MATRYOSHKALORA: LEARNING ACCURATE HIERARCHICAL LOW-RANK REPRESENTATIONS FOR LLM FINE-TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

With the rise in scale for deep learning models to billions of parameters, the computational cost of fine-tuning remains a significant barrier to deployment. While Low-Rank Adaptation (LoRA) has become the standard for parameter-efficient fine-tuning, the need to set a predefined, static rank r requires exhaustive grid searches to balance efficiency and performance. Existing rank-adaptive solutions such as DyLoRA mitigate this by sampling ranks during the training from a predefined distribution. However, they often yield sub-optimal results at higher ranks due to lack of consistent gradient signals across the full hierarchy of ranks, thus making these methods data-inefficient. We propose MATRYOSHKALORA, a general, Matryoshka-inspired training framework for LoRA-like techniques that learns accurate hierarchical low-rank representations by inserting a fixed, carefully crafted diagonal matrix P between the existing LoRA adapters to scale their sub-ranks accordingly. By introducing this simple modification, our general framework recovers LoRA and DyLoRA only by changing P and ensures all sub-ranks embed the available gradient information efficiently. Our MATRYOSHKALORA supports dynamic rank selection with minimal degradation in accuracy and show it is general enough by applying it to other LoRA-inspired techniques, such as DoRA and PiSSA. We theoretically justify a potential train–test mismatch by bounding the average optimality gap across the supported ranks, guaranteeing that the learned hierarchy remains close, on average, to independently optimized rank-specific adapters during inference. We further propose Area Under the Rank Accuracy Curve (AURAC), a metric that consistently evaluates the performance of hierarchical low-rank adapters. Our results demonstrate that MATRYOSHKALORA learns more accurate hierarchical low-rank representations than prior rank-adaptive approaches and achieves superior accuracy-performance trade-offs across ranks.

1 INTRODUCTION & RELATED WORK

The research advancements in the Deep Learning community allowed researchers and practitioners to train models with billions of parameters. Building and managing pipelines for such a complex system requires a significant engineering effort, from preparing the dataset to scaling the model and the optimizer states via multi-dimensional parallelization techniques across many GPUs in a cluster or even multiple physical clusters - a process spanning over months. Given the costs, these large models are rarely deployed to production-ready environments as they are; instead, they serve as a knowledge base for downstream adaptation, such as fine-tuning, which still remains computationally prohibitive for many applications and settings.

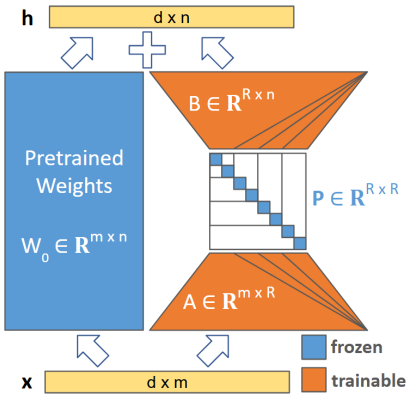


Figure 1: MATRYOSHKALORA

To alleviate this overhead, LoRA Hu et al. (2021) has emerged as the de-facto standard for parameter-efficient fine-tuning (PEFT). However, it introduces a significant structural constraint: the rank r must be predefined. Find-

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.

ing the optimal balance between parameter efficiency and model performance currently requires exhaustive search across multiple training runs, each with a different static rank.

One line of work studies adaptive-rank LoRA methods, which optimize how rank capacity is allocated across layers or modules under a given or learned parameter budget. These methods improve parameter efficiency, but they typically produce a single specialized adapter configuration rather than a nested adapter family that can be sliced into at inference time Zhang et al. (2023b); Ding et al. (2023); Liu et al. (2024b); Zhang et al. (2024; 2023a); Singh et al. (2025); Cui et al. (2026); Kumaravelu et al. (2026).

A second direction studies dynamic-rank LoRA methods, which aim to train a single adapter whose prefixes are independently usable as lower-rank adapters. Unlike adaptive allocation methods, these approaches are designed to provide multiple sub-adapters from one checkpoint Valipour et al. (2023); Luo et al. (2025). A conceptually related, but not directly comparable, line of work is slimmable Yu et al. (2018) and once-for-all networks Cai et al. (2020). These are not LoRA methods, but they provide the broader once-for-all training paradigm: a single set of weights is trained so that many nested sub-networks are valid deployment choices Yu & Huang (2019).

Our work can be interpreted as translating this paradigm from channel width to LoRA rank and belongs primarily to the second cluster. It differs from adaptive-rank allocation methods because it does not merely identify one rank configuration; it trains a nested adapter family (prefixes). It differs from general slimmable networks because the nested structure is imposed on low-rank adapter factors rather than full-model channels. Its closest comparison is DyLoRA Valipour et al. (2023), which also targets dynamic rank usage but enforces rank hierarchy through a different training mechanism discussed next.

DyLoRA Valipour et al. (2023) is an alternative solution to grid-searching for the rank r for LoRA. At each training step, a rank $k \in \mathbb{N}$ is randomly sampled from a predefined distribution and the forward pass is performed using only the first k rows and first k columns of adapters A and B, respectively, denoted by A_k and B_k. The loss is computed with respect to the output of the network that was generated using only A_k and B_k for all linear layers. The purpose of this approach is to train adapters A and B that still yield high accuracy when using only a slice of size k on the fly. DyLoRA solves the issue of adaptive rank r only partially because, as we will show in our experimental section, the accuracies of the models fine-tuned with DyLoRA do not exhibit a true hierarchical pattern in the context of reasoning tasks, such as math datasets. *We empirically show that DyLoRA strategy is sub-optimal because it constrains the learning only to a fixed rank k sampled at random, while the ranks $r > k$ do not receive any gradient signal for that particular optimization step.*

Our work proposes Matryoshka-style training framework for LoRA, inspired from Matryoshka Representation Learning Kusupati et al. (2024), which we call MATRYOSHKALORA. It is motivated by the drawback of DyLoRA, which fails to learn what we call hierarchical low-rank features. Instead of using only one randomly generated rank k per forward pass, we use all possible slices A_r and B_r, with $r \in \{1, 2, 4, ..., R\}$, where R is the maximum rank of A and B. This way, the lower-dimensional representations are contained in higher-dimensional representations as prefixes, thus building a nested hierarchy. We present the schematic of MATRYOSHKALORA in Figure 1.

We see three advantages of having an accurate technique to train hierarchical low-rank adapters: (1) it eliminates the need for multiple training runs to grid-search for different ranks, implicitly translating to a reduction in costs; (2) we benefit from a high-performance adapter at each rank r by deploying targeted ranks on different devices depending on their computational power and (3) we enable dynamic rank selection under varying cluster loads to serve the requests at the same rate with minimal accuracy drop.

Contribution. We summarize our contribution as follows:

- We introduce a general framework inspired by Matryoshka Representation Learning for training LoRA adapters whose lower-rank prefixes remain accurate and independently usable at inference time, which views LoRA, DyLoRA, and Matryoshka-style adapters as different parameterizations of a shared rank-weighting vector;
- Starting from the goal of learning hierarchical low-rank representations at every layer, we show that this naturally leads to a simple diagonal weighting between the standard LoRA adapters A and B, making the hierarchy explicit while keeping the implementation and exposition close to standard LoRA;

- We provide a theoretical justification for the potential train–test mismatch by bounding the average optimality gap across the supported ranks. We show that the difference between our surrogate objective and independently optimized rank-constrained adapters is controlled by a "rank dispersion" term, and that under standard scaling this term vanishes asymptotically, yielding a vanishing average optimality gap across the supported ranks;
- We propose **MATRYOSHKALORA**, a concrete instance of this framework that learns accurate nested low-rank prefixes via the new diagonal matrix. The resulting adapter can be evaluated or deployed at multiple ranks from a single checkpoint, without retraining separate adapters;
- We introduce **Area Under the Rank Accuracy Curve (AURAC)**, a metric for evaluating adapter performance across a set of ranks. AURAC summarizes the rank-performance tradeoff while weighting each rank according to its magnitude, reflecting the expectation that larger ranks should generally achieve stronger performance.

2 METHOD

In this section, we introduce the notation we will use throughout the paper and summarize LoRA and DyLoRA, our main baselines from the literature, which we use to justify our MATRYOSHKALORA approach to learn accurate hierarchical low-rank representations for fine-tuning. We show the generality of our framework by applying it to DoRA and PiSSA and introduce AURAC, the metric we propose to assess the performance of LoRA-like approaches when evaluated on multiple ranks.

2.1 NOTATION AND BASELINES

Notation. We consider the pre-trained weights of a fully connected layer $W_0 \in \mathbb{R}^{m\times n}$ and the LoRA adapters $A \in \mathbb{R}^{m\times R}$ and $B \in \mathbb{R}^{R\times n}$, where R is the maximum possible rank of A and B, which is usually a power of 2. Given the adapters A and B and an integer k, we extract the subsets of these adapters, denoted by $A_k = A[:, 1:k]$ (first k columns of A) and $B_k = B[1:k, :]$ (first k rows of B) by indexing into the R-dimension of each adapter.

We denote by $\mathcal{S}$ the set of ranks used for training/inference. If not specified otherwise, the set $\mathcal{S}$ is the same for both training and inference. Note that in our work we restrict $\mathcal{S}$ to contain only powers of two to be in line with most of the settings that already exist, as the ranks for LoRA are also chosen to be powers of 2.

