{x}}_g$}
    end for
  end for
  $M \leftarrow \text{ConcatRows}(V_1, \ldots, V_G) \in \mathbb{R}^{(\sum_g d_g)\times r}$;
  return M;

/* **Training Phase** */
for epoch $t \leftarrow 1$ to T **do**
  for each minibatch $\{x, y\}$ in training data **do**
    /* 1. Synthesize Rank Matrices via Generators */
    $B \leftarrow \text{GENERATOR}(Z_B, \Theta_B, r, G_B) \in \mathbb{R}^{m\times r}$;
    $A \leftarrow [\text{GENERATOR}(Z_A, \Theta_A, r, G_A)]^\top \in \mathbb{R}^{r\times n}$;
    /* 2. Forward pass with Adapter */
    $\Delta W \leftarrow BA$;
    $h \leftarrow W_0 x + \Delta W x$; {Eq. (1)}
    Compute loss $\mathcal{L}(h, y)$;
    /* 3. Backpropagate through Generators and Latents */
    Update $\{Z_B, Z_A, \Theta_B, \Theta_A\}$ via gradient descent;
  end for
end for

/* **Inference Preparation (Static Merge)** */
Function MERGEWEIGHTS():
  $B_{\text{final}} \leftarrow \text{GENERATOR}(Z_B, \Theta_B, r, G_B)$;
  $A_{\text{final}} \leftarrow [\text{GENERATOR}(Z_A, \Theta_A, r, G_A)]^\top$;
  $\Delta W \leftarrow B_{\text{final}} A_{\text{final}}$; {Construct effective update}
  $W \leftarrow W_0 + \Delta W$; {Merge into backbone}
  return W;

parameters Θ become static. Consequently, the synthesized matrices B and A evaluate to fixed constant matrices. This allows us to pre-compute the full update matrix $\Delta W = BA$ and merge it algebraically into the pretrained weights: $W_{\text{merged}} = W_0 + \Delta W$. During inference, the model utilizes W_{merged} directly, incurring **zero additional computational overhead** compared to the base model.

F STANDARD DEVIATIONS

To ensure the reproducibility and reliability of our findings, we conducted primary experiments across three independent runs using different seeds. While the main body of the paper reports mean performance, we provide a detailed breakdown of the corresponding standard deviations in Table 5. It is observed that the performance fluctuations (standard deviations) for both mathematical and commonsense reasoning tasks are remarkably low. Crucially, the margin of improvement delivered by GenLoRA consistently outweighs these minor variabilities, confirming that our method's gains are statistically substantial rather than products of random noise.

Table 5: The standard deviation of the average accuracy on mathematical and commonsense reasoning tasks.

Method	Mathematical Reasoning			Commonsense Reasoning		
	LLaMA-3-8B	Gemma-7B	Qwen2.5-14B	LLaMA-3-8B	Gemma-7B	Qwen2.5-14B
LoRA (r = 8)	0.76	0.24	0.54	0.17	0.52	0.21
MELoRA (r = 8)	0.61	0.14	0.74	0.49	0.17	0.24
HiRA (r = 8)	0.38	0.45	0.22	0.21	0.35	0.11
DoRA (r = 8)	0.19	0.43	0.26	0.34	0.47	0.17
AuroRA (r = 2)	0.72	0.55	0.41	0.28	0.48	0.16
AuroRA (r = 8)	0.65	0.42	0.36	0.22	0.41	0.14
GenLoRA (r = 8)	0.32	0.22	0.27	0.19	0.31	0.18
GenLoRA (Higher Rank)†	0.41	0.29	0.38	0.28	0.37	0.16

†Higher Rank denotes $r = 32$ for all backbones.

G Additional Comparative Experiments

This appendix extends the evaluation of GenLoRA with additional PEFT baselines, comparisons of task-specific adaptation and cross-task generation, training-time and GPU-memory measurements, matched-rank reasoning experiments, and visual recognition results. We also discuss adapter storage and potential compatibility with advanced LoRA variants.

G.1 Broader PEFT Paradigm Comparisons

We first compare GenLoRA against a wide spectrum of PEFT baselines on the Math10K dataset using LLaMA-3-8B. The baselines are broadly categorized into three distinct paradigms:

- **Prompt/Activation Tuning** (Prompt-Tuning(Lester et al., 2021), Prefix-Tuning(Li & Liang, 2021), IA^3(Liu et al., 2022)): These methods are extremely parameter-efficient (typically $\sim$ 1M parameters) but often incur additional inference overhead, such as extended context windows or latency costs, and generally struggle with complex reasoning tasks.
- **Spectral/Structural Tuning** (FourierFT(Gao et al., 2024), KronA(Edalati et al., 2025)): While these approaches outperform prompt tuning in mathematical reasoning, their rigid mathematical constraints inherently limit overall model expressivity.
- **Explicit Low-Rank/SVD** (LoRA, AdaLoRA(Zhang et al., 2023), PiSSA(Meng et al., 2024)): These methods are highly robust and introduce zero additional inference latency via weight merging. However, they suffer from the explicit-rank bottleneck, where scaling the model capacity requires linear parameter growth relative to the model dimensions m and n.

As demonstrated in Table 6, GenLoRA bridges the gap between these paradigms. By treating non-linearity as effective rank, GenLoRA's capacity scales with non-linear generators whose parameter size is strictly $\ll \min(m, n)$. Consequently, GenLoRA matches the minimal parameter footprint of prompt-tuning methods (0.98M) while significantly outperforming explicit-rank methods with an accuracy of 70.17%.

Table 6: Performance comparison across PEFT paradigms using LLaMA-3-8B fine-tuned on Math10K. Avg. is the unweighted mean across the six tasks.

