LoRA's ARK: Towards an Iso-Budget Rank Scaling Law with Adaptive-Rank Kernels

Anonymous authors
Paper under double-blind review

Abstract

Low-Rank Adaptation and its variants dominate parameter-efficient fine-tuning, yet practitioners still have to guess one hyperparameter: the rank r. A rank too low structurally limits the update; a rank too high buys headroom only by paying proportionally more parameters and memory. Behind both horns lies one coupling: the same r sets how expressive the update can be and how many parameters it takes, so no setting of r improves one without hurting the other. Existing solutions either expand rank through fixed parameterizations that lack task-specific adaptivity, or achieve adaptivity only downward, by pruning a rank budget that must be provisioned up front. We argue that the numerical rank should instead be a property that each module learns under a fixed parameter budget. To this end, we propose Adaptive-Rank Kernels (ARK). Instead of selecting kernels by traditional Mercer validity, which only bounds the attainable rank from above, we select them by the rank they measurably induce in an update synthesized from low-rank factors. We then train the kernel's own scale as a learnable rank controller, so that different modules settle at different numerical ranks under a fixed trainable-parameter count: an iso-budget rank scaling law. On language understanding (eight GLUE tasks with BERT) and mathematical reasoning (GSM8k with Qwen2.5), ARK matches or exceeds the PEFT baselines at a matched trainable-parameter budget. It averages 81.47 on GLUE, 5.3 points above LoRA and 3.1 above the strongest baseline at the same budget, statistically indistinguishable from full fine-tuning (81.72) while training about 1% of the parameters, and it leads the matched-cost methods on GSM8k at both model scales. Measuring where the updates actually settle shows the gain comes with the mechanism we asked for: under one fixed budget the attained numerical ranks spread widely across modules and tasks instead of standing at r, so the rank of the update and the parameter count are decoupled in practice. The code will be publicly available.

1 Introduction

Large-scale pretrained Transformers (Vaswani et al., 2017) have become the backbone of modern AI, ranging from encoder-only language models such as BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) to decoder-only large language models (LLMs) such as GPT-3 (Brown et al., 2020), LLaMA (Touvron et al., 2023) and Qwen2.5 (Yang et al., 2025). However, adapting these models via full fine-tuning (FFT) imposes prohibitive computational and storage costs (Wang & Chen, 2023) and is prone to catastrophic forgetting (McCloskey & Cohen, 1989). Parameter-Efficient Fine-Tuning (PEFT) (Houlsby et al., 2019) instead freezes the pre-trained weights and trains only a small set of parameters, a design later supported by the low *intrinsic dimensionality* of fine-tuning (Aghajanyan et al., 2021): a subspace orders of magnitude smaller than the full parameter space recovers most of the fine-tuning performance. Low-Rank Adaptation (LoRA) (Hu et al., 2022) goes one step further, hypothesizing that the weight update itself has a low intrinsic rank. Its simplicity and the fact that the trained factors merge into the frozen weights have made it the de facto standard.

A key hyperparameter of LoRA is the latent dimension r, which fixes the trainable-parameter count $(d_{in}+d_{out})r$ while capping the update through $\mathrm{rank}(\Delta W) \leq r$. We call the resulting tension the **Rank-Dilemma**: a small r is efficient but structurally limits the expressiveness of the update on complex tasks (Huang et al., 2025; Albert et al., 2025), whereas a larger r raises the cap only by paying proportionally more parameters and memory, forfeiting the efficiency that PEFT exists for.

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

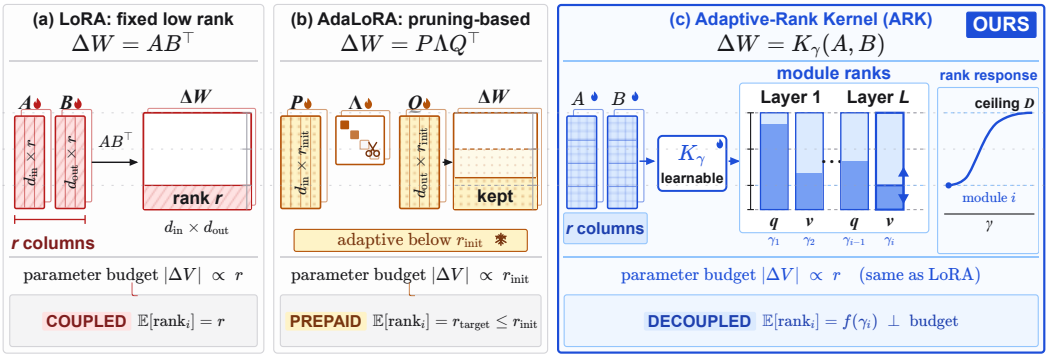


Figure 1: **Three paradigms for the rank of the update ΔW under a parameter budget.** Here rank_i is the rank of module i's update and $|\Delta V|$, labelled *budget*, counts the parameters stored and optimized; flames mark learnable parameters, snowflakes fixed hyperparameters, and stacked outlines the per-module copies. **(a)** Generic dense factors give rank r almost surely, so budget and rank move together. **(b)** Pruning is adaptive, but only downward: rank is cut below the pre-allocated r_{init}, whose budget is paid in full. **(c)** The same r-sized factors feed K_γ, whose per-module scale γ_i sets the rank at LoRA's budget. Shading is read against the shared rank scale, whose levels sit at equal heights across panels: dark is the attained rank, light the reachable head-room, white what the parameterization cannot reach. The inset, the Rank Scaling Law of Sec. 4.2, shares that axis, and its point is module i's reading. Rank slots are schematic, not literal sparsity in the factors.

Recent efforts to address such limitation follow two directions: The first raises the rank attainable under a fixed budget by designing fixed high-rank parameterizations (Hyeon-Woo et al., 2022; Yeh et al., 2024; Jiang et al., 2024). The second makes the rank adaptive, by pruning or by selecting after training, always from a rank budget that must be provisioned in memory before training begins (Zhang et al., 2023b; Valipour et al., 2023; Zhang et al., 2024b), at odds with the premise that motivates PEFT, namely that the larger budget is out of reach. Both directions leave the same knot untied: fixed parameterizations reach only the numerical rank their parameter count dictates, while adaptive schemes move the rank only downward, from a budget paid for in advance. We therefore ask:

Can each module learn the numerical rank it needs, up or down, instead of buying it with more parameters or reclaiming it from a budget provisioned up front?

We call the property this question asks for the **Iso-Budget Rank Scaling Law** (Def. 1): the scaling behaviour in which the numerical rank attainable by ΔW can be dialed up or down while the trainable-parameter count stays fixed. To this end, we begin to think about where this rank cap comes from: LoRA's two factors A and B carry r columns each,[1] so linear algebra leaves the update they synthesize with at most r linearly independent rows. Sending the rows of A and B through a feature map into a much higher-dimensional space would remove the cap at its source, as the two factors would then meet in a far wider inner dimension; but doing so explicitly means storing and training the mapped factors, whose size grows with that dimension, the very cost PEFT exists to avoid. This is exactly what the **"kernel trick"** achieves: a kernel function applied on the row vectors of A and B returns the inner products the mapped factors would give, without the map ever being built. And should that function carry a parameter setting that mapped dimension, it becomes a dial on the rank, one each module could turn up or down at a fixed parameter count: rank tuned rather than bought.

Realizing this idea, however, raises two questions classical kernel theory does not answer. 1. Which function to use? Mercer validity certifies that an implicit space exists, but its dimension only bounds the rank from above: it says neither how much rank the synthesized update actually gains nor whether that gain survives gradient descent (Sec. 3 shows the one prior kernelized LoRA (Shen et al., 2024) caught in exactly this gap, its best-performing kernel not Mercer-valid at all). 2. And can one of its parameters really serve as the dial? A rank is an integer-valued measurement of a matrix, not a differentiable function of the parameters, so nothing guarantees that a scale parameter moves it smoothly, monotonically, or at all. We therefore propose the **Adaptive-Rank Kernel** (ARK) method, and our contributions are three-fold:

[1]To align with the kernel notation of Sec. 4, our $B \in \mathbb{R}^{d_{out} \times r}$ is the transpose of the second factor in the original LoRA formulation, so that $A \in \mathbb{R}^{d_{in} \times r}$ and B both have r columns.

- We identify and formalize the Rank-Dilemma in LoRA, and state the property a remedy should have: the attainable numerical rank should move under a fixed trainable-parameter count, which we formalize as an Iso-Budget Rank Scaling Law (Def. 1).

- We answer both questions by studying the *rank scaling behaviour* of candidate kernels: we measure how the expected numerical rank of an update synthesized from low-rank factors responds to a kernel's scale, and retain only kernels whose response is stable across initializations, monotone, and predictable (Sec. 4.2). The scale of such a kernel is then trained alongside A and B as a learnable rank controller (Sec. 4).

- We validate ARK on GSM8k (Qwen2.5) and GLUE (BERT): at $\sim$1% of the parameters it lands within a quarter of a point of FFT on GLUE (81.47 against 81.72) and matches FFT on GSM8k (Sec. 5.2), merging into W at no inference cost. The ranks reached (Sec. 5.3, Fig. 1) vary across layers and projections at one fixed r: the Iso-Budget Rank Scaling Law we set out to obtain.

2 Related Work

Rank in Parameter-Efficient Fine-Tuning. PEFT adapts a frozen backbone by training a small set of parameters, conventionally grouped into addition-based, specification-based and reparameterization-based families (Ding et al., 2023). The first two families never synthesize a weight update matrix, so its rank is not a design variable for them: this covers adapters (Houlsby et al., 2019), prompt and prefix tuning (Lester et al., 2021; Li & Liang, 2021), activation scaling (Lian et al., 2022; Liu et al., 2022) and bias-only tuning (Ben Zaken et al., 2022). Reparameterization-based methods, to which LoRA (Hu et al., 2022) and this work belong, do synthesize one. A first line among them keeps LoRA's low-rank product and improves how it is initialized or optimized (Zhang et al., 2023a; Meng et al., 2024; Hayou et al., 2024; Wang et al., 2024; Kalajdzievski, 2023), which leaves the rank of ΔW where LoRA puts it and addresses a question different from the one studied here. The methods that do act on the rank take two routes. One raises it by construction, through matrix constructions that multiply rank: Hadamard or Kronecker products of low-rank factors (Hyeon-Woo et al., 2022; Edalati et al., 2025; Yeh et al., 2024), a Hadamard product with the frozen weight (Huang et al., 2025), or combinations of fixed random bases (Albert et al., 2025). The rank so obtained is fixed a priori by the parameterization rather than learned, even though different layers are known to call for different ranks (Zhang et al., 2024b). The other route adjusts the rank during or after training, by pruning singular values under a shrinking budget (Zhang et al., 2023b), by training a block that is truncated to a randomly sampled rank (Valipour et al., 2023), or by meta-learning a per-component selection variable that is thresholded into a layer-specific rank (Zhang et al., 2024b). In each case the rank is set by a quantity supplied from outside the training objective, be it a rank budget, a maximum rank or a validation-time threshold, which has to be provisioned before training and can only be reduced from there. ARK instead moves the rank upward under an unchanged parameter count: nothing is pre-allocated, the attained rank is a by-product of optimizing the task loss alone, and no post-hoc selection step is required.