Baselines. Given the input activation $x \in \mathbb{R}^{d\times m}$ for the current layer, the forward pass for LoRA is $Y = x(W_0 + s_R AB)$, where $s_R \in \{1, 1/R, 1/\sqrt{R}\}$ is a scaling factor. For DyLoRA, we need to randomly sample a rank k from $\mathcal{S}$ and use the truncated adapters A_k and B_k to obtain the forward pass for DyLoRA as $Y = x(W_0 + s_k A_k B_k)$, where $s_k = 1/k$.

2.2 MATRYOSHKALORA

The goal of our MATRYOSHKALORA approach is to obtain accurate hierarchical low-rank representations for different ranks inside the same LoRA adapters A and B such that the accuracy drop for ranks $r < R$ is minimized, as the loss will contain contributions from all ranks $r \in \mathcal{S}$. We aim to update each adapter prefix A_k and B_k using gradient signal computed for the current batch of data. *In contrast, DyLoRA uses the gradient from the current batch to update only the first k columns/rows, while the rest $R - k$ do not receive any update, making it data-inefficient.* MATRYOSHKALORA includes all adapter prefixes A_r, B_r into the forward pass $Y = x\left(W_0 + \sum_{r\in\mathcal{S}} s_r A_r B_r\right)$. In Equation 4 in Section 3 and Appendix A we explain how we simplify the forward pass of MATRYOSHKALORA by replacing the sum across all ranks $r \in \mathcal{S}$ with a carefully crafted diagonal matrix P between A and B to reduce the computational overhead to a minimum, thus resulting in the forward pass $Y = x(W_0 + APB)$.

2.3 EVALUATION METRICS

Given a model fine-tuned with a LoRA-inspired technique and a set of evaluation ranks $\mathcal{S}$, we define the set of accuracies $A_\mathcal{S} = \{a_k \mid k \in \mathcal{S}\}$ obtained when evaluating the model with rank $k \in \mathcal{S}$ using the forward pass $Y = x(W_0 + s_k \cdot A_k B_k)$. For example, if the adapters A and B have the maximum rank $R = 16$ and we choose the evaluation ranks $k \in \mathcal{S} = \{1, 2, 4, 8, 16\}$ on a specific dataset $\mathcal{D}$,

we obtain a set of fine-tuning accuracies $A_{\mathcal{S}} = \{a(1), a(2), a(4), a(8), a(16)\}$, where $a(k)$ is the accuracy on dataset $\mathcal{D}$ for rank k.

Given $\mathcal{S}$ and $A_{\mathcal{S}}$, we denote r_i as the i-th rank in $\mathcal{S}$ and $a(r_i)$ as the accuracy obtained with rank r_i, we compute two versions for an aggregate metric called the Area Under the Rank Accuracy Curve (AURAC), which employs the trapezoidal rule:

$$\text{AURAC} = \frac{1}{r_{|\mathcal{S}|} - r_1} \sum_{i=1}^{|\mathcal{S}|-1} \frac{a(r_i) + a(r_{i+1})}{2} \cdot (r_{i+1} - r_i) \tag{1}$$

$$\text{log-AURAC} = \frac{1}{\log_2(r_{|\mathcal{S}|}) - \log_2(r_1)} \sum_{i=1}^{|\mathcal{S}|-1} \frac{a(r_i) + a(r_{i+1})}{2} \cdot (\log_2(r_{i+1}) - \log_2(r_i)) \tag{2}$$

The AURAC metric takes into account the distance between consecutive ranks in the set of training ranks $\mathcal{S}$. For our example mentioned above, the accuracy area between ranks 8 and 16 will have the largest proportion in the resulting AURAC metric, equivalent to $(16-8)/(16-1) = 8/15 = 53.3\%$, which would bias the final AURAC value towards the accuracy of the largest interval between ranks 8 and 16, which is in line with the expectation that the LoRA adapters should learn hierarchical, low-rank features: a large value for AURAC metric means all intermediary ranks achieve high accuracy.

The log-AURAC metric is designed only for the case when $\mathcal{S}$ contains ranks that are power of two, which would result in all ranks being equally distant, therefore all intervals will contribute uniformly to the log-AURAC metric. For our example, all intervals will be weighted by $1/4 = 25\%$. For the datasets we tested on, we haven't seen significant differences between AURAC and log-AURAC. However, we report the values of both metrics in our experimental section.

2.4 Extending Matryoshka Framework to Other LoRA Variants

In this section, we explain how we can apply our MATRYOSHKA framework to other low-rank adaptation techniques, such as DoRA Liu et al. (2024a) and PiSSA Meng et al. (2025).

DoRA is a PEFT technique that inherits from LoRA and decomposes the pre-trained weights into magnitude and direction components. Instead of updating weights via a purely additive low-rank matrix $W' = W + AB$, the original weight matrix is separated into a one-dimensional magnitude vector and a 2D directional matrix. During fine-tuning, DoRA trains the magnitude vector directly while applying the low-rank updates (A and B) exclusively to the directional matrix. To create MatryoshkaDoRA, we add the scaling matrix P between the adapters A and B in exactly the same way as in MATRYOSHKALORA.

PiSSA is also a PEFT technique that improves upon LoRA by initializing the trainable adapters A and B with the principal spectral components of the pre-trained weights computed via SVD. This replaces the standard LoRA initialization of adapters with random noise and zeros, respectively. The remaining, non-principal spectral components are frozen as the base residual matrix. During fine-tuning, only the principal-initialized A and B matrices are updated. Next, we explain how we apply our Matryoshka framework to PiSSA because the SVD decomposition makes it slightly more complicated than LoRA and DoRA.

Formally, let the initial, pre-trained matrix be $W_0 = U\Sigma V^\top$. For a maximum adapter rank R, we initialize the adapters as $A^{(0)} = U_R\Sigma_R^{1/2}$ and $B^{(0)} = \Sigma_R^{1/2}V_R^\top$, therefore we have $A^{(0)}B^{(0)} = U_R\Sigma_R V_R^\top$. For a given rank $r \leq R$, we use our prefix projector P_r to obtain $A^{(0)}P_rB^{(0)} = W_{pri}(r)$, which results in $W_{res}(r) = W_0 - A^{(0)}P_rB^{(0)}$. During training, when we update factors A and B, we have the effective weight $W_r(A,B) = W_{res}(r) + AP_rB = W_0 + (AP_rB - A^{(0)}P_rB^{(0)})$. We can easily verify that at initialization we obtain $W_r(A^{(0)}, B^{(0)}) = W_0$.

To implement MatryoshkaPiSSA (M-PiSSA), we define the rank r adaptation as $\Delta_r(A,B) = s_r(AP_rB - A^{(0)}P_rB^{(0)})$ and therefore the M-PiSSA training is given by $W_{\text{M-PISSA}}(A,B) = W_0 + \sum_{r\in\mathcal{S}} \lambda_r s_r(AP_rB - A^{(0)}P_rB^{(0)}) = W_0 + (APB - A^{(0)}PB^{(0)})$, where $P = \sum_{r\in\mathcal{S}} \lambda_r s_r P_r$ is our aggregated prefix projector. During inference, we discard the aggregation matrix P and for a given rank k the M-PiSSA weights are given by $W_k = W_0 + s_k(AP_kB - A^{(0)}P_kB^{(0)})$, which is

equivalent to $W_k = W_{res}(k) + s_k A_k B_k$ with $W_{res}(k) = W_0 - s_k A_k^{(0)} B_k^{(0)}$. For more details about how we set the diagonal projector P_k, see Equation 4 in Section 3 and Appendix A.

3 THEORETICAL VIEW OF MULTI-RANK OPTIMIZATION

Given a fine-tuning loss ℓ, the corresponding optimization problem for LoRA is given by the objective $A^*, B^* = \min_{A,B} \ell(W_0 + s_R AB)$. This formulation ties fine-tuning to the chosen rank R. After training, truncating A to its first $r < R$ columns and B to its first r rows *does not generally yield a competitive rank-r adapter, because the standard objective does not encourage the factors to be useful in prefix order*.

Let R be the maximum supported rank and let $\mathcal{S} = \{r_1, \ldots, r_K\}$, with $1 \le r_1 < \cdots < r_K = R$, be the set of ranks to be supported by a single pair of factors. For each $r \in \mathcal{S}$, we use the truncated factors A_r and B_r for the associated rank-r loss $L_r(A, B) = \ell(W_0 + s_r A_r B_r)$. Letting $f(\Delta) = \ell(W_0 + \Delta)$, a natural objective for training all supported ranks jointly is following the weighted multi-rank objective:

$$L_{\text{multi}}(A, B) = \textstyle\sum_{r\in\mathcal{S}} \lambda_r L_r(A, B) = \sum_{r\in\mathcal{S}} \lambda_r \ell(W_0 + s_r A P_r B) = \sum_{r\in\mathcal{S}} \lambda_r f(\Delta_r), \quad (3)$$

with normalized weights $\lambda_r > 0, \sum_{r\in\mathcal{S}} \lambda_r = 1$, diagonal projector P_r onto the first r coordinates and perturbation $\Delta_r = s_r A_r B_r = s_r A P_r B$. DyLoRA with non-frozen updates samples rank r with probability λ_r, providing unbiased estimates of this objective and its gradient. MATRYOSHKALORA instead evaluates the surrogate $L_{\text{surr}}(A, B) = f(\bar{\Delta})$ in one forward pass, using

$$\bar{\Delta} = \textstyle\sum_{r\in\mathcal{S}} \lambda_r \Delta_r = APB, \qquad P = \sum_{r\in\mathcal{S}} \lambda_r s_r P_r. \quad (4)$$

3.1 AVERAGE OPTIMALITY GAP

Assume f is lower bounded, differentiable, and has an L-Lipschitz gradient in the Frobenius norm. Define the *rank dispersion* $\delta_{\mathcal{S}}(A, B) = \sum_{r\in\mathcal{S}} \lambda_r \|\Delta_r - \bar{\Delta}\|_{\text{F}}^2$ and the optimal loss for an independently chosen perturbation of rank at most r, $f_r^* = \inf_{\text{rank}(\Delta)\le r} f(\Delta)$. Smoothness gives $|L_{\text{multi}} - L_{\text{surr}}| \le (L/2)\delta_{\mathcal{S}}(A, B)$ for every A, B (Appendix B, Theorem B.3). Moreover, P is invertible, so APB represents every perturbation of rank at most R and $\inf_{A,B} L_{\text{surr}}(A, B) = f_R^*$.