Method	#Params	AddSub	MultiArith	SingleEq	SVAMP	GSM8K	AQuA	Avg.
Prompt-Tuning	1.02M	76.62	81.23	83.77	62.40	42.95	20.48	61.25
Prefix-Tuning	1.31M	80.49	83.64	85.76	64.50	45.09	23.41	63.82
FourierFT	1.06M	85.06	88.67	88.98	64.30	45.40	27.95	66.73
IA^3	1.25M	86.84	83.50	86.42	64.50	50.64	26.38	66.38
LoRA ($r = 8$)	4.72M	82.28	85.83	89.76	66.40	56.18	23.23	67.28
AdaLoRA ($r = 8$)	4.72M	80.57	84.22	89.47	64.80	52.12	25.44	66.10
KronA	1.52M	80.76	84.17	89.37	63.80	50.11	27.95	66.03
PiSSA ($r = 8$)	4.72M	82.78	86.89	91.27	66.40	54.11	25.33	67.80
GenLoRA ($r = 8, g = 16$)	**0.98M**	88.35	90.67	93.70	70.80	53.90	23.62	**70.17**

G.2 Recent Nonlinear and Generative PEFT Baselines

We extend the Math10K evaluation on LLaMA-3-8B to NoRA and NoRA++, MiSS (Kang & Yin, 2026), PEANuT (Zhong et al., 2026), and WeGeFT (Savadikar et al., 2023), complementing the broader PEFT comparison in Table 6. These baselines cover activation-space and hybrid adaptation, parameter sharing, and weight generation. A complementary comparison with LoRAGen (Huang et al., 2026) in the cross-task generation setting is provided in Appendix G.3. Table 7 reports trainable parameter counts and per-task accuracy across six mathematical reasoning benchmarks, together with their unweighted average. With 0.98M parameters, GenLoRA achieves 70.17% average accuracy, exceeding rank-16 LoRA by 0.50 percentage points while using approximately $9.6\times$ fewer parameters. It also outperforms the additional baselines in this evaluation, supporting the accuracy–parameter trade-off of compact basis-vector generation.

Table 7: Additional nonlinear and generative PEFT comparisons using LLaMA-3-8B fine-tuned on Math10K. Avg. is the unweighted mean accuracy (%) across the six tasks.

Method	#Params	AddSub	MultiArith	SingleEq	SVAMP	GSM8K	AQuA	Avg.
LoRA ($r = 8$)	4.72M	82.28	85.83	89.76	66.40	56.18	23.23	67.28
LoRA ($r = 16$)	9.44M	83.56	91.34	92.43	68.80	56.97	24.89	69.67
NoRA ($r = 16$)	1.06M	81.76	82.26	85.14	64.70	53.19	19.07	64.36
NoRA++ ($r = 16$)	5.78M	83.08	85.79	89.96	68.80	56.06	24.61	68.04
MiSS ($r = 16$)	3.15M	84.67	86.93	88.71	64.60	55.21	25.35	67.58
PEANuT	3.14M	84.04	85.07	89.87	64.00	54.19	24.14	66.89
WeGeFT	6.29M	87.45	83.09	92.34	68.80	56.23	21.62	68.25
GenLoRA ($r = 8, g = 16$)	**0.98M**	88.35	90.67	93.70	70.80	53.90	23.62	**70.17**

G.3 Cross-Task Generation versus Task-Specific Adaptation

GenLoRA is designed for parameter-efficient task-specific adaptation, whereas LoRAGen (Huang et al., 2026) amortizes adaptation through a shared generator trained across source tasks. Text-to-LoRA and Doc-to-LoRA likewise motivate the distinction between conditional adapter generation and direct target-task optimization; the quantitative comparison below evaluates LoRAGen. We follow LoRAGen's unseen-task protocol: its generator is trained on 136 FLAN source tasks with 20 natural-language descriptions per task, excluding AP-Rec, AP-Pos, QASC-1, QASC-2, WQA-T, and WQA-A. At evaluation, LoRAGen receives only task descriptions; we generate three adapters per task and average their scores. LoRA and GenLoRA instead use the corresponding target-task training data. All adapted methods use FLAN-T5-Large, rank $r = 16$, and the same six evaluation tasks and pipeline.

Table 8: Cross-task generation and task-specific adaptation on FLAN-T5-Large. Parameter counts and training times for LoRA and GenLoRA are totals across the six target tasks; LoRAGen's entries refer to its shared generator and exclude source-task adapter-library construction. LoRAGen uses no target-task training data, whereas LoRA and GenLoRA do.

Method	#Params	Time (h)	AP-Rec	AP-Pos	QASC-1	QASC-2	WQA-T	WQA-A	Avg.
FLAN-T5-Large (unadapted)	0	0	69.5	35.1	24.5	10.2	9.6	62.4	35.22
LoRAGen ($r = 16$)	54.33M	3.90	75.4	42.2	29.6	15.2	14.9	68.4	40.95
LoRA ($r = 16$)	4.72M	0.32	80.1	45.3	68.1	56.7	75.2	72.9	66.38
GenLoRA ($r = 16, g = 16$)	1.33M	0.32	82.3	49.2	67.4	57.2	76.8	74.3	**67.87**

As shown in Table 8, LoRAGen improves over unadapted FLAN-T5-Large by 5.73 points without target-task training. With target-task supervision, GenLoRA reaches 67.87%, exceeding LoRA by 1.49 points with $3.55\times$ fewer stored adapter parameters and the same reported total training time. Its 26.92-point advantage over LoRAGen reflects a different adaptation regime, not an isolated comparison of generator architectures. LoRAGen trades an upfront training cost for adaptation without target-task optimization; GenLoRA is suitable when lightweight target-task training is available. The reported times describe these particular workloads and do not establish a universal runtime advantage as the number of tasks grows.

Storage across tasks. Using the aggregate counts in Table 8, the average GenLoRA adapter contains $1.33/6 \approx 0.222$M parameters. For comparable target-task configurations, retaining T adapters requires

$$S_{\text{GenLoRA}}(T) \approx \frac{1.33}{6} T \text{ M}, \qquad T^\star \approx \frac{54.33}{1.33/6} \approx 245. \tag{38}$$

Thus the adapter collection remains smaller than the 54.33M-parameter shared generator below approximately 245 tasks under this accounting. This estimate assumes equal storage precision, excludes the shared backbone, optimizer states, metadata, and LoRAGen's source-task library, and assumes that generated LoRAGen adapters are not cached. It is a configuration-specific parameter-count comparison rather than a general storage bound. GenLoRA adapters can be stored and loaded on demand using an adapter-library workflow, or merged into a deployment model without additional inference computation.