Kernels in Deep Learning. Kernels serve in deep learning both as analysis tools (Jacot et al., 2018; Novak et al., 2019) and as architectural components (Wilson et al., 2016; Luo et al., 2022; Chowdhury et al., 2022; Wu et al., 2023). The idea sketched in Sec. 1 comes instead from the classical kernel trick behind support vector machines, where $K(x, y) = \langle \phi(x), \phi(y) \rangle$ yields inner products in a high-dimensional feature space without the map ϕ ever being built (Bishop, 2006). Read on the rows of the low-rank factors, this suggests a way past the rank cap that costs no extra parameters, and it is the route KLoRA (Shen et al., 2024) takes, the kernelized-LoRA component of SNELL and to our knowledge the first kernelized LoRA: it invokes Mercer's theorem for the existence of ϕ, then picks its kernel by downstream accuracy, never measures the rank that results, and introduces its trainable kernel coefficients as rank-neutral. Existence, however, is not the guarantee this purpose needs, since the dimension of that space only bounds the rank from above (Sec. 3): we therefore select kernels by the expected numerical rank they are measured to induce, and train the kernel scale as a rank controller, which frees the choice from Mercer validity altogether. The attained rank is thus fixed by the kernel choice there, but optimized here.

3 PRELIMINARIES AND MOTIVATION

Decoupling the Parameter Count from the Matrix Rank. The success of PEFT relies on the hypothesis that adaptation occurs within a low *intrinsic dimension* (Aghajanyan et al., 2021). LoRA (Hu et al., 2022) instantiates this by freezing the pre-trained weights W and modeling the update as:

$$W' = W + \Delta W = W + \alpha\gamma AB^T \tag{1}$$

where $A \in \mathbb{R}^{d_{in}\times r}$ and $B \in \mathbb{R}^{d_{out}\times r}$ are trainable, α is a constant scaling factor, γ is a scalar that standard LoRA fixes as a function[2] of r, and $r \ll \min(d_{in}, d_{out})$ is the latent dimension, which sets the trainable-parameter count $(d_{in}+d_{out})r$ and, since both factors carry exactly r columns, also the inner dimension in which the two of them meet.

While that count must stay low for efficiency, LoRA ties a second quantity to the same r: the *numerical rank* of the update cannot exceed that shared inner dimension, and for continuously distributed factors it is in fact equal to r almost surely (App. C.1.2). Gradient descent therefore cannot settle on a rank below r of its own accord: the rank is pinned by the parameterization, whatever rank the task turns out to need. The cost of that coupling is quantitative. Assuming an optimal update ΔW^* with descending singular values σ_i, the Eckart-Young-Mirsky theorem (Eckart & Young, 1936) fixes the error of every rank-r approximation at $\min_{A,B}\|\Delta W^* - AB^T\|_F = \sqrt{\sum_{i=r+1}^{D}\sigma_i^2}$, so an adaptation whose spectrum carries a heavy tail of significant singular values has that tail discarded, and the residual acts as a performance barrier insofar as the loss is sensitive to it. In concrete terms this is the Rank-Dilemma of Sec. 1: the only way to lower the barrier is to raise r, which raises the parameter count in step. We formalize both regimes below.

Definition 1 (Rank Scaling Paradigms)**.** *Consider any PEFT method that synthesizes an update ΔW from a set of tunable parameters ΔV, and let $|\Delta V|$ be its parameter budget, i.e. the number of scalars that have to be optimized and stored for each task (the "budget" referred to in the Iso-Budget Rank Scaling Law); for LoRA, $|\Delta V| = (d_{in}+d_{out})r$. Expectations below are taken over the random initialization of ΔV. Under the* ***Budget-Coupled Rank Scaling Law****, the paradigm of existing methods, the expected numerical rank is fixed before training by the chosen parameterization and its budget, $\mathbb{E}[\mathrm{rank}(\Delta W)] = f(|\Delta V|)$ with f determined by that choice, so reaching a different rank means provisioning a different budget; LoRA is the special case in which f is linear; in AdaLoRA (Zhang et al., 2023b) it is set by the pre-allocated r_{init}, which pruning can only lower (Fig. 1(b)). Under the* ***Iso-Budget Rank Scaling Law****, the one we propose, the parameterization also carries a set Θ of $O(1)$ additional parameters, independent of r, whose values set the expected rank, $\mathbb{E}[\mathrm{rank}(\Delta W)] \approx g(\Theta)$, so that the rank moves as Θ moves while $|\Delta V|$ stays constant.*

We call the range of numerical ranks attainable by ΔW under a fixed parameter budget a method's **rank expressiveness**. Materializing the law thus requires an operator that raises the numerical rank of the update beyond r without enlarging A or B, and kernel functions are natural candidates: they implicitly map the low-rank factors into a higher-dimensional feature space at no extra cost.

Rethinking Kernel Validity. One relevant attempt in this direction is KLoRA (Shen et al., 2024), which kernelizes the factors to obtain high-rank updates and, following classical SVM theory, treats Mercer's theorem as the licence for the implicit mapping. Its own results sit awkwardly with that licence: strictly valid Mercer kernels such as RBF underperform, while the best-performing "piecewise linear" formulation it proposes is not a valid Mercer kernel at all (proof of its non-PSD nature is in App. C.8). The discrepancy is attributed to optimization difficulty, yet it points at something more basic: what makes a kernel good for an SVM, that an implicit mapping exists, is not what makes it good here, that the rank actually grows. Mercer validity is the wrong gate on two counts. It does not require the mapping $\phi(\cdot)$ to increase the spatial dimensionality, and indeed the linear kernel (AB^T) and the Mercer-Tanh kernel (Carrington et al., 2014) are both PSD while offering zero rank expansion, staying exactly on the Budget-Coupled Rank Scaling Law. Nor does an infinite-dimensional feature space, as with RBF, keep the mapped rows meaningfully independent under gradient descent, where saturation or trivial alignment collapses the numerical rank that the theory promises.

[2] Set to $\frac{1}{r}$ in (Yang & Hu, 2021) and to $\frac{1}{\sqrt{r}}$ in (Kalajdzievski, 2023). We keep the symbol γ because it plays the same role as in the SVM sigmoid kernel $\tanh(\gamma\, x^\top z + c)$ (Lin & Lin, 2003), where it likewise scales an inner product and is set by convention rather than learned; Sec. 4 learns it instead.

4 ADAPTIVE-RANK KERNELS

To resolve the theoretical gap identified in Sec. 3 and materialize the Iso-Budget Rank Scaling Law, we discard the restrictive Mercer validity. Instead, we focus on a kernel's ability to manipulate rank.

4.1 RANK SCALING DENSITY

An SVM kernel evaluates a symmetric similarity $k(u, v)$ over a single set of vectors to form a Gram matrix. We instead extend a continuous bivariate $k : \mathbb{R}^r \times \mathbb{R}^r \to \mathbb{R}$ across the two factors, giving $K(A, B) \in \mathbb{R}^{d_{in} \times d_{out}}$ with $[K(A, B)]_{ij} = k(A_{i*}, B_{j*})$, the matrix kernel of k.

Mercer theory classifies kernel functions k into valid and invalid, according to the existence of implicit mapping or equivalently PSD. Shifting away from Mercer validity, we classify them instead by the rank they induce.

Definition 2 (Rank Scaling Density ρ and Rank Stability)**.** *Let $A \in \mathbb{R}^{d_{in} \times r}$ and $B \in \mathbb{R}^{d_{out} \times r}$ have row vectors drawn i.i.d. from a Gaussian distribution $\mathcal{N}(0, \sigma^2 I)$, where $r \ll \min(d_{in}, d_{out})$. The numerical rank of the resulting kernel matrix, $\mathbf{R}_K = \mathrm{rank}(K(A, B))$, is a bounded random variable, and we define the **Rank Scaling Density** of K as its expected scaling ratio:*

$$\rho(K, \sigma^2) = \frac{\mathbb{E}_{A,B}[\mathbf{R}_K]}{r} \tag{2}$$

*For convenience, we denote the standard $\rho(K, 1^2)$ as $\rho(K)$, and we suppress in this notation the dependence of ρ on r and on the layer dimensions, which we examine empirically in App. C.4. We call K **rank-stable** if $\mathbf{R}_K$ concentrates, i.e. if its spread over random draws of A and B is negligible next to r; the hypothesis tests by which we decide this are given in App. C.6.*

Concentration is what licenses treating the rank of a random matrix as a property of the kernel: the entries of A and B are random, yet for the kernels we retain $\mathbf{R}_K$ barely moves across draws (Tao, 2023), so ρ can be measured once and then read as a function of the kernel and its scale. Not every non-linearity behaves this way, which is why stability enters the definition instead of being assumed. This definition classifies kernels by their ability to manipulate the rank of the synthesized update. While relaxing strict PSD constraints and providing a more general extension, this criterion aligns with classical kernel theory when analyzing PSD kernels: for those, as detailed in App. C.2, $r\rho$ is a lower bound on the effective dimensionality of the implicit Reproducing Kernel Hilbert Space, so measuring ρ probes that space from below without requiring it to exist in the first place. In this metric the LoRA update of Sec. 3 sits at $\rho(K, \sigma^2) = \mathbf{1}_{\{\sigma \neq 0\}}$, while ρ recovers the $\binom{r+p}{p}$ combinatorial expansion of the Polynomial kernel whenever the layer is large enough to accommodate it (App. C.7), so it applies to Mercer and non-Mercer kernels alike.

To make the attainable rank respond to the task, not the budget, we single out a rank-stable sub-class.