Theorem 3.1 (Average optimality gap)**.** *Under the smoothness assumption above, suppose the learned factors $(\widehat{A}, \widehat{B})$ satisfy $L_{\text{surr}}(\widehat{A}, \widehat{B}) \le f_R^* + \varepsilon$ for some $\varepsilon \ge 0$. Then, with $\widehat{\Delta}_r = s_r \widehat{A} P_r \widehat{B}$,*

$$\textstyle\sum_{r\in\mathcal{S}} \lambda_r \big[f(\widehat{\Delta}_r) - f_r^*\big] \le \varepsilon + \frac{L}{2}\delta_{\mathcal{S}}(\widehat{A}, \widehat{B}). \quad (5)$$

Thus, surrogate optimization error and learned rank dispersion jointly control performance relative to independently optimized adapters; The proof is given in Appendix B, Theorem B.4.

Structural control of rank dispersion. The learned dispersion obeys $\delta_{\mathcal{S}}(A, B) \le \|A\|_2^2 \|B\|_2^2 \delta(\mathcal{S})$, where $\|\cdot\|_2$ is the spectral norm and $\delta(\mathcal{S}) = \sum_{r\in\mathcal{S}} \lambda_r \|s_r P_r - P\|_{\text{F}}^2$ depends only on the rank support, normalized weights, and scaling. A complementary bound using the energies of the rank-one components, together with exact support formulas, appears in Appendix B, Section B.3.

For uniform weights $\lambda_r = 1/K$ and $K \to \infty$ with r_1 and $\alpha > 0$ fixed, *standard scaling* $s_r = \alpha/r$ gives $\delta(\mathcal{S}) = \Theta(\alpha^2 \log K/K)$ for consecutive ranks $r_i = r_1 + i - 1$, and $\Theta(\alpha^2/(r_1 K))$ for dyadic ranks $r_i = 2^{i-1} r_1$: *both vanish*. Fixed scaling $s_r = \alpha$ instead gives $\Theta(\alpha^2 K)$ and $\Theta(\alpha^2 R/K)$, respectively, while rank-stabilized scaling $s_r = \alpha/\sqrt{r}$ yields the nonzero limits $\alpha^2/3$ and α^2, respectively. For standard scaling, a uniform bound on $\|\widehat{A}\|_2^2 \|\widehat{B}\|_2^2$ as $K \to \infty$ therefore implies that the learned dispersion in equation 5 vanishes.

3.2 GENERAL FRAMEWORK FOR RECOVERING EXISTING LORA APPROACHES

Our Matryoshka framework is general enough to obtain both LoRA and DyLoRA by changing the matrix P. To recover LoRA, we have to choose $P = \text{diag}(1/R \cdot \mathbf{1}_R)$, a square diagonal matrix that contains only the values $1/R$. To recover DyLoRA for a sampled rank k from $\mathcal{S}$, we have to choose $P = \text{diag}(1/k \cdot [\underbrace{1, \cdots, 1}_{k \text{ values}}, \underbrace{0, \cdots, 0}_{R-k \text{ values}}])$, where only the first k diagonal entries are equal to $1/k$.

3.3 Gradient Computation for MatryoshkaLora

In this section we explain how inserting the constant matrix diag(P) between A and B scales the gradients for A and B, even though in the implementation we multiply the columns of A with elements of vector P for efficiency.

Let $W \in \mathbb{R}^{m\times n}$ be the pretrained layer and $\Delta \in \mathbb{R}^{m\times n}$ be the upstream gradient for the current linear layer, for which we compute the gradients ∇_A and ∇_B for A and B, respectively. Then, the gradients used in the optimizer for A and B have the following expressions: $\nabla_A = \Delta \cdot B^\top \cdot \text{diag}(P) \in \mathbb{R}^{m\times R}$ and $\nabla_B = \text{diag}(P) \cdot A^\top \cdot \Delta \in \mathbb{R}^{R\times n}$. Given the structure of diag(P) described in previous sections, the equations for the gradients of adapters A and B illustrate how the matrix diag(P) scales the gradients ∇_A and ∇_B, thus learning the hierarchical, low-rank representations we are targeting in MatryoshkaLora.

4 Experimental Results

Setting. To prove the effectiveness of our MatryoshkaLora in learning hierarchical low-rank features, we fine-tune Llama-3.2-1B-Instruct and Llama-3.1-8B-Instruct on GSM-8k. Due to the high computational demands of exhaustive multi-rank evaluation, we focus our experiments on this challenging mathematical reasoning benchmark.

Ranks used. We reiterate that the LoRA adapters in our work have shapes $A \in \mathbb{R}^{m\times R}$ and $B \in \mathbb{R}^{R\times n}$, where R is the maximum rank of adapters, which we ablate over in our experiments to assess the quality of each sub-rank. We choose the set $\mathcal{S}$ that contains all powers of 2 up to the maximum rank R (such as 32 or 256). Then, the ranks used for training and evaluation are determined by the set $\mathcal{S}_R = \{1, 2, 4, ..., R\}$ (all powers of 2).

Adapter Scaling. In our experiments we use adapter scaling $\lambda_r = 1$ and $s_r = 1$ for simplicity. We have not seen significantly different results between $s_r = 1/r$ and $s_r = 1$ for MatryoshkaLora. The choice of s_r affects the best performing learning rate. Because of this, we use a large enough grid for all methods, which makes the choice of s_r redundant.

Hyper-params & Adapter Type. We consider the best performing runs the ones achieving largest average log-AURAC metric across all seeds, reported for each particular rank. Averaging is performed across 3 seeds. Due to computational constraints, we report standard deviations only for smaller-scale experiments; for large-scale runs, we omit them and restrict our sweep to learning rate tuning. We use 3 epochs for GSM-8k in our study. To keep the compute costs for the evaluation low, we mention the grid we used to ablate over the learning rate in each section separately. We use the default AdamW Loshchilov & Hutter (2017) optimizer for fine-tuning without weight decay.

Overhead. Since our approach adds one simple operation (multiplying the columns of adapter A with vector P), the required resources in terms of running time and memory of MatryoshkaLora are identical to LoRA and DyLoRA.

4.1 Fine-Tuning Llama-3.2-1B-Instruct on GSM-8K for different ranks R

The first set of experiments focuses on fine-tuning Llama-3.2-1B-Instruct Grattafiori et al. (2024) on GSM-8k Cobbe (2021) using LoRA, DyLoRA and MatryoshkaLora. We fix the maximum rank to individual values $R \in \{1, 2, 4, 8, 16, 32\}$ and use the ranks in the set $\mathcal{S}_R = \{r : r \leq R \wedge r = 2^k\}$ for training. We use 8-shot evaluation via lm-evaluation-harness Gao et al. (2023).

We perform grid search over learning rates $\eta \in \{1e-5, 2e-5, 4e-5, 6e-5, 8e-5\}$ and report the results in Table 1. For the discussion that follows, we mention that the baseline accuracy of the pre-trained model (before any fine-tuning) is 34.7%.

Rank-wise Accuracy. Our MatryoshkaLora dominates across the entire rank spectrum. While LoRA and DyLoRA exhibit relatively modest gains as rank increases, typically pleateauing in the mid 34% range, our approach scales more effectively with rank and exhibits a significant increase in evaluation accuracy, starting with $R = 2$. At higher ranks, such as $R \in \{16, 32\}$, MatryoshkaLora achieves accuracies up to 39% and 38%, respectively, clearly surpassing the best LoRA ($\approx 35\%$) and DyLoRA ($\approx 35.3\%$).

We would like to emphasize that the evaluation accuracy obtained for the sub-rank $r = 1$ obtained when trained with bottleneck size $R = 32$ is already higher than any other sub-rank for LoRA and DyLoRA. When we compare against the evaluation accuracy of 34.7% for the pre-trained model, we observe that fine-tuning with LoRA and DyLoRA does not improve accuracy significantly. By deploying rank 16 would lead to better performance compared to the pre-trained model by up to 5%.

AURAC. Our metric provides a clear advantage also in terms of (log-)AURAC. Both LoRA and DyLoRA achieve peak (log-)AURAC values of up to 35%, whereas MATRYOSHKALORA reaches 38.4%, representing a substantial improvement. This gap highlights both stronger peak performance and more consistent gains across all ranks.

Key takeaway. Selecting each rank r from MATRYOSHKALORA trained with global rank $R = 32$ (the last row of Table 1) yields higher accuracy compared to any sub-rank chosen from either LoRA or DyLoRA.