G.4 Training Time and GPU Memory Analysis

We compare LoRA and GenLoRA at the same rank using the code-generation setting of Table 3: LLaMA-3-8B fine-tuned on Magicoder-Evol-Instruct-110k and evaluated on HumanEval+. Table 9 reports training time, peak GPU memory, and accuracy. At $r = 8$, GenLoRA adds approximately 1% training time and reduces peak memory by 1,264 MB (4.0%). At $r = 32$, both methods take 12.2 hours at the reported precision; GenLoRA uses $8.7\times$ fewer trainable parameters, saves 223 MB of peak memory, and improves Pass@1/5/10 by 1.78/1.73/2.08 percentage points. The $1.05\times$ relative time is measured against LoRA at $r = 8$, not against a matched-rank baseline. Thus, the measured generator overhead is small in these settings, while peak-memory savings depend on the workload.

Table 9: Matched-rank code-generation results. Relative training time uses LoRA ($r = 8$) as the reference for all rows.

Method	#Params	Time (h)	Relative	Peak GPU (MB)	Pass@1	Pass@5	Pass@10
LoRA ($r = 8$)	4.72M	11.6	1.00×	31,724	20.79	35.48	41.31
GenLoRA ($r = 8, g = 16$)	0.98M	11.7	1.01×	30,460	25.42	36.36	41.48
LoRA ($r = 32$)	18.87M	12.2	1.05×	32,095	26.52	40.19	45.69
GenLoRA ($r = 32, g = 16$)	2.16M	12.2	1.05×	31,872	**28.30**	**41.92**	**47.77**

G.5 Matched-Rank Comparisons on Reasoning Tasks

To separate the effect of rank from that of parameterization, we evaluate LoRA, MELoRA, HiRA, DoRA, AuroRA, and GenLoRA at matched ranks under the same training and evaluation settings for each model and task family. We use $r = 32$ for all three backbones. Table 10 reports average accuracy on the mathematical and commonsense reasoning benchmarks. GenLoRA attains the highest accuracy in four of the six settings and trails the strongest baseline by only 0.06 and 0.13 percentage points in the two Qwen2.5-14B settings. Its trainable parameter count is $8.7\times$–$15.3\times$ smaller than matched-rank LoRA under the stated QKV configuration. These results show that the parameter-efficiency advantage persists when the baselines use the same rank.

G.6 Compatibility with Advanced LoRA Variants

GenLoRA changes how the factors are parameterized while retaining the update structure $\Delta W_{\text{Gen}} = BA$. This permits potential combinations with methods that modify weight decomposition, rank allocation, or initialization. The constructions below describe structural compatibility; we do not report combined-method experiments here.

Magnitude–direction decomposition. In DoRA (Liu et al., 2024a), generated factors can parameterize the low-rank directional update:

$$W' = m \odot \frac{W_0 + \Delta W_{\text{Gen}}}{\|W_0 + \Delta W_{\text{Gen}}\|_c}, \qquad \Delta W_{\text{Gen}} = B_\Theta(Z) A_\Theta(Z), \tag{39}$$

where $\|\cdot\|_c$ denotes column-wise norms and the learnable magnitude vector m is broadcast over columns. For nonzero column norms, this preserves DoRA's magnitude–direction decomposition while replacing explicit low-rank factors with compact generators.

Table 10: Matched-rank reasoning comparisons. Parameter counts and average accuracies are reported separately for each task family. GenLoRA uses $g = 16$ for LLaMA-3-8B and $g = 8$ for Qwen2.5-14B; for Gemma-7B it uses $g = 16$ on Math10K and $g = 8$ on commonsense reasoning. Bold accuracies mark the best result within each model and task family.

		Math10K		Commonsense	
Model / Rank	Method	#Params	Avg.	#Params	Avg.
LLaMA-3-8B ($r = 32$)	LoRA	18.87M	70.58	18.87M	84.77
	MELoRA	18.87M	69.16	18.87M	82.16
	HiRA	18.87M	70.97	18.87M	83.93
	DoRA	19.07M	70.75	19.07M	84.58
	AuroRA	18.87M	70.42	18.87M	83.73
	GenLoRA	2.16M	**71.14**	2.16M	**84.89**
Gemma-7B ($r = 32$)	LoRA	19.27M	71.79	19.27M	86.14
	MELoRA	19.27M	71.05	19.27M	83.67
	HiRA	19.27M	71.64	19.27M	86.16
	DoRA	19.61M	71.52	19.61M	85.15
	AuroRA	19.27M	71.23	19.27M	84.78
	GenLoRA	1.98M	**71.90**	1.29M	**86.74**
Qwen2.5-14B ($r = 32$)	LoRA	34.60M	80.09	34.60M	90.68
	MELoRA	34.60M	79.45	34.60M	89.75
	HiRA	34.60M	79.78	34.60M	90.89
	DoRA	34.95M	**80.42**	34.95M	**91.23**
	AuroRA	34.60M	80.17	34.60M	90.77
	GenLoRA	2.26M	80.36	2.26M	91.10

Adaptive rank allocation. For AdaLoRA (Zhang et al., 2023), the generated update can be expressed as $\sum_{i=1}^{r} b_i a_i^\top$, where each pair of generators supplies one rank component. Importance-based selection can in principle retain or prune these components across layers. A concrete integration must adapt the importance scores, singular-value gates, and orthogonality regularization to the generated factors; retaining the factorized form alone does not make the original allocation algorithm a drop-in replacement.