Definition 3 (Adaptive-Rank Kernel)**.** *An **Adaptive-Rank Kernel** (ARK) is a family of rank-stable kernels $\mathcal{K} = \{K_\Theta\}$ with $|\Theta| = O(1)$ independent of r, whose Rank Scaling Density $\rho(K_\Theta, \sigma^2)$ is continuous, differentiable and non-constant in Θ on an open domain, so Θ moves the attainable rank.*

Because the criterion asks about rank rather than positive semi-definiteness, the pool of candidates is no longer confined to Mercer kernels: any differentiable element-wise non-linearity applied to AB^T gives a matrix kernel whose ρ we can measure. Two of them carry the analysis below. Tanh is a classical activation that also serves in SVMs as the sigmoid kernel, where it is known not to be PSD[3]; **Swish** (Ramachandran et al., 2018) has, to our knowledge, not been used as a kernel before.

To study these inner-product-based ARK candidates[4], we employ a unified matrix-form representation:

$$\Delta W = K_\Theta(A, B) = \alpha \cdot \mathrm{Act}\big(\gamma \cdot (AB^T) + \delta\big) + \beta \tag{3}$$

where AB^T serves as the base linear similarity, $\mathrm{Act}(\cdot)$ applies the non-linearity (e.g., Tanh, Swish, Polynomial), and $\Theta = \{\alpha, \beta, \gamma, \delta\}$ dictates the structural transformation.

[3] In SVM literature, the "Sigmoid" kernel typically refers to the hyperbolic tangent form ("Tanh Sigmoid"), whereas in deep learning the term "sigmoid" usually denotes the logistic sigmoid activation. To avoid confusion, we abbreviate the "Tanh Sigmoid" kernel as the Tanh kernel in this paper.

[4] The RBF and Chi2 kernel are not inner-product-based; their formulas are in Tab. 6 in the appendix.

4.2 IDENTIFYING EFFECTIVE ARKS

Identifying the Rank Controller: Of the four parameters in Θ, we single out one to study and hold the others fixed. The post-activation scale α only sets the magnitude of the update, which leaves the singular value spectrum, and with it the numerical rank, untouched. The two shifts do reach the rank, each through a mechanism of its own: β adds a rank-one term, raising the attainable rank by one, and the pre-activation shift δ changes the homogeneity of the kernel, which is a route to rank expansion in its own right (Apps. C.7 and C.3.5). We set $\beta = \delta = 0$ so that the scale is the one quantity moving in the analysis below, not because the other two are inert.

This leaves the pre-activation scale γ ($\Theta = \{\gamma\}$). For standard Gaussian factors $A, B \sim \mathcal{N}(0, 1^2)$ and $\gamma > 0$, $\gamma AB^T = (\sqrt{\gamma}A)(\sqrt{\gamma}B)^T$, so scaling by γ is the same as drawing the factors from $\mathcal{N}(0, \gamma)$, and $\rho(K_\gamma, 1) = \rho(K_1, \sigma^2)$ with $\sigma^2 = \gamma$. The latent dimension r enters the same scale through the r-term inner product, which is why standard LoRA ties γ to r. Rescaling A and B or enlarging r moves the variance too, but those routes act on the parameterization itself, and r carries the parameter count with it. Freeing γ then becomes one route that shifts the variance continuously and differentiably while keeping d_{in}, d_{out}, r and $|\Delta V|$ stay fixed.

This equivalence gives a usable diagnostic: *if its ρ responds strongly and predictably to the input standard deviation σ, a function of the form in Eq. (3) is an ARK.* We use the condition in this one direction and keep the analysis to the single scale γ; whether the other members of Θ can serve as controllers as well is examined in Apps. C.7 and C.3.5. Not all functions pass: the linear and polynomial kernels have a rank that is invariant to variance scaling. For variance-responsive non-linearities, γ becomes the **learnable rank controller** we expose: it acts through the kernel's own scale and leaves the update itself untouched, whereas prior work fixes that scale to a constant and we let the task loss set it. By optimizing γ, the model adapts the effective input variance during training, pushing the matrix toward the rank $r \times \rho(K_\gamma)$.

This linkage turns kernel selection into a measurement. By measuring a kernel's expected ρ across different σ, we can map out its Rank Scaling Law (RSL, as charted in Fig. 2). Many non-linearities are variance-responsive, yet their scaling trajectories differ, which raises the next question.

What kind of RSL curves are suitable for adaptive matrix rank learning? To answer this question, we evaluate the empirical RSL landscapes of these variance-responsive candidates (Fig. 2) against three practical criteria that a kernel must meet before its rank can be steered.

1. Stability. For the Rank Scaling Law to be reliably exploited, the rank must concentrate across random draws of A and B rather than follow the seed, *i.e.* effective ARKs should be rank-stable. As visualized in Fig. 2a, the Chi-Square kernel varies with the random seed, so it provides no reliable signal for rank adjustment. The tests we use to separate volatile from stable kernels are in App. C.6.

2. Monotonicity. An effective ARK should map an increase in σ (set by γ) to a higher expected rank monotonically. Under a non-monotone map the sign of the rank response depends on where γ currently sits, so the same step can raise or lower the rank and the direction in which to move γ is not determined by the map alone. The RBF kernel (Fig. 2b) is of this type: as σ grows, its kernel matrix entries are driven toward zero and the rank rises and then collapses.

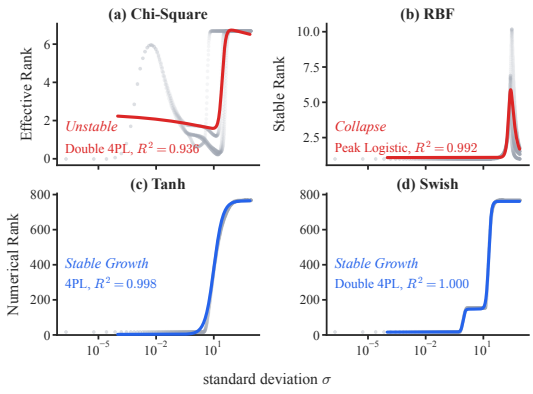


Figure 2: **Rank Scaling Law landscapes against the input standard deviation σ ($d_{in} = d_{out} = 768$, $r = 16$).** Panels (a) Chi-Square, (b) RBF, (c) Tanh, and (d) Swish; the criteria each kernel meets or fails are analyzed in the text. Stability tests at other scales and indicators are in App. C.6.

3. Predictability. The rank is an integer-valued measurement of a matrix, not a differentiable function of the parameters, so we never differentiate it. What has to be well behaved is the map from γ to numerical rank: if that map is erratic, a step on γ can move the rank either way, so it cannot be steered at all. We use this as a criterion for screening candidates, not a claim about training dynamics.

Table 1: Performance on GLUE with BERT-Base-Cased (Matthews for CoLA, Spearman for STS-B, accuracy otherwise). Rows are grouped by Relative Parameter Cost (RPC): the unconstrained FFT reference, methods at our budget (RPC 1.09×10^{-2}, *i.e.* LoRA at r=32), methods at their own recommended budget, and ours. **Bold** and underline mark the best and second-best *at our budget*; *Pos.*, the mean position over the eight tasks (1 = best), and the significance markers cover all rows. AdaLoRA prunes rather than grows, so at our budget it ends at rank 16; § provisions twice the budget to retain r=32. † marks symmetric residual initialization ($W_0 = W - K_\gamma(A, B)$); other LoRA-family rows use $B = 0$. $^*/^{**}$: worse than ARK-Tanh at $p < 0.05/0.001$ (Conover post-hoc, Holm-corrected).

Method (Year)	MRPC	STS-B	RTE	CoLA	MNLI	QNLI	QQP	SST-2	Avg.↑	Pos.↓	RPC
FFT	85.05	88.97	65.70	56.50	84.12	90.50	90.75	92.20	81.72	2.06	1.00e+0
LoRA '21	68.38	87.04	52.71	56.76	75.92	89.69	88.13	90.83	76.18	10.25^{**}	1.09e-2
AdaLoRA '22	69.36	85.71	57.04	42.24	81.27	89.42	87.62	91.06	75.47	10.56^{**}	1.09e-2
DyLoRA '23	79.41	86.85	63.90	43.73	80.25	89.18	86.70	91.06	77.63	9.62^{**}	1.09e-2
KLoRA '24	81.86	88.11	60.29	44.96	80.40	<u>90.79</u>	88.19	**92.09**	78.34	6.75^{*}	1.09e-2
AdaLoRA§ '22	78.43	87.64	57.04	51.80	82.08	90.50	88.38	91.51	78.42	7.38^{*}	2.18e-2
VeRA '23	82.11	87.99	60.65	37.15	76.36	87.00	85.01	88.53	75.60	11.62^{**}	2.41e-4
BOFT '23	78.92	86.24	59.93	46.66	73.22	88.83	86.68	90.83	76.41	11.81^{**}	8.65e-4
LoKr '23	83.09	87.42	62.09	53.14	78.01	88.38	86.32	90.48	78.62	10.25^{**}	3.69e-4
OFT '23	83.33	88.56	62.82	56.25	80.30	89.69	88.02	90.94	79.99	7.06^{*}	4.27e-3
ARK-Swish	**85.54**	88.36	63.90	52.32	81.22	89.68	88.16	90.94	80.02	6.38	1.09e-2
ARK-Swish†	<u>84.07</u>	88.10	<u>66.06</u>	53.90	81.65	89.73	88.44	<u>91.63</u>	80.45	4.50	1.09e-2
ARK-Tanh	83.82	<u>88.58</u>	**66.79**	**58.82**	**82.83**	**90.92**	**89.05**	90.94	**81.47**	**2.94**	1.09e-2
ARK-Tanh†	83.82	**88.63**	63.90	<u>57.02</u>	<u>82.42</u>	90.39	<u>88.94</u>	<u>91.06</u>	<u>80.77</u>	3.81	1.09e-2

For Tanh (Fig. 2c) the map is a saturating S-curve. When $\sigma \to 0$, the Taylor expansion $\tanh(z) \approx z$ leaves the update linear and the rank at its baseline; as σ pushes the inputs into the non-linear zone the rank grows, and it saturates at the structural rank ceiling D. A four-parameter logistic (4PL) curve $y = y_0 + (a - y_0)/(1 + (x/c)^b)$ describes it with $R^2 = 0.9981$, while Swish (Fig. 2d) takes a two-stage form (Double-4PL, $R^2 = 0.9996$). The candidate families and the parsimony rule that selects among them are given in App. C.4. Tanh and Swish meet all three criteria, and we adopt them as our default kernels. Sec. 5.3 confirms this screening: the kernels rejected here fall behind LoRA.