Table 1: [Results for Section 4.1] AURAC (A) and log-AURAC (log-A) metrics and per-rank, evaluation accuracies for Llama-3.2-1B-Instruct on GSM-8k (8-shot). We emphasize the accuracy of pre-trained model (before finetuning) of 34.7%.

ADAPTER TYPE	EVAL ACCURACY PER RANK						A	LOG-A
	1	2	4	8	16	32		
LORA	$32.4_{\pm 1.2}$						32.4	32.4
	$32.4_{\pm 1.1}$	$33.6_{\pm .5}$					33.0	33.0
	$33.5_{\pm .2}$	$34.8_{\pm .4}$	$34.3_{\pm .1}$				34.4	34.3
	$34.2_{\pm .6}$	$34.1_{\pm .6}$	$34.6_{\pm .5}$	$34.9_{\pm .4}$			34.6	34.4
	$33.0_{\pm .4}$	$33.9_{\pm .5}$	$34.1_{\pm .3}$	$34.8_{\pm .3}$	$34.8_{\pm .4}$		34.5	34.2
	$33.3_{\pm 1.1}$	$34.3_{\pm .5}$	$34.3_{\pm .2}$	$34.3_{\pm .5}$	$34.5_{\pm .2}$	$34.6_{\pm .6}$	34.4	34.3
DYLORA	$32.4_{\pm 1.3}$						32.4	32.4
	$32.9_{\pm .2}$	$33.5_{\pm .3}$					33.2	33.2
	$33.4_{\pm .4}$	$34.1_{\pm .1}$	$34.3_{\pm .5}$				34.1	34.0
	$34.6_{\pm .1}$	$34.5_{\pm .7}$	$34.0_{\pm .4}$	$34.5_{\pm .7}$			34.3	34.4
	$32.9_{\pm .5}$	$33.9_{\pm .9}$	$34.6_{\pm .9}$	$34.8_{\pm .4}$	$34.8_{\pm .2}$		34.6	34.3
	$33.2_{\pm .9}$	$34.7_{\pm .3}$	$35.2_{\pm .4}$	$34.8_{\pm .5}$	$35.3_{\pm .2}$	$33.9_{\pm .8}$	34.8	34.7
MATRYOSHKALORA	$32.2_{\pm 1.2}$						32.2	32.2
	$35.7_{\pm .8}$	$35.4_{\pm .6}$					**35.5**	**35.5**
	$36.1_{\pm .2}$	$36.7_{\pm .8}$	$35.9_{\pm .3}$				**36.3**	**36.4**
	$34.3_{\pm .9}$	$36.2_{\pm .7}$	$36.8_{\pm .6}$	$36.4_{\pm .8}$			**36.4**	**36.1**
	$34.4_{\pm .6}$	$35.9_{\pm .0}$	$36.7_{\pm .2}$	$36.9_{\pm 1.3}$	$36.9_{\pm .1}$		**36.7**	**36.3**
	$35.5_{\pm .7}$	$36.1_{\pm .4}$	$37.4_{\pm 1.1}$	$38.5_{\pm .6}$	$39.4_{\pm .2}$	$38.4_{\pm .9}$	**38.6**	**37.7**

4.2 FINE-TUNING LLAMA-3.1-8B-INSTRUCT ON GSM-8K

The second set of experiments focuses on fine-tuning Llama-3.1-8B-Instruct Grattafiori et al. (2024) on GSM-8k Cobbe (2021) using the same LoRA, DyLoRA and MATRYOSHKALORA, evaluated using 3/8-shots strategies. We experiment with maximum ranks $R \in \{32, 256\}$ and use the ranks in the sets $\mathcal{S}_R = \{r : r \leq R \wedge r = 2^k, k \in \mathbb{N}\}$ (all powers of 2 smaller than R) for both training and evaluation via the same lm-evaluation-harness Gao et al. (2023).

We use a large grid to find the best learning rate $\eta \in \{a \cdot 10^{-b}, a \in \{1, 2, 4, 6, 8\}, b \in \{3, 4, 5, 6\}\}$ and report the results Table 2. We mention the accuracy of the pre-trained model (before any fine-tuning) for the 3/8-shot evaluation in the first column of the table. For readability, we report only mean accuracies here; across three seeds, the run-to-run variation was of similar magnitude to that observed in Table 4.1.

3-shot setting. Our method consistently outperforms both LoRA and DyLoRA in terms of (log-) AURAC metrics. For $R = 32$, the evaluation accuracy for the two baselines fluctuates in the range $74 - 75\%$ and MATRYOSHKALORA achieves higher and more stable per-rank performance, reaching up to 78.6% for rank $r = 128$, an improvement of over 4% compared to both pre-trained model and any rank prefix of the baselines. Our method improves at low ranks for $R = 32$ and does not exhibit a big accuracy drop for higher ranks, as it is the case for the two baselines LoRA and

DyLoRA. The improvements in per-rank accuracies translate to the (log-)AURAC score, where MATRYOSHKALORA reaches more than 3% compared to baselines.

8-shot setting. The gap between the baselines and MATRYOSHKALORA remains consistent, reaching 80% accuracy for ranks $r \in \{32, 64, 128\}$. However the gap for both per-rank accuracy and (log-) AURAC gap decreases. Our explanation is that more in-context information helps learning overall and shadows the influence of the adapters.

Drops from 128 to 256. In both 3/8-shot settings, there is a drop in accuracy when going from rank 128 to 256 for both LoRA and MATRYOSHKALORA. However, MATRYOSHKALORA alleviates this issue in contrast to LoRA for $R = 256$.

Key takeaway. MATRYOSHKALORA closes the gap between 3-shot and 8-shot: if the goal is to achieve 77% accuracy with 3-shots, one should pick the adapters trained with MATRYOSHKALORA for $R = 256$ and use $r = 32$ or $r = 64$ with 3-shots. In principle, it would translate to shorter contexts compared to 8-shots and therefore lower inference costs.

Table 2: [Results for Section 4.2] AURAC (A) and log-AURAC (log-A) metrics and per-rank evaluation accuracies for GSM-8k (3/8-shot) for $R \in \{32, 256\}$ for Llama-3.1-8B-Instruct on GSM-8k. PT stands for the accuracy of the **P**re-**T**rained model (before fine-tuning) with N-shots.

N-SHOTS PT ACC	MAX RANK	ADAPTER TYPE	EVAL ACCURACY PER RANK									A	LOG-A
			1	2	4	8	16	32	64	128	256		
3-SHOT	32	LORA	74.1	74.5	74.2	74.3	74.1	73.8	-	-	-	74.1	74.2
		DYLORA	74.3	74.9	74.3	74.1	74.0	73.8	-	-	-	74.1	74.3
		MATRYOSHKA	74.8	76.9	77.3	77.6	77.5	77.6	-	-	-	**77.5**	**77.1**
PT: 74.0	256	LORA	73.8	74.2	74.9	75.7	76.9	76.8	78.0	77.2	67.7	74.8	75.5
		DYLORA	72.8	73.4	74.0	73.8	74.3	71.8	70.4	69.8	70.4	70.6	72.4
		MATRYOSHKA	73.5	73.5	74.7	75.0	75.9	76.8	77.6	78.6	76.4	**77.4**	**75.9**
8-SHOT	32	LORA	77.8	77.9	77.5	77.6	77.0	76.9	-	-	-	77.2	77.5
		DYLORA	78.4	78.8	78.5	79.0	78.8	79.2	-	-	-	78.9	78.8
		MATRYOSHKA	77.6	78.6	79.6	80.1	79.9	79.8	-	-	-	**79.8**	**79.4**
PT: 77.8	256	LORA	78.6	78.4	79.1	78.6	78.5	78.7	78.9	79.5	79.1	79.1	78.8
		DYLORA	78.5	78.5	77.9	78.6	78.5	78.4	78.8	79.5	79.8	79.2	78.7
		MATRYOSHKA	77.9	77.7	79.2	79.1	79.8	80.2	80.5	80.2	78.1	**79.7**	**79.3**

4.3 INFLUENCE OF RANKS SET $\mathcal{S}$ OVER TRAINING STABILITY

In this section we show that using all ranks from 1 to R in the set of training ranks $\mathcal{S}$ does not destabilize the training. We fine-tune Llama-3.1-8B-Instruct on GSM-8k with 8-shot evaluation using MATRYOSHKALORA (M-LoRA) with rank 32. We impose the set of training ranks $\mathcal{S}$ to contain all ranks, leading to the matrix $P_{all} = diag([32, 31, \cdots, 3, 2, 1])$ or only the powers of two, thus leading to the largest element in the matrix P to be $1 + \log_2(32) = 6$. We use a wide learning rate grid with 20 values $\{a \cdot 10^{-b} | a \in \{1, 2, 4, 6, 8\}, b \in \{3, 4, 5, 6\}\}$ and 3 seeds. We evaluate all runs on the ranks that are power of two and report the results in Table 3, which shows the best values for the individual rank accuracies and the joint (log-)AURAC metrics.

MATRYOSHKALORA with powers of 2 achieves higher accuracies at smaller ranks and exhibits a small accuracy drop for the largest rank. However, when we use all ranks, the accuracies for smaller ranks are lower than for the powers of two case and they are monotonically increasing with the rank. Note there is no drop anymore for the largest rank. Given these observations related to per-rank accuracies discussed here, we would like to emphasize that the differences in (log-)AURAC metrics are not significant. In our experiments we always run MATRYOSHKALORA with both choices of $\mathcal{S}$ (all ranks vs powers of two) and report the best performing runs by log-AURAC metric.