Task-aware initialization and allocation. Task-aware strategies such as TLoRA could determine layer-wise rank budgets, scaling, and target initial subspaces, while GenLoRA generates the associated basis vectors. Matching prescribed initial factors may require fitting the generators within their representational capacity. Such combinations offer an extension of the parameterization, but their optimization behavior and empirical benefits require separate evaluation.

G.7 Comparison with Basis-Combination Methods

NOLA (Koohpayegani et al., 2023) and HyperLoRA (Li et al., 2025a) construct updates through basis combinations, but differ in how the bases are parameterized: NOLA uses fixed random bases, whereas HyperLoRA uses trainable bases. GenLoRA instead synthesizes low-rank basis vectors from shared latent vectors using nonlinear RBF generators.

Table 11 compares GenLoRA with NOLA under comparable parameter budgets. GenLoRA achieves 70.17% average accuracy versus 65.15% for NOLA. This result supports the effectiveness of the proposed parameterization in the evaluated setting; it is not a direct empirical comparison with HyperLoRA or a proof of greater expressiveness.

Table 11: Comparison with NOLA under comparable parameter budgets using LLaMA-3-8B fine-tuned on Math10K. Avg. is the unweighted mean accuracy (%) across the six tasks.

Method	#Params	AddSub	MultiArith	SingleEq	SVAMP	GSM8K	AQuA	Avg.
NOLA	1.04M	82.73	83.34	85.48	65.80	50.70	22.83	65.15
GenLoRA ($r = 8, g = 16$)	**0.98M**	88.35	90.67	93.70	70.80	53.90	23.62	**70.17**

G.8 Generalization to Visual Modalities and Discriminative Architectures

To evaluate the generalizability of GenLoRA beyond text-based generative autoregressive language models, we extended our experiments to the visual modality and discriminative classification tasks. Specifically, we applied GenLoRA to encoder-only Vision Transformer (ViT) architectures (ViT-Base and ViT-Large) and evaluated them across eight standard image classification datasets.

As shown in Table 12, GenLoRA exhibits strong scalability and parameter efficiency on visual tasks. At an equivalent rank of $r = 8$, GenLoRA matches the performance of standard LoRA while utilizing approximately 25% of the parameters. More importantly, when allocated comparable parameter budgets (e.g., scaling GenLoRA to $r = 32$ for ViT-Base and $r = 64$ for ViT-Large), our method achieves significant performance improvements of 1.02 and 1.46 percentage points in average accuracy, respectively. These results firmly establish that the structural advantages of our generative adaptation approach are highly modality-agnostic, generalizing exceptionally well from natural language domains to diverse visual modalities and encoder-only discriminative architectures.

Table 12: Evaluation of GenLoRA on visual modalities and discriminative models (Vision Transformers) across 8 standard image classification datasets.

Model	Method	#Params	OxfordPet	StanfordC	CIFAR10	DTD	EuroSAT	FGVC	RESISC45	CIFAR100	Avg
ViT-Base	LoRA ($r = 8$)	0.29M	93.11	74.12	98.21	77.85	98.60	50.36	93.55	91.12	84.62
	GenLoRA ($r = 8, g = 8$)	0.09M	93.34	74.96	98.25	76.99	98.35	50.54	93.81	91.21	84.68
	GenLoRA ($r = 32, g = 8$)	0.23M	93.50	77.24	98.80	77.44	98.80	52.66	94.30	92.40	**85.64**
ViT-Large	LoRA ($r = 8$)	0.79M	94.12	84.40	98.50	77.93	98.60	58.42	93.68	93.25	87.36
	GenLoRA ($r = 8, g = 8$)	0.20M	93.87	84.58	98.64	79.34	98.63	57.89	93.84	93.89	87.59
	GenLoRA ($r = 64, g = 8$)	0.88M	94.88	85.99	99.10	81.45	98.94	59.76	95.43	94.98	**88.82**

H Additional Ablation Studies

In this section, we present additional micro-level ablation studies on the Math10K dataset using LLaMA-3-8B with GenLoRA ($r = 8, g = 16$) to validate our core architectural choices and hyperparameter settings.

H.1 Ablation on Hyperparameter K (RBF Centers)

We first evaluate the impact of the number of RBF centers, denoted as K. As shown in Table 13, $K = 15$ serves as a robust "sweet spot" for our architecture. While a lower value of K (e.g., $K = 5$ or 10) limits the mapping resolution of the generator, increasing K beyond 15 introduces additional parameters without yielding further performance gains, leading to diminishing returns.

Table 13: Ablation on the number of RBF centers (K) for GenLoRA ($r = 8, g = 16$) using LLaMA-3-8B fine-tuned on Math10K. Avg. is the unweighted mean accuracy (%) across the six tasks.

K	#Params	AddSub	MultiArith	SingleEq	SVAMP	GSM8K	AQuA	Avg.
5	0.74M	73.42	80.83	86.22	56.90	48.83	20.47	61.11
10	0.86M	78.70	83.73	89.83	63.80	50.78	23.87	65.12
15	0.98M	88.35	90.67	93.70	70.80	53.90	23.62	**70.17**
20	1.11M	87.15	90.83	93.63	69.30	52.42	23.26	69.43

H.2 Impact of Different RBF Types

We investigate the superiority of the Gaussian RBF compared to other commonly used radial basis functions. Theoretically, the Gaussian function excels due to its rapid exponential decay, which produces effectively localized responses despite its noncompact support. Let $d = \|x - \mu_k\|$ denote the distance to the grid center, and h denote the bandwidth parameter.

Empirical results in Table 14 are consistent with this interpretation: the Gaussian RBF achieves the highest accuracy (70.17%). Heavy-tailed kernels (IMQ, IQ) produce broader overlap, whereas

growing kernels (MQ, TPS) may couple responses across more distant centers. These locality differences offer a possible explanation for the measured accuracy gaps; the ablation does not isolate their causal effect on optimization.