5 Experiments

5.1 Experimental Setup

Datasets and Models. We span two task types and two backbone families. For **language understanding**, we fine-tune BERT-Base-Cased (Devlin et al., 2019) on the eight-task GLUE benchmark (Wang et al., 2019) (dropping WNLI, as in LoRA and its successors), and rerun the same protocol on RoBERTa-Base (Liu et al., 2019) as a backbone-transfer check. For **text-based reasoning**, we fine-tune Qwen2.5-3B and Qwen2.5-7B (Yang et al., 2025) (base checkpoints) on MetaMathQA (Yu et al., 2024) and evaluate on GSM8k (Cobbe et al., 2021), following Mangrulkar et al. (2022). This covers encoder-only and decoder-only architectures across discriminative and generative tasks.

Baselines and Metrics. We measure cost by the *Relative Parameter Cost* $\mathrm{RPC} = |\Delta V|/|W|$, the number of tunable parameters ΔV relative to the parameters W that FFT updates, measured at initialization (App. B.2). We compare against FFT and fourteen recent PEFT methods (Hu et al., 2022; Zhang et al., 2023b; Valipour et al., 2023; Shen et al., 2024; Kopiczko et al., 2024; Edalati et al., 2025; Qiu et al., 2023; Liu et al., 2024b; Albert et al., 2025; Bilican et al., 2025; Jung et al., 2025; Bini et al., 2025; Liu et al., 2024a; Nayak et al., 2026)[5]. Every LoRA-like baseline is run at our own budget; methods whose parameterization we could not budget-match are reported at their recommended configuration with their RPC listed.

Implementation Details. All methods share one training loop (PyTorch (Paszke et al., 2019), AdamW (Loshchilov & Hutter, 2019)) on a single NVIDIA L20 (48GB) GPU, are applied to the same target modules (W_q, W_v), and use fixed batch sizes and training steps per task. Method-specific hyperparameters (learning rate, `lora_alpha`) are set by a deterministic grid search, and

[5]A detailed summary of each method is in Tab. 11.

we report the median over multiple random seeds. Full configurations, dataset statistics, and the generation/scoring protocol are detailed in App. D.

5.2 Main Results

We report the two strongest kernel variants identified by our ablations (Sec. 5.3), ARK-Tanh and ARK-Swish; a rerun of the GLUE protocol on RoBERTa-Base is in App. D.4.

RQ1: Under a matched budget, does adaptive rank beat fixed and pruned rank? Yes, on language understanding. On eight GLUE tasks with BERT-Base-Cased (Tab. 1), ARK-Tanh achieves the highest average score of 81.47, surpassing every PEFT baseline we evaluate (+3.13 over KLoRA, the strongest at our own budget, and +1.48 over OFT, the strongest of the remaining rows) and closely approaching the Full Fine-Tuning (FFT) bound of 81.72, all while utilizing $\sim$1% of the trainable parameters. The gap is not a budgeting artifact: AdaLoRA, which prunes an over-provisioned rank instead of growing one, still reaches only 78.42 even when granted twice the provisioned budget. A Friedman test over the eight tasks rejects equal mean positions, and Conover's post-hoc test with Holm correction places ARK-Tanh first among PEFT rows with no significant difference from FFT.

Table 2: Accuracy on GSM8k test set with Qwen2.5-3B and Qwen2.5-7B. Grouping, †, **bold** and underline follow Tab. 1. **OOM** indicates out-of-memory on a single 48GB GPU.

Method (Year)	Qwen2.5-3B		Qwen2.5-7B	
	Acc (%)	RPC	Acc (%)	RPC
FFT	78.24	1.00e+0	**OOM**	-
LoRA '21	77.41	2.39e-3	82.64	1.33e-3
AdaLoRA '22	77.71	2.39e-3	82.98	1.33e-3
DyLoRA '23	77.94	2.39e-3	83.21	1.33e-3
KLoRA '24	77.03	2.39e-3	81.80	1.33e-3
DoRA '24	77.41	2.39e-3	82.79	1.33e-3
GraLoRA '25	77.56	2.39e-3	82.49	1.33e-3
DeLoRA '25	78.24	2.39e-3	80.74	1.33e-3
RandLoRA '24	76.57	1.58e-3	82.64	1.52e-3
VeRA '23	75.82	3.29e-5	82.18	1.69e-5
BOFT '23	76.72	2.18e-4	82.82	1.20e-4
OFT '23	77.10	3.09e-3	82.11	2.97e-3
OSF '25	78.77	4.22e-2	82.11	5.11e-2
WaveFT '25	77.03	2.52e-4	82.64	1.91e-5
ARK-Swish	77.79	2.39e-3	82.87	1.33e-3
ARK-Swish†	77.56	2.39e-3	82.79	1.33e-3
ARK-Tanh	77.94	2.39e-3	**83.24**	1.33e-3
ARK-Tanh†	**78.39**	2.39e-3	83.09	1.33e-3

Does the result transfer to another encoder? Yes. We rerun the same eight-task GLUE protocol on RoBERTa-Base against the rank-control family. ARK variants again hold the best average and the ordering of the main results carries over; the per-task numbers, the parameter-cost caveats and the significance analysis for this backbone are deferred to App. D.4 (Tab. 12).

RQ2: Does the answer hold for generative reasoning at scale? Yes, though by a narrower margin. Tab. 2 reports GSM8k accuracy on Qwen2.5-3B and Qwen2.5-7B, decoded greedily and scored by exact match on the full test split (protocol in App. D). Among methods at matched parameter cost, ARK-Tanh† is the best on 3B (78.39%), on par with FFT (78.24%) while training $\sim$0.24% of the parameters, and ARK-Tanh is the best on 7B (83.24%), where FFT exceeds our single-48GB-GPU budget. One baseline leads in one setting but not the other: OSF is higher on 3B (78.77%) at roughly $18\times$ the parameter cost yet falls to 82.11% on 7B. Since the leading matched-cost methods are close at 7B (83.24% against 83.21% for DyLoRA), we read the table as ARK being the only method that stays in the top group at both scales rather than as a large per-setting gain. This is consistent with our Rank-Dilemma hypothesis (Sec. 3).

Table 3: Efficiency on Qwen2.5-7B under the same GSM8k setting as Tab. 2. "Added Mechanism" is the extra operation each method puts on top of the frozen backbone. Time and memory are normalized to LoRA (18.4 GiB reserved CUDA memory, 35 min 7 s; median over runs); differences within the low-rank family are at the level of implementation variance (DyLoRA's rank dropout shrinks the effective adapter during training). Every method here, ours included, merges back into W before deployment, so the table concerns training-time cost only.

Method (Year)	Added Mechanism	RPC	Time	Memory	Acc (%)
LoRA '21	Matmul	1.33e-3	1.00×	1.00×	82.64
AdaLoRA '22	SVD form matmul&Pruning	1.33e-3	1.06×	1.02×	82.98
DyLoRA '23	Rank-truncated matmul	1.33e-3	0.94×	1.00×	83.21
BOFT '23	Ortho. decomp.	1.20e-4	**2.99**×	**1.41**×	82.82
OSF '25	Adaptive SVD & Ortho. proj.	5.11e-2	2.07×	1.29×	82.11
ARK-Tanh	Elementwise Act.	1.33e-3	1.01×	1.08×	**83.24**

RQ3: What does the rank expansion cost? On Qwen2.5-7B it costs $1.01\times$ the training time and $1.08\times$ the memory of LoRA (Tab. 3), because ARK adds only an element-wise activation rather than the orthogonal decompositions or adaptive SVDs used by other high-rank designs (BOFT: $2.99\times$ time). At inference, the synthesized update $K_\Theta(A, B)$ is precomputed and merged into W, so the forward-pass cost is identical to standard LoRA.

5.3 Ablations and Rank Allocation

Ablations use a three-task GLUE subset for rapid iteration; results are unified in Tab. 4.

RQ4: Which ingredient does the work, the kernel or the learnable γ? Both, and in that order. Under identical parameter budgets the kernel families form a ladder, validating our Kernel Selection Guidelines (Sec. 4.2). Unstable or non-monotonic kernels (`RBF` and `Chi2`) suffer from rank collapse, trailing behind standard LoRA. While the `Polynomial` kernel theoretically expands rank, its discrete degree causes numerical instability, and it suffers from catastrophic vanishing gradients under asymmetric initialization (App. C.7). Crucially, applying Tanh or Swish even with a fixed pre-activation scaling is already on par with or better than standard LoRA, consistent with such kernels providing a higher Rank Scaling Density. Elevating γ to a learnable rank controller improves performance further, as the Iso-Budget Rank Scaling Law requires.

Extensibility Analysis. Convex combinations of two Mercer kernels produce a new valid kernel(Bishop, 2006). The same closure holds for our kernels, so two of them can be added into a hybrid (See C.2). We evaluate a Hybrid ARK, $K^{\ddagger}_{\gamma}(A,B) = AB^T + K_{\gamma}(A,B)$, which retains the low-rank linear pathway while injecting high-rank non-linear compensation; it attains the best average in Tab. 4.

Initialization Strategy and Rank Allocation. We evaluate two initialization schemes: standard asymmetric LoRA ($B=0$) and the † scheme of Tab. 1. Although our Def. 2 is symmetric, for fair comparison with LoRA, ARK also supports the asymmetric setting: $K(A, B=0) = 0$ gives a strict zero-rank update at step 0, so the pre-trained behaviour is preserved exactly. Where the update then settles is shown in Fig. 3: under one budget it *resides* at a rank set by the kernel rather than by r: a LoRA-equivalent kernel never leaves 32, whereas the same budget spent through an ARK lands anywhere from 42 to 534 depending on kernel, module and task. Both schemes are competitive (Tab. 1), so the choice can follow deployment constraints.

Table 4: **Ablation Study on the GLUE Subset.** We evaluate kernel types and structural extensions under identical parameter budget. Methods marked with '$*$' use a fixed scale parameter; '$\dagger$' is the symmetric residual initialization of Tab. 1; '$\ddagger$' is the hybrid kernel that mixes the row's kernel with the linear (LoRA) one.