4.4 MATRYOSHKA APPLIED TO DORA AND PISSA

DoRA. Standard DoRA exhibits a significant accuracy drop at the maximum rank ($r = 32$), falling to 32.9% and demonstrating instability compared to its mid-rank performance, as seein in the upper half

Table 3: Evaluation accuracies and AURAC (A) / log-AURAC (log-A) metrics comparing the effect of ranks set $\mathcal{S}$ (all vs. 2^k) over training stability for MATRYOSHKALORA (M-LoRA) for Llama-3.1-8B-Instruct fine-tuned on GSM-8k and evaluated with 8-shot strategy.

ADAPTER	$\mathcal{S}$	EVAL ACCURACY PER RANK						A	LOG-A
		1	2	4	8	16	32		
M-LORA	2^k	79.1 $\pm.2$	79.5 $\pm.3$	79.7 $\pm.3$	79.4 $\pm.3$	79.8 $\pm.3$	76.7 $\pm.5$	78.9	79.2
M-LORA	ALL	78.1 $\pm.4$	78.3 $\pm.2$	79.3 $\pm.6$	79.2 $\pm.4$	79.4 $\pm.4$	79.9 ±1.0	79.4	79.0

of Table 4. MATRYOSHKADORA (M-DoRA) resolves this degradation, showing a steady increasing trend in accuracy as rank increases. It peaks at 39.0% at $r = 16$ and maintains strong performance (37.7%) at the maximum rank ($\mathcal{S}$ contains ranks 2^k here). Because of this stable rank-wise scaling, MATRYOSHKADORA outperforms standard DoRA across the full rank spectrum, yielding a $+2.0\%$ improvement in AURAC and a $+0.9\%$ improvement in log-AURAC.

PiSSA. Unlike standard LoRA or DoRA, standard PiSSA maintains robust performance across all ranks without degrading at $r = 32$, peaking at 78.7% at $r = 16$, as shown in the lower half of Table 4. MATRYOSHKAPISSA (M-PiSSA) performs comparably to standard PiSSA, but yields marginally lower overall metrics, trailing by 0.2% in AURAC and 0.1% in log-AURAC. Because PiSSA initializes weights using SVD, inherently ordering components by importance, the nested regularization introduced by the Matryoshka structure does not provide additional scaling benefits and acts as a slight constraint on the adapter's capacity. Our intuition is that the SVD already introduces a hierarchical structure into the adapters, however at a higher computational and memory cost than our Matryoshka framework, as it has to store the residual components.

Table 4: Comparison between standard DoRA and M-DoRA for $R = 32$: evaluation accuracies and AURAC (A) / log-AURAC (log-A) metrics for GSM-8k using Llama-3.2-1B-Instruct.

MODEL	ADAPTER	EVAL ACCURACY PER RANK						A	LOG-A
		1	2	4	8	16	32		
LLAMA 3.2-1B	DORA	35.0 $\pm.5$	35.8 $\pm.2$	37.5 $\pm.5$	37.7 $\pm.3$	37.4 $\pm.7$	32.9 ±1.0	36.2	36.5
	M-DORA	35.0 $\pm.4$	36.2 $\pm.5$	37.0 $\pm.6$	38.5 $\pm.2$	39.0 $\pm.6$	37.7 $\pm.3$	38.2	37.4
LLAMA 3.1-8B	PISSA	77.3 $\pm.3$	77.9 $\pm.6$	78.4 $\pm.5$	78.1 $\pm.2$	78.7 $\pm.3$	77.8 $\pm.3$	78.3	78.1
	M-PISSA	77.5 $\pm.3$	77.8 $\pm.2$	78.1 $\pm.3$	78.5 $\pm.5$	78.1 $\pm.4$	78.0 $\pm.5$	78.1	78.0

5 CONCLUSION AND LIMITATIONS

Conclusion. We propose a Matryoshka-inspired framework applied to LoRA by inserting a carefully crafted diagonal matrix P between adapters A and B that enables learning hierarchical, low-rank features inside the same adapters as embedded prefixes. We demonstrate the effectiveness of this simple approach on two Llama models on the popular GSM-8k dataset. We show our framework is general by recovering LoRA and DyLoRA by simply changing the expression of the vector P. In addition, we propose a metric to assess the performance of a LoRA-like model when evaluated on multiple ranks to facilitate choosing the best performing model across all evaluation ranks during the hyper-parameter tuning process.

Limitations. Despite this method being simple and having the same overhead as LoRA, the development of this work requires many model evaluations for each rank, which increases the actual runtime to assess the performance. Therefore, we believe our results can be improved by further hyper-parameter tuning, specifically a larger grid for the learning rate, different number of epochs or weight decay. Our current evaluation is restricted to the GSM-8k benchmark with a uniform rank k across all layers due to computational constraints. Although extending this to coding and commonsense reasoning tasks remains for future work, GSM-8k's complexity provides strong evidence that our framework achieves its goal of learning hierarchical low-rank features during fine-tuning.

AI USE STATEMENT

In this work, we used generative AI tools to edit large tables and rephrase our statements to fit the page limit. We have not used generative AI tools for generating ideas, write code or paper sections from scratch. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All our experiments are performed in single-GPU setting on Nvidia H100 GPUs with 80GB of RAM under PyTorch v2.8.0, `lm-eval` v0.4.9.1, `transformers` v4.57.6 and `datasets` v3.6.0. The seeds we use throughout the paper are $42, 666, 2408$. We provide the code in the supplementary material and use open-source models and datasets.

REFERENCES

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment, 2020. URL https://arxiv.org/abs/1908.09791.

Karl et al. Cobbe. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Xuan Cui, Huiyue Li, Run Zeng, Yunfei Zhao, Jinrui Qian, Wei Duan, Bo Liu, and Zhanpeng Zhou. Igu-lora: Adaptive rank allocation via integrated gradients and uncertainty-aware scoring, 2026. URL https://arxiv.org/abs/2603.13792.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. Sparse low-rank adaptation of pre-trained language models, 2023. URL https://arxiv.org/abs/2311.11696.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes

Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta,

Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Vishnuprasadh Kumaravelu, Sunil Gupta, and P. K. Srijith. Post-optimization adaptive rank allocation for lora, 2026. URL https://arxiv.org/abs/2604.27796.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. Matryoshka representation learning, 2024. URL https://arxiv.org/abs/2205.13147.

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, 2024a. URL https://arxiv.org/abs/2402.09353.

Zequan Liu, Jiawen Lyn, Wei Zhu, Xing Tian, and Yvette Graham. Alora: Allocating low-rank adaptation for fine-tuning large language models, 2024b. URL https://arxiv.org/abs/2403.16187.

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

Dan Luo, Kangfeng Zheng, Chunhua Wu, Xiujuan Wang, and Jvjie Wang. Erat-dlora: Parameter-efficient tuning with enhanced range adaptation in time and depth aware dynamic lora. *Neurocomputing*, 614:128778, 2025. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2024.128778. URL https://www.sciencedirect.com/science/article/pii/S0925231224015492.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models, 2025. URL https://arxiv.org/abs/2404.02948.

Raul Singh, Nicolo Brunello, Vincenzo Scotti, and Mark James Carman. L1ra: Dynamic rank assignment in lora fine-tuning, 2025. URL https://arxiv.org/abs/2509.04884.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation, 2023. URL https://arxiv.org/abs/2210.07558.

Jiahui Yu and Thomas Huang. Universally slimmable networks and improved training techniques, 2019. URL https://arxiv.org/abs/1903.05134.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks, 2018. URL https://arxiv.org/abs/1812.08928.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning, 2023a. URL https://arxiv.org/abs/2308.12043.

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, 2023b. URL https://arxiv.org/abs/2303.10512.

Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning, 2024. URL https://arxiv.org/abs/2403.09113.

APPENDIX

CONTENTS

1 Introduction & Related Work **1**

2 Method **3**

2.1 Notation and Baselines 3

2.2 MATRYOSHKALORA 3

2.3 Evaluation Metrics 3

2.4 Extending Matryoshka Framework to Other LoRA Variants 4

3 Theoretical view of multi-rank optimization **5**

3.1 Average optimality gap 5

3.2 General Framework for Recovering Existing LoRA Approaches 5

3.3 Gradient Computation for MATRYOSHKALORA 6

4 Experimental Results **6**

4.1 Fine-Tuning Llama-3.2-1B-Instruct on GSM-8K for different ranks R 6

4.2 Fine-Tuning Llama-3.1-8B-Instruct on GSM-8K 7

4.3 Influence of Ranks Set $\mathcal{S}$ over Training Stability 8

4.4 Matryoshka Applied to DoRA and PiSSA 8

5 Conclusion and Limitations **9**

A Simplification of MATRYOSHKALORA Forward Pass **14**

A.1 How to compute P? 15

A.2 Practical Example for Small Sized Adapters 15

B Theory **16**

B.1 Theorem & Proof for the MatryoshkaLoRA Surrogate Objective 16

B.2 Average optimality gap 17

B.3 Structural control of rank dispersion 17

A SIMPLIFICATION OF MATRYOSHKALORA FORWARD PASS

The forward pass for MATRYOSHKALORA cannot be implemented as it is in PyTorch because the framework does not allow propagating gradients only through a slice of parameter. Therefore, we must use the full adapters A and B and mask the gradients accordingly:

$$Y = x\left(W_0 + \sum_{r\in S} s_r \cdot (A \odot M_r^A)(B \odot M_r^B)\right) \tag{6}$$

where M_r^A and M_r^B are binary masks of the same shape as A and B where we set to 1 only the first r columns and first r rows of M^A and M^B, respectively. For example, for $m = n = R = 3$ and $S = \{1, 2\}$, we would have the following masks:

$$M_1^A = \begin{pmatrix}1 & 0 & 0\\ 1 & 0 & 0\\ 1 & 0 & 0\end{pmatrix}; M_2^A = \begin{pmatrix}1 & 1 & 0\\ 1 & 1 & 0\\ 1 & 1 & 0\end{pmatrix}; M_1^B = \begin{pmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}; M_2^B = \begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 0 & 0 & 0\end{pmatrix}.$$