Table 14: RBF kernel ablation for GenLoRA ($r = 8, g = 16$) using LLaMA-3-8B fine-tuned on Math10K. IMQ, IQ, MQ, and TPS denote inverse multiquadric, inverse quadric, multiquadric, and thin plate spline, respectively. Avg. is the unweighted mean accuracy (%) across the six tasks.

RBF Type	Function $\varphi(d)$	AddSub	MultiArith	SingleEq	SVAMP	GSM8K	AQuA	Avg.
Inverse Multiquadric (IMQ)	$\frac{1}{\sqrt{1+(d/h)^2}}$	86.06	88.31	90.78	69.20	52.38	26.54	68.88
Inverse Quadric (IQ)	$\frac{1}{1+(d/h)^2}$	86.74	87.36	92.08	69.60	54.29	26.33	69.40
Multiquadric (MQ)	$\sqrt{1+(d/h)^2}$	83.07	87.26	88.73	67.40	52.85	20.86	66.70
Thin Plate Spline (TPS)	$d^2 \ln(d)$	87.27	87.74	88.04	68.50	52.66	23.08	67.88
Gaussian (Ours)	$\exp(-(d/h)^2)$	88.35	90.67	93.70	70.80	53.90	23.62	**70.17**

H.3 Necessity of the Linear Residual Term

Our generator architecture employs a "Global + Local" approximation scheme. The base linear residual term is designed to capture global, low-frequency trends in the weight manifold, thereby freeing the RBF grid to focus on fitting high-frequency, fine-grained details. As demonstrated in Table 15, removing this base linear projection forces the RBF network to waste its representational capacity on general shifts, leading to a noticeable performance drop from 70.17% to 68.00%.

Table 15: Ablation on the linear residual term in GenLoRA ($r = 8, g = 16$) using LLaMA-3-8B fine-tuned on Math10K. Avg. is the unweighted mean accuracy (%) across the six tasks.

Method	#Params	AddSub	MultiArith	SingleEq	SVAMP	GSM8K	AQuA	Avg.
GenLoRA w.o. res	0.95M	81.77	85.83	91.93	66.50	55.19	26.77	68.00
GenLoRA	0.98M	88.35	90.67	93.70	70.80	53.90	23.62	**70.17**

H.4 Comparison of Different Generator Bases

We compare generator bases under comparable parameter budgets. For the Gaussian component $F(t) = \sum_{k=1}^{K} w_k \varphi_k(t)$, the derivative bound in Appendix B yields

$$|F(t) - F(t')| \leq \frac{\sqrt{2/e}}{h} \|w\|_1 |t - t'|. \tag{40}$$

Thus bandwidth and coefficient magnitude explicitly control sensitivity. The mapping is nonlinear in its input but linear in its K coefficients; Gaussian responses are smooth, decay away from their centers, and admit vectorized evaluation. Chebyshev bases have global support, with derivative bounds that can grow quadratically with degree on $[-1, 1]$, whereas B-spline smoothness depends on degree and knot multiplicity. These properties motivate Gaussian RBFs for compact basis-vector synthesis, without implying universal superiority over other generators.

Table 16 provides the corresponding Math10K comparison. At $g = 16$, Chebyshev, B-spline, and Gaussian RBF achieve 64.01%, 64.54%, and 70.17%; at $g = 32$, they achieve 66.63%, 68.04%, and 70.11%, respectively. Gaussian RBF performs best in both reported settings, complementing the structural and computational analysis.

H.5 Ablation Study on Latent Vectors

In GenLoRA, the latent vectors Z_A and Z_B serve as the input "seeds" for the RBF generators. A critical question is whether these vectors actively encode task-specific information or merely act as static anchors for the non-linear mapping. To investigate this, we conducted an ablation study on the

Table 16: Comparison of generator bases under comparable parameter budgets using LLaMA-3-8B fine-tuned on Math10K. All GenLoRA configurations use $r = 8$. Avg. is the unweighted mean accuracy (%) across the six tasks.

Group	Generator Basis	#Params	AddSub	MultiArith	SingleEq	SVAMP	GSM8K	AQuA	Avg.
$g = 16$	Chebyshev	0.95M	74.52	85.47	88.13	61.20	54.60	20.12	64.01
	B-spline	1.06M	76.71	86.33	88.58	59.80	55.34	20.47	64.54
	Gaussian RBF (Ours)	0.98M	88.35	90.67	93.70	70.80	53.90	23.62	**70.17**
$g = 32$	Chebyshev	1.22M	81.15	85.47	90.13	64.30	55.31	23.41	66.63
	B-spline	1.52M	83.54	86.50	90.75	66.70	52.77	27.95	68.04
	Gaussian RBF (Ours)	1.38M	88.61	89.67	90.35	70.40	57.24	24.41	**70.11**

LLaMA-3-8B model using the Math10K benchmark. We selectively froze the latent vectors during training while keeping the generator parameters tunable, using the GenLoRA configuration with $r = 8$ and $g = 16$ as the baseline. In these frozen-latent experiments, Z_B receives Kaiming initialization before freezing, as does Z_A. Zero initialization applies to the B-side generator coefficients rather than to the latent vector itself, so the initial generated matrix remains $B_0 = 0$.

Table 17 presents the experimental results. The standard GenLoRA, where both latent vectors are learnable, achieves an average accuracy of 70.17% with 0.98M parameters. In contrast, when we freeze both latent vectors (Z_A, Z_B), the performance drastically drops to 60.63%, a decrease of 9.54 percentage points. This indicates that the generator functions alone are insufficient to capture the full adaptation dynamics; the optimization of the input latent space is crucial. Freezing only Z_A results in an accuracy of 62.19%. Since A determines the projection into the low-rank subspace, its rigidity likely prevents the model from extracting optimal input features. Similarly, freezing only Z_B yields an accuracy of 61.26%. As B is responsible for projecting back to the high-dimensional space, fixing its latent representation severely limits the flexibility of the update direction.

Notably, although freezing latents reduces the parameter count (e.g., to 0.39M), the trade-off in performance is severe. These findings confirm that the learnability of latent vectors is a fundamental component of GenLoRA, allowing the model to dynamically adjust the topological input to the RBF generators to synthesize optimal rank components.