Kernel	MRPC	STS-B	RTE	Avg.
RBF	68.38	-3.37	52.71	39.24
Sigmoid	68.38	1.33	52.71	40.81
Chi2	68.38	63.82	57.04	63.08
Polynomial	68.38	69.89	57.04	65.10
Linear (*a.k.a.* LoRA)	68.38	87.04	52.71	69.38
Piecewise Linear (*a.k.a.* KLoRA)	81.86	88.11	60.29	76.75
Tanh	83.82	88.58	66.79	79.73
Tanh*	82.60	88.46	64.62	78.56
Tanh†	83.82	**88.63**	63.90	78.78
Tanh‡	84.56	87.77	**70.04**	80.79
Swish	**85.54**	88.36	63.90	79.27
Swish*	79.90	88.29	65.34	77.84
Swish†	84.07	88.10	66.06	79.41
Swish‡	85.05	87.77	69.68	**80.83**

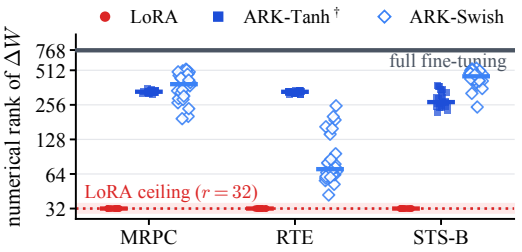


Figure 3: **Which ranks are reachable under one parameter budget?** Numerical rank of ΔW after training for each of the 24 layer–module pairs (`query`/`value` of BERT-Base's 12 layers) at fixed r=32; bars are medians, the vertical axis $\log_2$. A LoRA-equivalent kernel ($\Delta W = AB^T$, no learnable kernel parameter) is pinned at exactly 32 on every module and task, whereas the same budget spent through an ARK lets it reside anywhere between that ceiling and the full fine-tuning reference: ARK-Tanh† keeps a narrow band (219–381) while ARK-Swish spreads wider and is markedly task-dependent (42–251 on RTE against 194–534 on MRPC and STS-B). ARK's adaptivity lets one kernel buy a feasible *range*, not a single point.

6 Conclusion

We addressed the Rank-Dilemma in low-rank adaptation, where one hyperparameter both sets the trainable-parameter count and caps the numerical rank of the update. Rather than selecting kernels by Mercer validity, which only bounds the attainable rank from above, we select them by the rank they measurably induce and train the kernel scale as a rank controller, so modules settle at different numerical ranks under one fixed budget. On the eight GLUE tasks with BERT and on GSM8k with Qwen2.5 at two model scales, ARK matches or exceeds the PEFT baselines at a matched trainable-parameter budget, averaging 81.47 on GLUE against 81.72 for full fine-tuning at about 1% of the parameters. The measured ranks then show the mechanism at work: they spread across layers and projections at one fixed r. The update still merges into W at no inference cost.

AI USE STATEMENT

The scientific content of this work, including the research questions, the adaptive-rank kernel method, the theoretical analysis, the experimental design, and the conclusions, was produced by the authors. Generative AI (large language model) tools were used only as execution and writing assistants in the production pipeline. Specifically, such tools were used for: (i) drafting and iteratively editing parts of the manuscript and improving its readability; (ii) formatting the bibliography and suggesting a structure; (iii) summarizing and surveying the related literature; and (iv) writing and debugging the automation scripts that orchestrate the experiment runs.

In all cases, the authors made the final decisions: they chose the questions to test, the content to include, and the conclusions to draw; they verified, corrected, and approved all AI-assisted text, code, and scripts before use; and they are responsible for every claim in the paper. The experimental results, figures, and tables were produced by the authors' own runs and were not generated, altered, or selected by AI. The mathematical proofs were written and checked by the authors.

The authors take full responsibility for the final content of this work, including any text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, and no personal or sensitive data are used. All experiments rely exclusively on publicly available benchmark datasets (GLUE, MetaMathQA, GSM8K) and publicly released pretrained models (BERT-Base-Cased, RoBERTa-Base, and Qwen2.5). The methods and released artifacts do not pose foreseeable risks of harmful application beyond standard parameter-efficient fine-tuning, and the authors are not aware of any conflicts of interest. The authors adhere to the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

A reference implementation of the adaptive-rank kernel adapter accompanies this submission as the Code and Data Supplement (App. D.1), written in pure PyTorch and runnable on CPU, together with a test suite that checks the properties the construction claims: exact reduction to LoRA under a linear kernel, rank expansion and rank concentration under the adopted kernels, the effect of the rank controller and its gradient, and the control the criterion rejects. The supplement also carries the measurement files and the scripts behind the efficiency and kernel-curve figures, and a single-file training entry point that reruns the ARK-versus-LoRA comparison at a fixed parameter budget on a GLUE task, from the public checkpoint and dataset, with the two runs differing only in the kernel. Full experimental configurations, including hyperparameter grids and the median-over-seeds protocol, are given in App. D, and the dataset filtering and evaluation protocols in App. D.1. Every theoretical claim is accompanied by its assumptions and a complete proof in App. C (Theorems 5, 6, and 7). The code will be released publicly.

REFERENCES

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *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. 7319–7328, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.568. URL https://aclanthology.org/2021.acl-long.568.

Paul Albert, Frederic Z. Zhang, Hemanth Saratchandran, Cristian Rodriguez-Opazo, Anton van den Hengel, and Ehsan Abbasnejad. RandLoRA: Full rank parameter-efficient fine-tuning of large models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Zhi-Dong Bai, Yong-Qua Yin, et al. Limit of the smallest eigenvalue of a large dimensional sample covariance matrix. *Ann. Probab*, 21(3):1275–1294, 1993. doi: 10.1214/aop/1176989118.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In Smaranda Muresan, Preslav Nakov, and Aline

Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–9, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.1. URL https://aclanthology.org/2022.acl-short.1/.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv: 1308.3432*, 2013.

Ahmet Bilican, M Akın Yılmaz, A Murat Tekalp, and R Gökberk Cinbiş. Exploring sparsity for parameter efficient fine tuning using wavelets. *arXiv preprint arXiv:2505.12532*, 2025.

Massimo Bini, Leander Girrbach, and Zeynep Akata. Decoupling angles and strength in low-rank adaptation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=X1U74IwuxG.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York, 2006. ISBN 978-0-387-31073-2.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, and Amanda Askell. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998. doi: 10.1023/A:1009715923555.

André M. Carrington, Paul W. Fieguth, and Helen H. Chen. A new Mercer sigmoid kernel for clinical data classification. In *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 6397–6401, August 2014. doi: 10.1109/EMBC.2014.6945092.

Daria Cherniuk, Aleksandr Mikhalev, and Ivan Oseledets. Run LoRA run: Faster and lighter LoRA implementations. In Georg Rehm and Yunyao Li (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 200–207, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-288-6. doi: 10.18653/v1/2025.acl-industry.15. URL https://aclanthology.org/2025.acl-industry.15/.

Sankalan Pal Chowdhury, Adamos Solomou, Kumar Avinava Dubey, and Mrinmaya Sachan. Learning the transformer kernel. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=tLIBAEYjcv.

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.

Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, (3):326–334, 2006.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423/.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, March 2023. ISSN 2522-5839. doi: 10.1038/s42256-023-00626-4.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J. Clark, and Mehdi Rezagholizadeh. KronA: Parameter-Efficient Tuning with Kronecker Adapter. In Peyman Passban, Andy Way, and Mehdi Rezagholizadeh (eds.), *Enhancing LLM Performance: Efficacy, Fine-Tuning, and Inference Techniques*, pp. 49–65. Springer Nature Switzerland, Cham, 2025. ISBN 978-3-031-85747-8. doi: 10.1007/978-3-031-85747-8_4.

Alan Edelman. Eigenvalues and condition numbers of random matrices. *SIAM journal on matrix analysis and applications*, 9(4):543–560, 1988.

Xinlong Feng and Zhinan Zhang. The rank of a random matrix. *Applied mathematics and computation*, 185(1):689–694, 2007.

Ziqi Gao, Qichao Wang, Aochuan Chen, Zijing Liu, Bingzhe Wu, Liang Chen, and Jia Li. Parameter-Efficient Fine-Tuning with Discrete Fourier Transform. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 14884–14901. PMLR, July 2024.

Lev Goldfarb. A unified approach to pattern recognition. *Pattern Recognition*, 17(5):575–582, 1984.

Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press, 2016.

Bernard Haasdonk. Feature space interpretation of svms with indefinite kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):482–492, 2005.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 17783–17806. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/hayou24a.html.

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.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171–1220, 2008. doi: 10.1214/009053607000000677.

Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

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, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022.

Shengding Hu, Ning Ding, Weilin Zhao, Xingtai Lv, Zhen Zhang, Zhiyuan Liu, and Maosong Sun. OpenDelta: A Plug-and-play Library for Parameter-efficient Adaptation of Pre-trained Models. In Danushka Bollegala, Ruihong Huang, and Alan Ritter (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics: System Demonstrations, ACL 2023, Toronto, Canada, July 10-12, 2023*, pp. 274–281. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.ACL-DEMO.26.

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.

Nam Hyeon-Woo, Moon Ye-Bin, and Tae-Hyun Oh. FedPara: Low-rank Hadamard Product for Communication-Efficient Federated Learning. In *International Conference on Learning Representations*, 2022.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Ting Jiang, Shaohan Huang, Shengyue Luo, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, Qi Zhang, Deqing Wang, et al. Mora: High-rank updating for parameter-efficient fine-tuning. *arXiv preprint arXiv:2405.12130*, 2024.

Yeonjoon Jung, Daehyun Ahn, Hyungjun Kim, Taesu Kim, and Eunhyeok Park. Gralora: Granular low-rank adaptation for parameter-efficient fine-tuning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/89f8c6687c4adafe4d500444cd3db39a-Abstract-Conference.html.

Damjan Kalajdzievski. A Rank Stabilization Scaling Factor for Fine-Tuning with LoRA, November 2023. URL https://arxiv.org/abs/2312.03732.

Jiale Kang and Qingyu Yin. Balancing LoRA Performance and Efficiency with Simple Shard Sharing, May 2025. URL https://arxiv.org/abs/2409.15371.

Eugenia-Maria Kontopoulou, Gregory-Paul Dexter, Wojciech Szpankowski, Ananth Grama, and Petros Drineas. Randomized linear algebra approaches to estimate the von neumann entropy of density matrices. *IEEE transactions on information theory*, 66(8):5003–5021, 2020. doi: 10.1109/TIT.2020.2971991.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. VeRA: Vector-based Random Matrix Adaptation. In *The Twelfth International Conference on Learning Representations*, 2024.