If we ignore the overhead of the element-wise multiplications between adapters A and B and their corresponding masks M^A and M^B, then the Equation 6 would require $|S|$ matmuls for each layer with inner dimensions $r \in S$, compared to one matmul for LoRA and DyLoRA with inner dimensions R and k, which is clearly an undesired overhead. During training, the boolean masks M_r^A and M_r^B must be stored and applied to the gradients computed for A and B at each forward pass. Even though they are boolean values, storing two masks per linear layer is against the simplicity of LoRA. Next, our goal is to simplify the formulation of MATRYOSHKALORA. A careful inspection of the forward pass of MATRYOSHKALORA suggests there exist two matrices C_A, C_B with the same shape as A and B such that:

$$(A \odot C_A)(B \odot C_B) = \sum_{r\in S}(A \odot M_r^A)(B \odot M_r^B) \tag{7}$$

In other words, we do not have to store individual boolean masks $M_r^{A,B}$ for $r \in S$ and potentially share these masks among layers whenever the dimensions allow. Instead, we can simply use two matrices C_A, C_B per layer instead of $2 \cdot |S|$ matrices, which completely remove the need for masks $M_r^{A,B}$, as well as the need the loop in the forward pass of MATRYOSHKALORA. At this stage, we removed the boolean masks $M_r^{A,B}$ and we are left with the floating point matrices C_A, C_B, which still increase memory usage, even when stored in half precision. We go one step further towards simplifying the forward pass for MATRYOSHKALORA and observe that C_A, C_B can be replaced with a diagonal matrix P. We provide more details in Appendix A. Therefore, we obtain the simplest form as:

$$(A \odot C_A)(B \odot C_B) = A \cdot \text{diag}(P) \cdot B = (A * P) \cdot B, \tag{8}$$

where P is an R-dimensional vector. Note we do not need to explicitly create the matrix $diag(P) \in \mathbb{R}^{R\times R}$. Instead, we can simply multiply the row i of A by P_i to obtain the same effect at the lowest possible cost. The vector P is the same for all layers and therefore the memory overhead of our MATRYOSHKALORA is only the additional vector P with R elements. The final formulation of the forward pass for MATRYOSHKALORA is based on the observation that there exists an R-dimensional vector P_k such that $A_k \cdot B_k = A \cdot \text{diag}(P_k) \cdot B$, with $P_k = \text{diag}(\underbrace{1, \cdots, 1}_{k \text{ values}}, \underbrace{0, \cdots, 0}_{R-k \text{ values}})$ such that:

$$Y = xW = x\left(W_0 + (A * P)B\right), \quad \text{with} \quad P = \sum_{r\in S} s_r \cdot P_r \tag{9}$$

Figure 1 shows the schematic of MATRYOSHKALORA: the diagonal matrix P is used during the training using the forward pass in Equation 9; during evaluation, we discard the matrix P, choose a specific rank k and perform dynamic inference according to the default forward pass of LoRA. In Appendix B we provide a theoretical view of our formulation.

A.1 How to compute P?

In Algorithm 1 we show the steps to create the vector P with R components. For each rank r from R down to 1, we use the integer c to count how many times the sub-rank k is employed in the sum in the forward pass of MatryoshkaLora. After that, the value for P at component r (denoted by p_r) is obtained as the sum of s_{r_k}, where r_k is the rank at index k in set $\mathcal{S}$ containing training ranks, as shown in Equations 10, 11, 12 and 13 as an example. Note that the summation loops through indices of $\mathcal{S}$ indexed from 1 to $|S|$ and not its elements. For example, $R = 8$ and $S = \{1, 2, 4, 8\}$ will generate the vector P with the following components p_i:

$$p_1 = s_1 + s_2 + s_4 + s_8 \quad (10)$$
$$p_2 = s_2 + s_4 + s_8 \quad (11)$$
$$p_3 = p_4 = s_4 + s_8 \quad (12)$$
$$p_5 = p_6 = p_7 = p_8 = s_8 \quad (13)$$

Algorithm 1 Algorithm to compute vector P in Equation 8 for our MatryoshkaLora

inputs:
- $\mathcal{S}$ - set of training ranks
- R - maximum rank of A, B

$P \leftarrow 0_R$ // R zeros
$c \leftarrow 0$
for $r = R$ down to 1 **do**
  if $r \in S$ **then**
    $c \leftarrow c + 1$
  end if
  $p_r \leftarrow \sum_{k=|S|-c}^{|S|} s_{r_k}$
end for
return $P \in \mathbb{R}^R$

Specifically, p_r will have contributions from all ranks r that appear as subscripts in its expression above. When we use $s_r = 1, \forall r \in S$, we will obtain $P = [4, 3, 2, 2, 1, 1, 1, 1]$, which has the effect of scaling the first column of A and first row of B by 4 and so on, thus increasing the contribution of the gradient. Observe the last components of P are all 1, meaning that the columns/rows $5, 6, 7, 8$ from A and B respectively will get gradient signal only from the rank-8 component. In short, the vector P embeds the global contribution for each column/row in the forward pass of MatryoshkaLora. The vector P induces the desired effect of learning hierarchical low-rank features in our MatryoshkaLora.

A.2 Practical Example for Small Sized Adapters

In this section we explain how we simplify the expression of MatryoshkaLora presented in Section 2.2, where we choose scaling $s_r = 1$ for simplicity. Let's take an example of two LoRA adapters, where $A \in (3, 3)$ and $B \in (3, 3)$, from which we want to use ranks $r \in \{1, 2, 3\}$ and denote by A_r, B_r the first r columns for A and first r rows for B.

$$A = A_3 = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \quad A_1 = \begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix} \quad A_2 = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}$$

$$B = B_3 = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} \quad B_1 = (b_{11} \quad b_{12} \quad b_{13}) \quad B_2 = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix}$$

We want to find matrices C_A, C_B of the same shape as A and B such that $(A \odot C_A)(B \odot C_B) = \sum_r A_r B_r$. First, let's look at the result of multiplications $A_r \cdot B_r$ for each $r \in \{1, 2\}$ by explicitly writing the scalar products between rows of A_r and columns of B_r

$$A_1 \cdot B_1 = \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} \\ a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} \\ a_{31}b_{11} & a_{31}b_{12} & a_{31}b_{13} \end{pmatrix} \quad A_2 \cdot B_2 = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} & a_{21}b_{13} + a_{22}b_{23} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} & a_{31}b_{13} + a_{32}b_{23} \end{pmatrix}$$

$$A_3 \cdot B_3 = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} & a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} & a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} & a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} \end{pmatrix}$$

$$\sum_{r \in \{1,2,3\}} A_r B_r = \begin{pmatrix} \mathbf{3}a_{11}b_{11} + \mathbf{2}a_{12}b_{21} + \mathbf{1}a_{13}b_{31} & \mathbf{3}a_{11}b_{12} + \mathbf{2}a_{12}b_{22} + \mathbf{1}a_{13}b_{32} & \mathbf{3}a_{11}b_{13} + \mathbf{2}a_{12}b_{23} + \mathbf{1}a_{13}b_{33} \\ \mathbf{3}a_{21}b_{11} + \mathbf{2}a_{22}b_{21} + \mathbf{1}a_{23}b_{31} & \mathbf{3}a_{21}b_{12} + \mathbf{2}a_{22}b_{22} + \mathbf{1}a_{23}b_{32} & \mathbf{3}a_{21}b_{13} + \mathbf{2}a_{22}b_{23} + \mathbf{1}a_{23}b_{33} \\ \mathbf{3}a_{31}b_{11} + \mathbf{2}a_{32}b_{21} + \mathbf{1}a_{33}b_{31} & \mathbf{3}a_{31}b_{12} + \mathbf{2}a_{32}b_{22} + \mathbf{1}a_{33}b_{32} & \mathbf{3}a_{31}b_{13} + \mathbf{2}a_{32}b_{23} + \mathbf{1}a_{33}b_{33} \end{pmatrix}$$

By visual inspection, we observe the elements from the first row of A and first column of B are scaled by 3, which is the total number of ranks we test for. The second row/column is scaled by 2

and finally, the third row/column is scaled by 1. Since we need to make sure we learn hierarchical features for both A and B, we can use the square root of coefficients $3, 2, 1$ for C_A and C_B:

$$C_A = \begin{pmatrix} \sqrt{3} & \sqrt{2} & 1 \\ \sqrt{3} & \sqrt{2} & 1 \\ \sqrt{3} & \sqrt{2} & 1 \end{pmatrix} \quad C_B = \begin{pmatrix} \sqrt{3} & \sqrt{3} & \sqrt{3} \\ \sqrt{2} & \sqrt{2} & \sqrt{2} \\ 1 & 1 & 1 \end{pmatrix}$$

Moreover, instead of using matrices C_A and C_B, we can obtain the matrix $\sum_{r\in\{1,2,3\}} A_r B_r$ by multiplying column i of A by P_i, thus yielding the final MATRYOSHKALORA forward pass during training as in Equation 9. Specifically, the product $A * P$ yields the following matrix:

$$A * P = \begin{pmatrix} \mathbf{3}a_{11} & \mathbf{2}a_{12} & \mathbf{1}a_{13} \\ \mathbf{3}a_{21} & \mathbf{2}a_{22} & \mathbf{1}a_{23} \\ \mathbf{3}a_{31} & \mathbf{2}a_{32} & \mathbf{1}a_{33} \end{pmatrix}$$

B THEORY

B.1 THEOREM & PROOF FOR THE MATRYOSHKALORA SURROGATE OBJECTIVE

To develop any further theory, we impose the following smoothness assumption for the loss that is common in the optimization literature.