Table 17: Ablation on latent-vector learnability using LLaMA-3-8B fine-tuned on Math10K. All GenLoRA configurations use $r = 8, g = 16$. "Frozen" indicates latent vectors not updated during training; #Params counts trainable parameters. Avg. is the unweighted mean accuracy (%) across the six tasks.

Method	#Params	AddSub	MultiArith	SingleEq	SVAMP	GSM8K	AQuA	Avg.
LoRA ($r = 8$)	4.72M	82.28	85.83	89.76	66.40	56.18	23.23	67.28
GenLoRA ($r = 8, g = 16$)	0.98M	88.35	90.67	93.70	70.80	53.90	23.62	**70.17**
GenLoRA (Frozen Z_A, Z_B)	0.39M	74.32	82.06	83.76	62.30	46.32	15.03	60.63
GenLoRA (Frozen Z_A)	0.59M	75.01	84.28	86.71	60.90	48.39	17.83	62.19
GenLoRA (Frozen Z_B)	0.79M	73.10	84.27	84.98	61.10	48.26	15.85	61.26

I Extended Scalability Analysis and Parameter Efficiency

In our main experiments, we configured GenLoRA with a rank of $r = 8$ to maintain an "Iso-Rank" alignment with standard baselines such as LoRA and DoRA. Additionally, we included a setting of $r = 32$ to validate our core hypothesis: that GenLoRA enables the utilization of significantly higher ranks to unlock superior performance, all while maintaining a highly compact parameter footprint. Even at $r = 32$, GenLoRA requires only 2.16M parameters, which is less than half of the standard LoRA at $r = 8$ (4.72M), demonstrating its capability to trade minimal parameters for substantial rank increases.

We did not prioritize a strict "Iso-Parameter" comparison in the main results for two primary reasons. First, due to the structural differences between explicit matrix factorization and our generative approach, precise parameter alignment is geometrically constrained by integer hyperparameters such

as rank and group size, making exact matching computationally infeasible without altering the model architecture. Second, and more importantly, the primary objective of this work is to demonstrate that GenLoRA can achieve superior performance by efficiently scaling up the rank with significantly *fewer* parameters than standard methods, rather than merely competing at equal parameter counts.

To further investigate the scalability of our approach and validate that GenLoRA can effectively leverage a larger parameter budget, we conducted an extended experiment scaling the rank to $r = 64$. As shown in Table 18, we present the performance trajectory across ranks $r = 8$, 32, and 64, with the group size consistently set to $g = 16$.

Remarkably, even at the high-rank configuration of $r = 64$, GenLoRA requires only 3.74M parameters, which is still approximately 20% fewer than the standard LoRA at $r = 8$ (4.72M). In terms of performance, GenLoRA exhibits a consistent upward trajectory as the rank increases. This confirms that GenLoRA successfully decouples parameter cost from rank, allowing for scalable improvements in model capacity without the prohibitive memory overhead typically associated with high-rank adaptation.

Table 18: Extended comparison on Mathematical Reasoning tasks (LLaMA-3-8B) across varying ranks and parameters.

Method	#Params	AddSub	MultiArith	SingleEq	SVAMP	gsm8k	AQuA	Avg
LoRA($r = 8$)	4.72M	82.28	85.83	89.76	66.40	56.18	23.23	67.28
GenLoRA($r = 8$)	0.98M	88.35	90.67	93.70	70.80	53.90	23.62	70.17
GenLoRA($r = 32$)	2.16M	87.09	92.71	94.49	70.80	56.18	25.59	71.14
GenLoRA($r = 64$)	3.74M	89.36	92.50	92.82	72.10	56.88	24.97	**71.44**

J Detailed Numerical Results for Singular Value Energy

Table 19 provides the layer-wise values underlying the singular value analysis in Figure 4(b), covering the self-attention query-projection updates across all 32 layers. The reported spectral energy is

$$E(\Delta W) = \sum_i \sigma_i(\Delta W)^2 = \|\Delta W\|_F^2. \tag{41}$$

It measures squared update magnitude, rather than the expressiveness of the parameterization.

GenLoRA has larger update magnitudes than LoRA in the reported runs. For example, at layer 0, GenLoRA ($r = 8$, 0.98M parameters) has spectral energy 38.03, whereas LoRA ($r = 32$, 18.87M parameters) has energy 0.28. This is approximately a $136\times$ difference in squared Frobenius norm, or an $11.7\times$ difference in Frobenius norm. The observation shows that fewer trainable parameters need not imply smaller learned updates in this setting. Update magnitude also depends on scaling and optimization; spectral energy alone cannot demonstrate greater expressiveness, more efficient use of rank, or better generalization.

K Experiment Details

K.1 Dataset Details

In this section, we provide detailed descriptions of the benchmarks and datasets used to evaluate the Natural Language Generation (NLG) and Code Generation capabilities of GenLoRA.

Mathematical Reasoning We evaluate the mathematical reasoning capability of our model using the **Math10K** benchmark (Hu et al., 2023). This benchmark consists of a curated training corpus and requires the model to perform multi-step arithmetic and logical reasoning. We assess performance across the following six sub-tasks:

1. **AddSub** (Hosseini et al., 2014): A dataset of arithmetic word problems focusing on addition and subtraction operations.

Table 19: Layer-wise spectral energy of the learned query-projection updates, $\sum_i \sigma_i^2 = \|\Delta W\|_F^2$. Values measure squared update magnitude. The two panels report layers 0–15 and 16–31.