Brian Lester, Rami Al-Rfou, and Noah Constant. The Power of Scale for Parameter-Efficient Prompt Tuning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.243.

Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *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, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353.

Yang Li, Shaobo Han, and Shihao Ji. VB-LoRA: Extreme Parameter Efficient Fine-Tuning with Vector Banks. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, November 2024.

Dongze Lian, Daquan Zhou, Jiashi Feng, and Xinchao Wang. Scaling & shifting your features: A new baseline for efficient model tuning. *Advances in Neural Information Processing Systems*, 35: 109–123, 2022.

Hsuan-Tien Lin and Chih-Jen Lin. A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods. Technical report, Department of Computer Science, National Taiwan University, 2003. URL https://www.csie.ntu.edu.tw/~cjlin/papers/tanh.pdf.

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, December 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*, June 2024a.

Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, Yandong Wen, Michael J. Black, Adrian Weller, and Bernhard Schölkopf. Parameter-Efficient Orthogonal Finetuning via Butterfly Factorization. In *The Twelfth International Conference on Learning Representations*, 2024b.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. GPT understands, too. *AI Open*, 5:208–215, January 2024c. ISSN 2666-6510. doi: 10.1016/j.aiopen.2023.08.012.

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

Gaëlle Loosli, Stéphane Canu, and Cheng Soon Ong. Learning svm in krein spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(6):1204–1216, 2016. doi: 10.1109/TPAMI.2015.2477830.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Ao Luo, Fan Yang, Xin Li, and Shuaicheng Liu. Learning optical flow with kernel patch attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8906–8915, 2022.

Gen Luo, Minglang Huang, Yiyi Zhou, Xiaoshuai Sun, Guannan Jiang, Zhiyu Wang, and Rongrong Ji. Towards efficient visual adaption via structural re-parameterization. *arXiv preprint arXiv:2302.08106*, 2023.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. PEFT: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. PiSSA: Principal Singular Values and Singular Vectors Adaptation of Large Language Models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, November 2024.

Charles A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986. doi: 10.1007/BF01893414.

Nikhil Shivakumar Nayak, Krishnateja Killamsetty, Ligong Han, Abhishek Bhandwaldar, Prateek Chanda, Kai Xu, Hao Wang, Aldo Pareja, Oleg Silkin, Mustafa Eyceoz, and Akash Srivastava. Sculpting Subspaces: Constrained Full Fine-Tuning in LLMs for Continual Learning. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=vQcyqsGJDw.

Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Jiri Hron, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations*, 2019.

Cheng Soon Ong, Xavier Mary, Stéphane Canu, and Alexander J Smola. Learning with non-positive kernels. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, pp. 639–646, 2004. doi: 10.1145/1015330.1015443.

Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010. ISSN 1041-4347. doi: 10.1109/TKDE.2009.191.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8024–8035, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

Zeju Qiu, Weiyang Liu, Haiwen Feng, Yuxuan Xue, Yao Feng, Zhen Liu, Dan Zhang, Adrian Weller, and Bernhard Schölkopf. Controlling text-to-image diffusion by orthogonal finetuning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, pp. 79320–79362, Red Hook, NY, USA, December 2023. Curran Associates Inc.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=Hkuq2EkPf.

Olivier Roy and Martin Vetterli. The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference*, pp. 606–610. IEEE, 2007.

Mark Rudelson and Roman Vershynin. Sampling from large matrices: An approach through geometric functional analysis. *Journal of the ACM (JACM)*, 54(4):21–es, 2007.

Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

Shufan Shen, Junshu Sun, Xiangyang Ji, Qingming Huang, and Shuhui Wang. Expanding Sparse Tuning for Low Memory Usage. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, November 2024.

Gilbert Strang. *Linear algebra and its applications*. 2012.

Tanya Strydom, Giulio V. Dalla Riva, and Timothée Poisot. SVD Entropy Reveals the High Complexity of Ecological Networks. *Frontiers in Ecology and Evolution*, 9:623141, June 2021. ISSN 2296-701X. doi: 10.3389/fevo.2021.623141.

George PH Styan. Hadamard products and multivariate statistical analysis. *Linear algebra and its applications*, 6:217–240, 1973.

Terence Tao. *Topics in random matrix theory*, volume 132. American Mathematical Society, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models, February 2023. URL https://arxiv.org/abs/2302.13971.

Shashanka Ubaru and Yousef Saad. Fast methods for estimating the numerical rank of large matrices. In *International Conference on Machine Learning*, pp. 468–477. PMLR, 2016.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 3274–3287, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Roman Vershynin. *Introduction to the non-asymptotic analysis of random matrices*, pp. 210–268. Cambridge University Press, May 2012. ISBN 9780511794308. doi: 10.1017/cbo9780511794308.006. URL http://dx.doi.org/10.1017/CBO9780511794308.006.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rJ4km2R5t7.

Jindong Wang and Yiqiang Chen. *Introduction to Transfer Learning: Algorithms and Practice*. Machine Learning: Foundations, Methodologies, and Applications. Springer Nature Singapore, Singapore, 2023. ISBN 978-981-19-7583-7 978-981-19-7584-4. doi: 10.1007/978-981-19-7584-4.

Ruoxi Wang, Yingzhou Li, and Eric Darve. On the numerical rank of radial basis function kernels in high dimensions. *SIAM J. Matrix Anal. Appl.*, 39(4):1810–1835, 2018. doi: 10.1137/17M1135803. URL https://doi.org/10.1137/17M1135803.

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.

Xiaojing Weng, Altai Perry, Michael Maroun, and Luat T. Vuong. Singular value decomposition and entropy dimension of fractals. In *2022 International Conference on Image Processing, Computer Vision and Machine Learning (ICICML)*, pp. 427–431. IEEE, October 2022. doi: 10.1109/icicml57342.2022.10009680. URL http://dx.doi.org/10.1109/ICICML57342.2022.10009680.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pp. 370–378. PMLR, 2016.

Zhihao Wu, Zhao Zhang, and Jicong Fan. Graph convolutional kernel machine versus graph convolutional networks. *Advances in neural information processing systems*, 36:19650–19672, 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, Tianyi Tang, 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, 2025. URL https://arxiv.org/abs/2412.15115.

Greg Yang and Edward J. Hu. Tensor programs IV: feature learning in infinite-width neural networks. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 11727–11737. PMLR, 2021. URL http://proceedings.mlr.press/v139/yang21c.html.

Shih-Ying Yeh, Yu-Guan Hsieh, Zhidong Gao, Bernard B. W. Yang, Giyeong Oh, and Yanmin Gong. Navigating Text-To-Image Customization: From LyCORIS Fine-Tuning to Model Evaluation. In *The Twelfth International Conference on Learning Representations*, 2024.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=N8N0hgNDRt.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023a.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning. In *The Eleventh International Conference on Learning Representations*, 2023b.

Renrui Zhang, Jiaming Han, Chris Liu, Aojun Zhou, Pan Lu, Yu Qiao, Hongsheng Li, and Peng Gao. Llama-adapter: Efficient fine-tuning of large language models with zero-initialized attention. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024a. URL https://openreview.net/forum?id=d4UiXAHN2W.

Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5048–5060, 2024b.

Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization. In *Proceedings of the computer vision and pattern recognition conference*, pp. 14901–14911, 2025.

CONTENTS OF THE APPENDIX

A Notation **19**

B PEFT: Definitions and Metrics **19**

B.1 A Formal Definition of PEFT 20

B.2 Metrics for Evaluating Parameter Efficiency 20

B.3 Tuning Space and Mergeability 21

C Theory and Empirical Validation of Adaptive-Rank Kernels **22**

C.1 Quantitative Indicators for Measuring the Rank of Empirical Matrices 22

C.2 The Geometry of the Induced Rank 25

C.3 ARK Formulation and Kernel Specifications 26

C.4 Functional Form of the Rank Scaling Law: Candidate Families and Model Selection 30

C.5 Empirical Attained-Rank Ranges at Fixed Budget 33

C.6 Statistical Validation of Rank Scaling Law Stability 33

C.7 Analysis of the Polynomial Kernel as an ARK Candidate 34

C.8 Proof that Piecewise Linear Function Cannot Serve as Valid Mercer Kernel 35

C.9 The Rank Ladder: Two Necessary Conditions for a Rank Controller 36

C.10 Algebraic Bounds on the Attainable Rank 38

D Experimental Details and Additional Results **39**

D.1 Details on Datasets and Evaluation Protocols 39

D.2 Details on Compared Methods 40

D.3 Implementation and Hyperparameter Details 41

D.4 Backbone Transfer: GLUE on RoBERTa-Base 42

D.5 More Ablation Studies 43

D.6 Rank Trajectories under a Dense Evaluation Grid 43

D.7 What the Kernel Costs, and Why 45

E Positioning and Clarifications **48**

E.1 Relation to Kernelized LoRA and Other Adaptive PEFT 48

E.2 Disambiguating “Kernel” in Computer Science 49

E.3 Activation Functions in Neural Architectures 49

A NOTATION

Roadmap. App. B fixes the quantities and cost metric the main text assumes; App. C develops and validates the theory of ARK; App. D reports experimental details and additional results; App. E clarifies terminology and positioning.

The table below collects the symbols used throughout the paper, grouped in the order in which they are introduced. Each row points to where the symbol is defined; this page introduces nothing new of its own.