Assumption B.1 (Smoothness)**.** The loss ℓ is lower bounded, differentiable and L-smooth with respect to the Frobenius norm: there exists a constant $L > 0$ such that

$$\|\nabla\ell(W) - \nabla\ell(W')\|_{\mathrm{F}} \le L\|W - W'\|_{\mathrm{F}}$$

for all $W, W' \in \mathbb{R}^{m\times n}$.

The following notion of rank dispersion will be used throughout the main theoretical claims.

Definition B.2 (Rank dispersion)**.** Define rank dispersion of adapters A and B over rank support $\mathcal{S}$ as

$$\delta_{\mathcal{S}}(A, B) := \sum_{r\in\mathcal{S}} \lambda_r \|\Delta_r - \bar{\Delta}\|_{\mathrm{F}}^2 = \sum_{r\in\mathcal{S}} \lambda_r \left\| \Delta_r - \sum_{t\in\mathcal{S}} \lambda_t \Delta_t \right\|_{\mathrm{F}}^2. \tag{14}$$

Next we present our first main theoretical claim regarding the multi-rank and surrogate objectives.

Theorem B.3 (Surrogate gap)**.** *Under Assumption B.1, for every pair of LoRA factors A and B,*

$$|L_{\mathrm{multi}}(A, B) - L_{\mathrm{surr}}(A, B)| \le \frac{L}{2}\delta_{\mathcal{S}}(A, B). \tag{15}$$

Proof. Since $\nabla f(\Delta) = \nabla\ell(W_0 + \Delta)$, Assumption B.1 implies that f is also L-smooth. Hence, for any perturbations Δ and Δ', the first-order expansion satisfies

$$f(\Delta) = f(\Delta') + \langle \nabla f(\Delta'), \Delta - \Delta' \rangle + \mathcal{R}(\Delta, \Delta'), \quad \text{where} \quad |\mathcal{R}(\Delta, \Delta')| \le \frac{L}{2}\|\Delta - \Delta'\|_{\mathrm{F}}^2.$$

Expanding each rank-specific loss around $\bar{\Delta}$ gives

$$f(\Delta_r) = f(\bar{\Delta}) + \left\langle \nabla f(\bar{\Delta}), \Delta_r - \bar{\Delta} \right\rangle + \mathcal{R}_r, \quad \text{with} \quad |\mathcal{R}_r| \le \frac{L}{2}\|\Delta_r - \bar{\Delta}\|_{\mathrm{F}}^2.$$

After multiplying by λ_r and summing over $r \in \mathcal{S}$, the first-order term vanishes because $\sum_r \lambda_r(\Delta_r - \bar{\Delta}) = 0$. Therefore,

$$L_{\mathrm{multi}}(A, B) - L_{\mathrm{surr}}(A, B) = \sum_{r\in\mathcal{S}} \lambda_r \mathcal{R}_r,$$

and the remainder bound completes the proof. □

Theorem B.3 is a pointwise statement: it holds for arbitrary A and B, regardless of how they were obtained. In particular, it does not require the perturbations Δ_r to be small relative to the pretrained model. Instead, the approximation error is controlled by their dispersion around the perturbation used during surrogate training.

The rank dispersion is a weighted variance and admits the equivalent representations

$$\delta_{\mathcal{S}}(A,B) = \sum_{r\in\mathcal{S}} \lambda_r \|\Delta_r\|_{\mathrm{F}}^2 - \|\bar{\Delta}\|_{\mathrm{F}}^2 = \frac{1}{2}\sum_{r,t\in\mathcal{S}} \lambda_r\lambda_t \|\Delta_r - \Delta_t\|_{\mathrm{F}}^2.$$

B.2 Average optimality gap

The surrogate-gap theorem compares the objective used during training with the desired weighted multi-rank objective. To characterize performance across the supported ranks, we compare each deployed rank-r prefix with an independently optimized rank-r perturbation. Define

$$f_r^* := \inf_{\mathrm{rank}(\Delta)\le r} f(\Delta), \tag{16}$$

which is the best loss achievable by a perturbation of rank at most r.

Let $(\widehat{A},\widehat{B})$ denote the LoRA factors returned by MatryoshkaLora, and define

$$\widehat{\Delta}_r := s_r \widehat{A} P_r \widehat{B}, \qquad \widehat{\bar{\Delta}} := \widehat{A}P\widehat{B}.$$

Since P is invertible by construction, the parameterization APB can represent every perturbation of rank at most R and therefore

$$\inf_{A,B} f(APB) = f_R^*. \tag{17}$$

Theorem B.4 (Average optimality gap)**.** *Let Assumption B.1 holds and suppose the surrogate is optimized to accuracy $\varepsilon \ge 0$, meaning that $f(\widehat{\bar{\Delta}}) \le f_R^* + \varepsilon$. Then the weighted average optimality gap satisfies*

$$\sum_{r\in\mathcal{S}} \lambda_r \left[f(\widehat{\Delta}_r) - f_r^* \right] \le \varepsilon + \frac{L}{2}\delta_{\mathcal{S}}(\widehat{A},\widehat{B}). \tag{18}$$

Proof. Applying Theorem B.3 to $(\widehat{A},\widehat{B})$ gives

$$\begin{aligned} L_{\mathrm{multi}}(\widehat{A},\widehat{B}) &\le L_{\mathrm{surr}}(\widehat{A},\widehat{B}) + \frac{L}{2}\delta_{\mathcal{S}}(\widehat{A},\widehat{B}) \\ &= f(\widehat{\bar{\Delta}}) + \frac{L}{2}\delta_{\mathcal{S}}(\widehat{A},\widehat{B}) \le f_R^* + \varepsilon + \frac{L}{2}\delta_{\mathcal{S}}(\widehat{A},\widehat{B}). \end{aligned}$$

Because the feasible rank classes are nested, $f_R^* \le f_r^*$ for every $r \in \mathcal{S}$. Hence

$$\begin{aligned} \sum_{r\in\mathcal{S}} \lambda_r \left[f(\widehat{\Delta}_r) - f_r^* \right] &= L_{\mathrm{multi}}(\widehat{A},\widehat{B}) - \sum_{r\in\mathcal{S}} \lambda_r f_r^* \\ &\le \varepsilon + \frac{L}{2}\delta_{\mathcal{S}}(\widehat{A},\widehat{B}) + f_R^* - \sum_{r\in\mathcal{S}} \lambda_r f_r^* \le \varepsilon + \frac{L}{2}\delta_{\mathcal{S}}(\widehat{A},\widehat{B}), \end{aligned}$$

which proves equation 18. Furthermore, $\mathrm{rank}(\widehat{\Delta}_r) \le r$, so every term $f(\widehat{\Delta}_r) - f_r^*$ is nonnegative. □

The claim separates the two quantities that control the average optimality gap across the supported ranks: the surrogate optimization error ε and the learned rank dispersion $\delta_{\mathcal{S}}(\widehat{A},\widehat{B})$.

B.3 Structural control of rank dispersion

The preceding results depend on the learned rank dispersion $\delta_{\mathcal{S}}(A,B)$. We now separate this quantity into an adapter-dependent factor and an adapter-independent factor determined entirely by the rank support, rank weights, and scaling rule. Since $\Delta_r = s_r A P_r B$ and $\bar{\Delta} = APB$, we have

$$\Delta_r - \bar{\Delta} = A(s_r P_r - P)B. \tag{19}$$

Define the adapter-independent rank-support factor

$$\delta(\mathcal{S}) := \sum_{r\in\mathcal{S}} \lambda_r \|s_r P_r - P\|_{\mathrm{F}}^2, \qquad P = \sum_{r\in\mathcal{S}} \lambda_r s_r P_r. \tag{20}$$

Unlike $\delta_{\mathcal{S}}(A, B)$, the quantity $\delta(\mathcal{S})$ can be computed before training: it depends on $\mathcal{S}$, $\{\lambda_r > 0\}_{r\in\mathcal{S}}$, and $\{s_r > 0\}_{r\in\mathcal{S}}$, but not on the learned factors A and B.

Proposition B.5 (Adapter–support decomposition)**.** *For any LoRA factors A and B,*

$$\delta_{\mathcal{S}}(A, B) \le \|A\|_2^2 \|B\|_2^2 \, \delta(\mathcal{S}). \tag{21}$$

Moreover, if a_j is the j-th column of A and $b_j^\top$ is the j-th row of B, then

$$\delta_{\mathcal{S}}(A, B) \le \left(\sum_{j=1}^{R} \|a_j\|_2^2 \|b_j\|_2^2 \right) \delta(\mathcal{S}). \tag{22}$$

Consequently, the two bounds can be combined as

$$\delta_{\mathcal{S}}(A, B) \le \delta(\mathcal{S}) \min \left\{ \|A\|_2^2 \|B\|_2^2, \sum_{j=1}^{R} \|a_j\|_2^2 \|b_j\|_2^2 \right\}. \tag{23}$$

Here $\|\cdot\|_2$ denotes the spectral norm for matrices and the Euclidean norm for vectors.