Method	Config (Params)	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15
LoRA	$r = 8$ (4.72M)	0.09	0.15	0.15	0.16	0.19	0.24	0.21	0.18	0.21	0.17	0.23	0.22	0.32	0.32	0.31	0.36
	$r = 16$ (9.44M)	0.14	0.26	0.25	0.26	0.31	0.33	0.33	0.35	0.30	0.28	0.36	0.38	0.52	0.49	0.45	0.60
	$r = 32$ (18.9M)	0.28	0.42	0.52	0.47	0.49	0.57	0.58	0.58	0.61	0.50	0.63	0.64	0.81	0.77	0.78	1.05
GenLoRA(Ours)	$r = 8$ (0.98M)	38.0	33.4	24.4	36.3	20.1	27.2	32.8	24.6	26.6	30.9	30.8	28.8	30.7	40.9	32.7	46.9
	$r = 32$ (2.16M)	198.6	113.5	93.1	114.7	90.3	104.9	105.9	86.3	103.9	96.6	104.2	103.0	119.9	113.0	100.8	128.1
	$r = 64$ (3.74M)	386.8	232.5	228.4	230.5	212.5	227.8	216.5	230.5	207.7	199.6	217.4	205.7	205.7	229.8	233.8	246.7

Method	Config (Params)	L16	L17	L18	L19	L20	L21	L22	L23	L24	L25	L26	L27	L28	L29	L30	L31
LoRA	$r = 8$ (4.72M)	0.40	0.46	0.46	0.42	0.51	0.35	0.43	0.40	0.44	0.34	0.39	0.47	0.51	0.42	0.48	0.35
	$r = 16$ (9.44M)	0.59	0.70	0.74	0.64	0.75	0.58	0.66	0.73	0.74	0.57	0.70	0.71	0.78	0.70	0.75	0.70
	$r = 32$ (18.9M)	0.89	1.10	1.12	0.99	1.16	0.98	1.15	1.07	1.09	0.97	1.09	1.03	1.21	1.23	1.21	0.97
GenLoRA(Ours)	$r = 8$ (0.98M)	37.1	54.7	53.0	57.5	57.2	46.4	49.7	55.6	36.2	39.5	54.1	44.1	66.8	75.8	55.5	40.7
	$r = 32$ (2.16M)	121.0	161.8	164.7	143.1	142.5	139.1	138.5	152.7	140.1	138.3	158.7	163.7	135.0	226.2	173.4	131.3
	$r = 64$ (3.74M)	255.7	334.8	294.6	293.5	264.9	312.2	286.1	319.9	287.0	287.7	332.6	297.0	320.4	426.9	326.8	256.5

2. **MultiArith** (Roy & Roth, 2016): A dataset designed to test the model's ability to solve multi-step arithmetic problems involving various operations.
3. **SingleEq** (Koncel-Kedziorski et al., 2015): Comprises algebra word problems that map to single linear equations.
4. **SVAMP** (Patel et al., 2021): A challenge dataset created by applying variations to existing word problems to test robustness against linguistic perturbations.
5. **GSM8K** (Cobbe et al., 2021): A dataset of high-quality, linguistically diverse grade school math word problems requiring multi-step chain-of-thought reasoning.
6. **AQuA** (Ling et al., 2017): A large-scale dataset of algebra word problems with multiple-choice options, requiring complex reasoning and derivation.

Commonsense Reasoning We evaluate our model on the **Commonsense170K** benchmark (Hu et al., 2023), which aggregates multiple datasets for training and evaluation. The evaluation covers the following eight sub-tasks:

1. **BoolQ** (Clark et al., 2019): A binary question-answering task where the goal is to determine whether the answer to a question about a given passage is "yes" or "no."
2. **PIQA** (Physical Interaction Question Answering) (Bisk et al., 2020): Focuses on reasoning about physical commonsense to select the most plausible solution to a given problem.
3. **SIQA** (Social IQa) (Sap et al., 2019): Tests social commonsense reasoning by asking questions about motivations, reactions, or outcomes in social contexts.
4. **HellaSwag** (Zellers et al., 2019): A task designed to test contextual commonsense reasoning by selecting the most plausible continuation of a given scenario.
5. **WinoGrande** (Sakaguchi et al., 2021): A pronoun coreference resolution task that requires reasoning over ambiguous pronouns in complex sentences.
6. **ARC-e** (AI2 Reasoning Challenge - Easy) (Clark et al., 2018): A multiple-choice question-answering task focused on elementary-level science questions.
7. **ARC-c** (AI2 Reasoning Challenge - Challenge) (Clark et al., 2018): A more difficult subset of ARC, containing questions that require advanced reasoning and knowledge retrieval.
8. **OBQA** (OpenBookQA) (Mihaylov et al., 2018): A question-answering task requiring reasoning and knowledge synthesis from a provided "open book" of science facts.

Code Generation We assess the code generation capability of GenLoRA by fine-tuning on the **Magicoder-Evol-Instruct-110k** dataset (Wei et al., 2023) and evaluating on the HumanEval+ benchmark.

1. **Training Data (Magicoder-Evol-Instruct-110k)**: A curated and decontaminated subset of WizardCoder (Luo et al., 2023). It comprises approximately 110k high-quality instruction-response pairs developed via the Evol-Instruct method, designed to enhance the complexity and diversity of programming tasks.
2. **Evaluation Benchmark (HumanEval+)**: An extended version of the HumanEval benchmark used to rigorously test functional correctness in code generation. We follow the standard evaluation protocol via the BigCode Evaluation Harness (Allal et al., 2022), generating 50 sampled completions per problem ($n = 50$) and reporting **Pass@1**, **Pass@5**, and **Pass@10** accuracy scores.

K.2 Baseline Details

In this section, we briefly describe the baseline methods compared in our experiments:

- **DoRA** (Liu et al., 2024a) decomposes pretrained weights into magnitude and direction components, optimizing the magnitude vector while utilizing LoRA for direction updates.
- **MELoRA** (Ren et al., 2024) introduces mini-ensemble low-rank adapters that collectively achieve high-rank expressive power.
- **HiRA** (Huang et al., 2025) utilizes the Hadamard product to enable high-rank adaptation, thereby maintaining parameter efficiency without sacrificing model capacity.
- **AuroRA** (Dong et al., 2025) addresses the low-rank bottleneck by introducing nonlinear mappings within the adapter architecture to enhance expressivity.