Symbol	Meaning	Defined
Network, parameters and what is being tuned		
$\mathcal{F}(W; X)$	the network's forward map	Eq. (6)
$G(W, \Delta V; X)$	its PEFT-augmented form	Eq. (6)
$W,\ W_s,\ W_t$	weights; after pre-training; after full fine-tuning	Eq. (4)
ΔV	parameters a PEFT method introduces and tunes	Eq. (7)
$\lvert\cdot\rvert$	parameter count of its argument	Eq. (8)
TPR	trainable parameter ratio; method-dependent denominator, so we use RPC	Eq. (8)
RPC	relative parameter cost $\lvert\Delta V\rvert/\lvert W\rvert$: our budget measure	Eq. (9)
Adaptation update and the two knobs		
ΔW	induced update of a frozen matrix; synthesized, never stored	Sec. B.1
$\Delta W = AB^T$	LoRA update, $A \in \mathbb{R}^{d_{in} \times r}$ and $B \in \mathbb{R}^{d_{out} \times r}$	Sec. B.1
r	latent dimension of the factors: budget knob, not the rank	Sec. B.1
d_{in}, d_{out}, D	layer dimensions and the structural rank ceiling $D = \min(d_{in}, d_{out})$	Sec. 3
Kernels: the scalar form and the matrix form		
$k(u, v)$	scalar kernel on two row vectors; every matrix kernel builds on it	Sec. 4.1
$K(A, B)$	matrix kernel, the *cross-kernel form*: $[K(A, B)]_{ij} = k(A_{i*}, B_{j*})$	Def. 2
$K(X, X),\ n_{\text{self}}$	the same kernel with both arguments equal: the *self-kernel form* of order n_{self}	Def. 4
$\Delta W = K_\Theta(A, B)$	the ARK update	Eq. (3)
$\Theta = \{\alpha, \beta, \gamma, \delta\}$	structural parameters of the kernel	Eq. (3)
γ	pre-activation scale, the learnable rank controller (initial 0.5)	Eq. (3)
The rank a kernel induces		
σ^2	variance of the i.i.d. rows used to probe a kernel	Def. 2
$\rho(K, \sigma^2)$	Rank Scaling Density: expected induced rank over r	Def. 2
$\rho^{\text{self}}(K)$	the same ratio on the self-kernel form	Def. 4
$\mathbf{R}_K$	random numerical rank of $K(A, B)$	Def. 2
rank-stable	concentration of $\mathbf{R}_K$, defining the Adaptive-Rank Kernel class	Def. 3
Rank indicators for an empirical matrix		
$\sigma_1 \geq \sigma_2 \geq \cdots$	singular values of ΔW	Sec. C.1
$\text{rank}(\cdot)$	exact algebraic rank: default inside proofs	Eq. (14)
$\text{rank}_{\text{num}}(\cdot)$	numerical rank: default in every measured result	Eq. (15)
$\text{rank}_\tau(\cdot),\ \tau$	rank at tolerance τ; $\tau_{\text{num}} = \max(m, n)\,\epsilon$	Eq. (16)
$\text{srank}(\cdot)$, $\text{erank}(\cdot)$	stable rank and effective rank	Eq. (17), (18)
$\kappa(\cdot)$	condition number above the tolerance	Sec. C.1
The two competing scaling behaviours		
$f(\lvert\Delta V\rvert)$	budget $\to$ rank; the Budget-Coupled Rank Scaling Law only	Def. 1
$g(\Theta)$	controller $\to$ rank; the Iso-Budget Rank Scaling Law only, $g = r\rho$	Def. 1

B PEFT: DEFINITIONS AND METRICS

To understand parameter-efficient fine-tuning (PEFT), it is helpful to deconstruct the term itself. **"Fine-Tuning"** places PEFT within the broader paradigm of **transfer learning** (Pan & Yang, 2010; Wang & Chen, 2023), where knowledge from a large pre-trained model is adapted to downstream tasks. The concept of PEFT was first proposed by (Houlsby et al., 2019) to describe an **optimization process** that is **"Parameter-Efficient"**. The goal is to minimize the number of tunable parameters, thereby reducing the computational and storage resources required. This approach is also known as delta-tuning (Ding et al., 2023).

In this section, we make these concepts precise. We first offer a formal definition of PEFT as a specialized pre-training and fine-tuning methodology, then establish the metrics used to quantify its parameter efficiency, and finally fix the axis along which we classify methods and the sense in which an update is mergeable.

B.1 A Formal Definition of PEFT

Let us consider a standard pre-training and full fine-tuning process. Given a model architecture $\mathcal{F}$ with parameters W, pre-training on a source domain dataset (X_s, Y_s) yields $W_s = \arg\min_W \mathcal{L}(F(W; X_s), Y_s)$. Full fine-tuning then adapts the model to a target domain dataset (X_t, Y_t) by optimizing all parameters, starting from W_s:

$$W_t = \arg\min_W \mathcal{L}(\mathcal{F}(W; X_t), Y_t), \quad \text{initialized with } W_s \tag{4}$$

PEFT methods introduce a new paradigm by reparameterizing the original network $\mathcal{F}$. They create an augmented network G, which includes a small set of new adaptive parameters, referred to as **"delta parameters"** in the delta-tuning literature (Ding et al., 2023). In this paper, we denote these parameters as ΔV. This new network $G(W, \Delta V; X)$ is designed to be a superset of F, which implies that the augmented network can revert to the original network's behavior. We term this requirement as the **degeneracy condition**[6]:

$$\forall W, \forall X, \quad \exists \Delta V_0 \text{ such that } G(W, \Delta V_0; X) = \mathcal{F}(W; X) \tag{5}$$

The fine-tuning optimization problem in Equation 4 can then be reformulated over the augmented network:

$$W_t, \Delta V_t = \arg\min_{W, \Delta V} \mathcal{L}(G(W, \Delta V; X_t), Y_t) \tag{6}$$

The core principle of PEFT is to impose a crucial constraint on this optimization: the original pre-trained weights W_s are frozen ($W_t = W_s$). This simplifies the optimization problem to only learning the newly introduced delta parameters ΔV:

$$\Delta V_t = \arg\min_{\Delta V} \mathcal{L}(G(W_s, \Delta V; X_t), Y_t) \tag{7}$$

This formulation precisely defines the goal of PEFT: to adapt a large model to new tasks by tuning only a small, manageable set of incremental parameters.

Two instantiations make ΔV concrete, both of which the main text compares against. In LoRA the frozen W are the pre-trained projection matrices and $\Delta V = \{A, B\}$, so $|\Delta V| = (d_{in} + d_{out})\, r$ per adapted matrix. The induced weight update $\Delta W = AB^T$ is a $d_{in} \times d_{out}$ matrix that is synthesized from ΔV and never stored as parameters, which is why a cost metric must count $|\Delta V|$ and not $|\Delta W|$: the two differ by orders of magnitude, and conflating them is what makes "rank" and "parameter budget" appear to be the same knob. In AdaLoRA the factors are augmented by a diagonal Λ and the importance statistics that drive pruning, so ΔV additionally carries the singular-value vector; pruning changes which entries survive but not the budget provisioned at the start, so its cost is set by the initial rank rather than the surviving one. ARK keeps LoRA's ΔV unchanged and alters only how ΔW is synthesized from it.

B.2 Metrics for Evaluating Parameter Efficiency

Comparing parameter budgets across PEFT families requires a metric whose denominator does not shift with the method. We first show where the commonly used "trainable parameter ratio" (TPR) fails this requirement, then define the Relative Parameter Cost (RPC) used throughout the paper.

[6]This condition is met through different initialization strategies. For example, LoRA (Hu et al., 2022) ensures the initial update is null by initializing one of its factorized matrices to zero. In contrast, methods like PiSSA (Meng et al., 2024) and LoRA-GA (Wang et al., 2024) employ a compensatory strategy, initializing the new base weights as $W_0 = W_s - A_0 B_0^T$ to ensure the effective weights remain unchanged at the start of training. Similarly, in prompt-based methods, LLaMA-Adapter (Zhang et al., 2024a) uses a zero-initialized gating mechanism to ensure the adaptation has no effect at the start of training. The symmetric residual initialization marked † in the main text is the same compensatory device applied to our kernel, $W_0 = W_s - K_\gamma(A_0, B_0)$; we keep it as a first-class variant because the matrix kernel of Sec. 4.1 is symmetric in A and B whenever the underlying k is, so zeroing one factor is not the only natural way to meet this condition (App. C.2).

The Ambiguity of the Common "Trainable Parameter Ratio" (TPR) We begin by highlighting the inconsistencies found in the commonly used "trainable parameter ratio." A formal definition of TPR would be:

$$\text{TPR} = \frac{|\Delta V|}{|W \cup \Delta V|} \tag{8}$$

However, the denominator, $|W \cup \Delta V|$, behaves inconsistently across different PEFT families. For *selective* methods, the tuned parameters are a subset of the original weights ($\Delta V \subset W$), making the denominator simply $|W|$. For classic *additive* methods like Adapters (Houlsby et al., 2019), new parameters are introduced by injecting new modules, such as small feed-forward layers, into the original architecture. It is further compounded by the common practice of tuning task-specific components, such as a classification head, which belong to ΔV but not to W. TPR is therefore unreliable for comparing budgets across families.

Relative Parameter Cost (RPC) To resolve this issue, we propose the **Relative Parameter Cost (RPC)** as our primary metric for a fair and standardized comparison. RPC measures the "cost" of a PEFT method by normalizing the number of tunable parameters, $|\Delta V|$, against the scale of the original model, $|W|$:

$$\text{RPC} = \frac{|\Delta V|}{|W|} \tag{9}$$

This metric directly answers the question: "What fraction of the original model's parameter budget is required for adaptation?" By consistently using the original model size $|W|$ as its baseline, RPC avoids the issues inherent in TPR. Its reciprocal $|W|/|\Delta V|$ is the "how many times more efficient than full fine-tuning" phrasing common in the literature; we report RPC rather than the reciprocal because the tables compare costs that differ by orders of magnitude, where a small fraction is easier to read than a large factor.

The two metrics are moreover ordered. Since $W \subseteq W \cup \Delta V$ implies $|W| \leq |W \cup \Delta V|$, and both definitions share the numerator $|\Delta V|$, we have

$$\text{TPR} = \frac{|\Delta V|}{|W \cup \Delta V|} \leq \frac{|\Delta V|}{|W|} = \text{RPC}, \tag{10}$$

with equality exactly when $\Delta V \subseteq W$, i.e. for *selective* methods that tune a subset of the existing weights. For *additive* methods the gap widens with the size of the injected modules, so TPR systematically understates the adaptation budget. Reporting RPC is therefore the conservative choice, since any efficiency claim that holds under RPC also holds under the more permissive TPR.

RPC Composes Locally RPC has a property analogous to Amdahl's law, which is what makes a single global number meaningful. PEFT methods act on selected sub-networks, for instance the attention projections, so the model-level RPC is the sum of per-module costs weighted by each module's share of $|W|$. The global cost is bounded by the fraction of the network that is adapted at all, since every untouched module contributes zero. A method can therefore be aggressive locally, as ours is when it raises the numerical rank of an adapted matrix, while remaining negligible globally.