Proof. Using equation 19 and the standard norm inequality $\|AXB\|_{\mathrm{F}} \le \|A\|_2 \|X\|_{\mathrm{F}} \|B\|_2$ give

$$\delta_{\mathcal{S}}(A, B) = \sum_{r\in\mathcal{S}} \lambda_r \|A(s_r P_r - P)B\|_{\mathrm{F}}^2 \le \|A\|_2^2 \|B\|_2^2 \sum_{r\in\mathcal{S}} \lambda_r \|s_r P_r - P\|_{\mathrm{F}}^2,$$

which proves equation 21. For the second bound, let p_j denote the j-th diagonal entry of P. Then

$$p_j = \sum_{r\in\mathcal{S},\, r\ge j} \lambda_r s_r$$

and

$$A(s_r P_r - P)B = \sum_{j=1}^{R} \left(s_r \mathbf{1}_{\{j\le r\}} - p_j \right) a_j b_j^\top.$$

Cauchy–Schwarz in the space of matrices equipped with the Frobenius norm yields

$$\begin{aligned} \|A(s_r P_r - P)B\|_{\mathrm{F}}^2 &\le \left[\sum_{j=1}^{R} \left(s_r \mathbf{1}_{\{j\le r\}} - p_j \right)^2 \right] \left[\sum_{j=1}^{R} \|a_j b_j^\top\|_{\mathrm{F}}^2 \right] \\ &= \|s_r P_r - P\|_{\mathrm{F}}^2 \sum_{j=1}^{R} \|a_j\|_2^2 \|b_j\|_2^2. \end{aligned}$$

Averaging this inequality with weights λ_r proves equation 22. Since both upper bounds hold simultaneously, their minimum gives equation 23. □

The component-dependent factor in equation 22 is invariant under the component-wise LoRA rescaling

$$a_j \mapsto c_j a_j, \qquad b_j \mapsto c_j^{-1} b_j, \qquad c_j \ne 0.$$

It therefore measures the energy of the learned rank-one components without depending on an arbitrary redistribution of scale between the two LoRA factors.

Explicit form of the rank-support factor. Because P is the weighted mean of the diagonal matrices $s_r P_r$, $\delta(\mathcal{S})$ is their weighted variance. The pairwise variance identity gives

$$\delta(\mathcal{S}) = \frac{1}{2} \sum_{r,t\in\mathcal{S}} \lambda_r \lambda_t \|s_r P_r - s_t P_t\|_{\mathrm{F}}^2. \tag{24}$$

For $r \le t$, the pairwise term has the simple form

$$\|s_r P_r - s_t P_t\|_{\mathrm{F}}^2 = r(s_r - s_t)^2 + (t - r)s_t^2. \tag{25}$$

These identities make the dependence on the rank support and scaling rule fully explicit.

Table 5: Asymptotic behavior of the rank-support factor $\delta(\mathcal{S})$ under uniform rank weights. The limits and Θ statements take $K \to \infty$ with r_1 and α fixed.

Rank support	Fixed $s_r = \alpha$	Standard $s_r = \alpha/r$	Rank-stabilized $s_r = \alpha/\sqrt{r}$
Consecutive $K = 1 + R - r_1$	$\Theta(\alpha^2 K)$ grows linearly	$\Theta(\alpha^2 \log K/K)$ vanishes	$\to \alpha^2/3$ remains bounded
Dyadic $K = 1 + \log_2(R/r_1)$	$\Theta(\alpha^2 R/K)$ grows with R	$\Theta(\alpha^2/(r_1 K))$ vanishes	$\to \alpha^2$ remains bounded

Uniform rank weights. For all special cases below, we fix the uniform multi-rank objective. Thus,

$$\mathcal{S} = \{r_1, \dots, r_K\}, \qquad r_1 < \cdots < r_K, \qquad \lambda_{r_i} = \frac{1}{K} \quad \text{for every } i \in \{1, \dots, K\}.$$

Using symmetry in equation 24, we obtain the common starting point

$$\delta(\mathcal{S}) = \frac{1}{K^2} \sum_{1 \le i < j \le K} \|s_{r_i} P_{r_i} - s_{r_j} P_{r_j}\|_{\mathrm{F}}^2. \tag{26}$$

We consider three scaling rules and two rank-support geometries, giving six special cases in total.

Consecutive ranks. First suppose

$$r_i = r_1 + i - 1, \qquad i = 1, \dots, K, \qquad R = r_1 + K - 1.$$

Fixed scaling: $s_r = \alpha$. Because $\|\alpha P_{r_i} - \alpha P_{r_j}\|_{\mathrm{F}}^2 = \alpha^2(j - i)$ for $i < j$,

$$\delta(\mathcal{S}) = \alpha^2 \frac{K^2 - 1}{6K} \le \frac{\alpha^2 K}{6}. \tag{27}$$

Thus $\delta(\mathcal{S}) = \Theta(\alpha^2 K)$, or equivalently $\Theta(\alpha^2(R - r_1 + 1))$.

Standard LoRA scaling: $s_r = \alpha/r$. Let $H_n = \sum_{j=1}^{n} j^{-1}$ denote the n-th harmonic number, with $H_0 = 0$. Using

$$\left\| \frac{\alpha}{r_i} P_{r_i} - \frac{\alpha}{r_j} P_{r_j} \right\|_{\mathrm{F}}^2 = \alpha^2 \left(\frac{1}{r_i} - \frac{1}{r_j} \right), \qquad i < j,$$

the exact support factor is

$$\delta(\mathcal{S}) = \frac{\alpha^2}{K^2} \left[(R + r_1)(H_R - H_{r_1 - 1}) - 2K \right]. \tag{28}$$

Since $H_R - H_{r_1 - 1} \le r_1^{-1} + \log(R/r_1)$, a simple upper bound is

$$\delta(\mathcal{S}) \le \frac{\alpha^2 (R + r_1)}{K^2} \left[\frac{1}{r_1} + \log\left(\frac{R}{r_1} \right) \right]. \tag{29}$$

For fixed r_1, this gives $\delta(\mathcal{S}) = \Theta(\alpha^2 \log K/K)$ as $K \to \infty$.

Rank-stabilized LoRA scaling: $s_r = \alpha/\sqrt{r}$. In this case,

$$\delta(\mathcal{S}) = \frac{2\alpha^2}{K^2} \sum_{1 \le i < j \le K} \left[1 - \sqrt{\frac{r_1 + i - 1}{r_1 + j - 1}} \right] \le \alpha^2 \left(1 - \sqrt{\frac{r_1}{R}} \right). \tag{30}$$

The support factor is therefore $O(\alpha^2)$; more precisely, for fixed r_1,

$$\delta(\mathcal{S}) \longrightarrow \frac{\alpha^2}{3} \qquad \text{as } K \to \infty.$$

Dyadic ranks. Next suppose r_1 is a power of two and each successive supported rank is twice the preceding one:

$$r_i = 2^{i-1} r_1, \qquad i = 1, \dots, K, \qquad R = 2^{K-1} r_1.$$

Fixed scaling: $s_r = \alpha$. Summing the pairwise rank differences gives

$$\delta(\mathcal{S}) = \frac{\alpha^2}{K^2} \left[2(K-3)R + (K+3)r_1 \right] \leq \frac{2\alpha^2 R}{K}. \tag{31}$$

Consequently,

$$\delta(\mathcal{S}) = \Theta\left(\frac{\alpha^2 R}{K} \right) = \Theta\left(\frac{\alpha^2 R}{1 + \log_2(R/r_1)} \right)$$

for fixed r_1 and increasing K.

Standard LoRA scaling: $s_r = \alpha/r$. The reciprocal ranks form a decreasing geometric sequence, and the exact support factor is

$$\delta(\mathcal{S}) = \frac{\alpha^2}{r_1 K^2} \left[2K - 6 + (K+3)\frac{r_1}{R} \right] \leq \frac{2\alpha^2}{r_1 K}. \tag{32}$$

Thus

$$\delta(\mathcal{S}) = \Theta\left(\frac{\alpha^2}{r_1 K} \right) = \Theta\left(\frac{\alpha^2}{r_1 [1 + \log_2(R/r_1)]} \right).$$

Rank-stabilized LoRA scaling: $s_r = \alpha/\sqrt{r}$. Here the pairwise terms depend only on the index separation $d = j - i$, giving

$$\delta(\mathcal{S}) = \frac{2\alpha^2}{K^2} \sum_{d=1}^{K-1} (K-d) \left(1 - 2^{-d/2} \right) \leq \alpha^2 \frac{K-1}{K} < \alpha^2. \tag{33}$$

In particular,

$$\delta(\mathcal{S}) \longrightarrow \alpha^2 \qquad \text{as } K \to \infty.$$

For the two support families considered above, standard LoRA scaling $s_r = \alpha/r$ has the most favorable asymptotic rank-support factor: $\delta(\mathcal{S})$ vanishes for both consecutive and dyadic ranks. Fixed scaling produces a growing support factor, whereas rank-stabilized scaling keeps it bounded but generally nonvanishing. These observations concern $\delta(\mathcal{S})$ rather than the full rank dispersion $\delta_{\mathcal{S}}(A, B)$. Because adapter-dependent factor also depends on the support, scaling, and training dynamics, the support-factor asymptotics alone do not establish that one scaling rule yields a smaller surrogate gap after training.