K.3 Implementation Details

Details of Motivation Experiment. To ensure fairness and generality, we conducted this analysis on standard, publicly available pretrained LoRA checkpoints. We selected the `hfl/llama-3-chinese-8b-instruct-lora` adapter (based on Meta Llama-3-8B) as our representative data source. Specifically, we extracted the `lora_A` and `lora_B` weight matrices corresponding to the Self-Attention Query Projection (`q_proj`) from the first transformer layer to serve as the ground truth for our experiments. To reconstruct these basis vectors, we formulated a regression task where the RBF generators map a shared prototype vector derived by averaging the extracted rows to the specific target basis vectors. The training objective was to minimize the element-wise Mean Squared Error (MSE) between the synthesized vectors and the ground truth. Optimization was performed using the Adam optimizer with a learning rate of 1×10^{-3}. The training process spanned 2,000 epochs on a single NVIDIA GPU, with no weight decay applied, to ensure rigorous convergence to the target weight patterns.

Initialization Strategy. To preserve the pretrained model's behavior at the onset of fine-tuning, GenLoRA uses an asymmetric initialization with $\Delta W_0 = 0$: (1) **For Matrix** A: The latent vector Z_A is initialized using Kaiming Uniform initialization (He et al., 2015) to introduce diverse initial variance. Correspondingly, the RBF generator weights (Θ_A) are initialized from a normal distribution scaled by $1/\sqrt{d_{\text{group}}}$. Crucially, the weights for each of the r generator functions are independently sampled from a normal distribution, ensuring that the synthesized basis vectors are distinct and non-degenerate. (2) **For Matrix** B**:** The latent vector Z_B is initialized with element-wise random values rather than zeros. All B-side base-activation coefficients $w^B_{b,i,g}$ and RBF combination coefficients $w^B_{i,g,k}$ are initialized to zero and remain trainable. Thus, even with nonzero within-group variation in the latent inputs, the generated matrix satisfies $B_0 = 0$, ensuring $\Delta W_0 = B_0 A_0 = 0$ and preserving the pretrained model's initial output.

Hyperparameters and Experimental Setup All experiments were conducted on NVIDIA RTX A6000 GPUs. The training setup utilized the AdamW optimizer (Loshchilov & Hutter, 2017) with linear learning rate decay, a LoRA dropout rate of 0.05, and no weight decay. For the code generation task, we employed a learning rate of 2×10^{-4}. Regarding the reasoning tasks, we set the learning rate to 1×10^{-4} for mathematical reasoning and 5×10^{-3} for commonsense reasoning. Across all tasks, GenLoRA and the baseline methods were applied to the query, key, and value weights, training for one epoch. The warmup ratio is 0.06 for the reasoning tasks, while code generation uses 100

warm-up steps. Our implementation builds upon the code from (Hu et al., 2023). Regarding the RBF generator configuration, we set the number of uniform centers to $K = 15$ distributed over the fixed interval $[-3, 3]$ to align with the instance-wise normalized inputs. Accordingly, the bandwidth h is explicitly defined by the grid spacing as $h = \frac{6}{K-1}$. In our comparative analysis, we primarily aligned the baseline methods to a rank of 8 to maintain a comparable parameter budget. Specifically, LoRA, DoRA, and HiRA were evaluated at rank 8. For MELoRA, we adhered to the standard setting of rank 8 with four mini-ensemble groups. Regarding AuroRA, we conducted tests at both rank 2 and rank 8 to specifically investigate its capability in mitigating the low-rank bottleneck through nonlinear mappings. For our proposed GenLoRA, we evaluated various configurations with differing ranks and group sizes. It is worth noting that GenLoRA consistently requires significantly fewer parameters than the rank 8 baselines; notably, some configurations involve fewer parameters than even the rank 2 AuroRA setting, while achieving superior performance.

Summary of Hyperparameters. Tables 20 and 21 summarize the default reasoning settings. The ranks and group counts for individual comparisons are specified in the corresponding result tables. Table 22 summarizes the code-generation settings.

Table 20: Default training settings for GenLoRA on Math10K. The reasoning backbones are LLaMA-3-8B, Gemma-7B, and Qwen2.5-14B; additional ranks are specified in the corresponding result tables.

Hyperparameter	Setting
Optimizer	AdamW
Learning rate	1e-4
Weight decay	0.0
Warmup ratio	0.06
Learning-rate schedule	Linear
Epochs	1
Default adapter rank	8
Target modules	Q, K, V
Training precision	BF16
# GPUs per run	1
Effective training batch size	16
Maximum sequence length	256

Table 21: Default training settings for GenLoRA on Commonsense170K. The reasoning backbones are LLaMA-3-8B, Gemma-7B, and Qwen2.5-14B; additional ranks are specified in the corresponding result tables.

Hyperparameter	Setting
Optimizer	AdamW
Learning rate	5e-3
Weight decay	0.0
Warmup ratio	0.06
Learning-rate schedule	Linear
Epochs	1
Default adapter rank	8
Target modules	Q, K, V
Training precision	BF16
# GPUs per run	1
Effective training batch size	16
Maximum sequence length	256

Table 22: Training and evaluation settings for GenLoRA on code generation.

Hyperparameter	Setting
Backbone	LLaMA-3-8B
Training dataset	Magicoder-Evol-Instruct-110k
Optimizer	AdamW
Learning rate	2e-4
Weight decay	0.0
Warmup steps	100
Learning-rate schedule	Linear
Epochs	1
Adapter dropout	0.05
GenLoRA rank	8, 32
Group count	16
Target modules	Q, K, V
RBF centers (K)	15
RBF grid interval	$[-3, 3]$
RBF bandwidth (h)	$6/(K-1) = 3/7$
GPU model	NVIDIA RTX A6000
Evaluation benchmark	HumanEval+
Evaluation framework	BigCode Evaluation Harness
Sampled completions per problem	50
Metrics	Pass@1, Pass@5, Pass@10