B.3 Tuning Space and Mergeability

Transfer learning is commonly organised by asking "what to transfer?" (Pan & Yang, 2010), and fine-tuning answers "the pre-trained parameters W" (Wang & Chen, 2023). Every PEFT method gives that same answer, so the question does not separate them. What separates them is the dual question, "what to tune?", i.e. where the delta parameters ΔV of Eq. 7 act. We therefore label a method **Parameter-Space Adaptation** when ΔV modifies the weights, $G(W, \Delta V; X) = F(W \oplus \Delta V; X)$ for a combination operator $\oplus$ such as the matrix addition of LoRA (Hu et al., 2022), and **Feature-Space Adaptation** when it modifies intermediate activations instead, as Adapters and prompt-based methods do (Lester et al., 2021). This is the axis along which Tab. 11 is organised. We use it in place of the usual Additive/Selective/Reparameterized split because that split mixes the tuning objective with a structural property: LoRA is called "reparameterized" yet also "adds" trainable matrices, so the categories overlap. Mergeability is better treated as a property to be established, which is what the remainder of this subsection does.

Feature-Parameter Equivalence: A More Precise Concept The ability of some methods to be merged at inference time is a powerful property, but "reparameterizable" is an imprecise term for it, since in a broader context all PEFT methods are reparameterizing Eq. 4 with Eq. 6. We propose a more formal concept: **Feature-Parameter Equivalence**. A method possesses this property if its intended modification, whether at the parameter or feature level, can be mathematically expressed in the other form.

More formally, a method exhibits this equivalence if its parameter-level update $W' = W \oplus \Delta V$ can be expressed as an equivalent feature-level modification $y = \mathcal{F}(W; X) \oplus g(\Delta V; X)$, and vice-versa. This property is satisfied if the local network function f satisfies a condition analogous to Cauchy's functional equation for linearity. For a linear layer where $\mathcal{F}(W; X) = XW$, this condition is met:

$$\mathcal{F}(W + \Delta V; X) = X(W + \Delta V) \tag{11}$$
$$= XW + X\Delta V \tag{12}$$
$$= \mathcal{F}(W; X) + \mathcal{F}(\Delta V; X) \tag{13}$$

This equivalence is bidirectional. For instance, LoRA is designed as a **Parameter-Space Adaptation** method with the objective of finding an optimal update $\Delta V = AB^T$. Its parameter-level modification $h = X(W + AB^T)$ is functionally equivalent to the feature-level modification $h = XW + X(AB^T)$, which allows it to be implemented efficiently during training(Cherniuk et al., 2025). Conversely, methods like IA^3(Liu et al., 2022), SSF(Lian et al., 2022), and RepAdapter(Luo et al., 2023) are designed as **Feature-Space Adaptation** methods that scale activations. However, because these scaling operations are linear, they can be mathematically folded into the weight matrix W, making them equivalent to a parameter-tuning method at inference time. This fundamental mathematical relationship is what enables inference-time merging, and we argue that "Feature-Parameter Equivalence" is a more precise concept than "reparameterizability" as it shifts the focus from a procedural step to the core functional property.

Tab. 11 categorizes representative PEFT methods along these lines, and Sec. E.1 traces the evolution of the adaptive ones. ARK is a **Parameter-Space Adaptation** method: although it applies a non-linearity (e.g., Tanh or Swish) during training to expand the numerical rank, that non-linearity acts exclusively inside the parameter space, on LoRA's own decomposition, to synthesize $\Delta W = K_\Theta(A, B)$. It therefore satisfies Feature-Parameter Equivalence, so once training concludes the expanded high-rank ΔW is computed once and folded into the frozen W, leaving no additional latency or architectural overhead at inference.

C Theory and Empirical Validation of Adaptive-Rank Kernels

C.1 Quantitative Indicators for Measuring the Rank of Empirical Matrices

To investigate the Rank-Dilemma (Sec. 3) and to validate the empirical scaling behavior of Rank Scaling Kernels (Sec. 4.1), we need measurement tools before we need theorems. This subsection therefore fixes notation and the rank indicators used throughout the paper, then establishes the result the main text cites, namely that **a standard low-rank update cannot adapt its own rank** (the heading *Why LoRA Cannot Adapt Its Own Rank* below), and finally states what each indicator can and cannot certify in floating point.

C.1.1 Definitions

Throughout, $\Delta W \in \mathbb{R}^{m \times n}$ has singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0$, obtained from the decomposition $\Delta W = U\Sigma V^T$.

Exact Rank. In linear algebra the rank is the dimension of the column space, equivalently the number of non-zero singular values (Golub & Van Loan, 2013; Strang, 2012):

$$\operatorname{rank}(\Delta W) = \sum_i \mathbb{I}(\sigma_i > 0). \tag{14}$$

Numerical Rank. Following the convention adopted by standard libraries (Paszke et al., 2019),[7] the numerical rank (Ubaru & Saad, 2016; Wang et al., 2018) replaces the test against zero by a test against a relative tolerance:

$$\mathrm{rank}_{\mathrm{num}}(\Delta W) = \sum_i \mathbb{I}\big(\sigma_i > \sigma_1 \cdot \max(m,n) \cdot \epsilon\big), \tag{15}$$

where ϵ is the machine epsilon of the matrix's data type. It asks which directions are distinguishable from rounding error. **This is the quantity we mean by "numerical rank" throughout the paper, and by "rank" whenever we report a measured value.**

Coarse Tolerances. Eq. (15) is one member of a family that we will need in full later, since a kernel-induced matrix has to be judged by a criterion that does not depend on the float type. For a tolerance $\tau \in (0,1)$ write

$$\mathrm{rank}_{\tau}(\Delta W) = \sum_i \mathbb{I}\big(\sigma_i > \tau\, \sigma_1\big), \tag{16}$$

so that the numerical rank is the member of this family obtained at $\tau = \tau_{\mathrm{num}} := \max(m,n)\,\epsilon$, that is, $\mathrm{rank}_{\mathrm{num}} = \mathrm{rank}_{\tau_{\mathrm{num}}}$. We write the numerical rank with the subscript num rather than ϵ precisely because its tolerance is $\max(m,n)\,\epsilon$ and not ϵ itself, so a subscript ϵ would invite the reading $\tau = \epsilon$. From App. C.2 onwards we also use coarse values, τ of order 10^{-3} to 10^{-1}, which ask instead which directions carry non-negligible energy. The two are not interchangeable, and we say which one is meant wherever it matters: whenever a conclusion depends on the value of τ, we state that value and report the sensitivity.

Rank convention for proofs and measurements. Unless a theorem, proof, or equation explicitly specifies rank_{τ} or numerical rank, an unqualified rank in a mathematical argument denotes the exact algebraic rank, that is, the rank over exact arithmetic. In empirical plots, tables, and measured results, an unqualified "rank" denotes the PyTorch-style numerical rank of Eq. (15), unless another indicator or tolerance is stated explicitly. Thus, the same word is interpreted by the surrounding context: mathematical arguments use exact/algebraic rank by default, whereas empirical reports use numerical rank by default. Bounds transferred from algebraic rank to a numerical-rank claim are understood as upper bounds only and do not imply equality.

Stable Rank. The stable rank (Rudelson & Vershynin, 2007) is the ratio of squared spectral energies,

$$\mathrm{srank}(\Delta W) = \frac{\|\Delta W\|_F^2}{\|\Delta W\|_2^2} = \frac{\sum_i \sigma_i^2}{\sigma_1^2}, \tag{17}$$

and satisfies $1 \le \mathrm{srank}(\Delta W) \le \mathrm{rank}(\Delta W)$. It uses no threshold, and because each direction is weighted by σ_i^2 it is insensitive to small singular values. One caveat is worth stating in advance, because the two properties are easy to conflate: being threshold-free makes the stable rank robust to the choice of τ, but it does not make it robust across random draws. It divides by the extreme order statistic σ_1, whose relative fluctuation enters the ratio with weight -2, so its seed-to-seed variance is systematically larger than that of the integer counts, which are locally constant under small perturbations. App. C.6 reports both.

Effective Rank. The effective rank (Roy & Vetterli, 2007) is a continuous surrogate based on the Shannon entropy of the normalized spectrum,

$$w_i = \frac{\sigma_i + \zeta}{\sum_j (\sigma_j + \zeta)}, \quad \mathrm{erank}(\Delta W) = \exp\Big(-\sum_i w_i \log w_i\Big), \tag{18}$$

with ζ a small constant added for numerical stability, distinct from the machine epsilon ϵ of Eq. (15). Up to that regulariser it obeys $1 \le \mathrm{erank}(\Delta W) \le \mathrm{rank}(\Delta W)$, the upper bound being attained when the energy is spread evenly over the non-zero directions; a higher value therefore indicates a more uniform use of the available dimensions.[8]

[7] PyTorch's `matrix_rank`: https://docs.pytorch.org/docs/2.8/generated/torch.linalg.matrix_rank.html

[8] The effective rank is called **SVD Entropy** in parts of the literature, with slight differences in scaling (Strydom et al., 2021; Weng et al., 2022). It is distinct from the **Von Neumann Entropy**, which characterises density matrices by their eigenvalues (Kontopoulou et al., 2020).

Condition Number. We also report $\kappa(\Delta W) = \sigma_1/\sigma_{\min}$ (Goodfellow et al., 2016) as a measure of ill-conditioning, with $\sigma_{\min}$ the smallest singular value *above the tolerance* of Eq. (15); including the sub-tolerance ones would make κ infinite.

C.1.2 Why LoRA Cannot Adapt Its Own Rank: the Rank of Stochastic AB^T

A common misconception is that the original LoRA may adaptively learn the matrix rank through gradient descent without mechanisms like AdaLoRA or ARK. This is incorrect: the rank of the LoRA update is confined to $[0, r]$ and, in practice, sits at r. Below we separate what holds unconditionally from what holds generically.

Proposition 1 (the cap is unconditional). For any $A \in \mathbb{R}^{d_{in} \times r}$ and $B \in \mathbb{R}^{d_{out} \times r}$,

$$\operatorname{rank}(AB^T) \;\le\; \min\big(\operatorname{rank}(A), \operatorname{rank}(B)\big) \;\le\; r. \tag{19}$$

Indeed, every column of AB^T is a linear combination of the columns of A, so the column space of AB^T is contained in that of A; transposing the same argument bounds it by $\operatorname{rank}(B)$. Both factors have only r columns, whence the second inequality. □

No distributional or optimization assumption enters Eq. (19): it holds at initialization, at every training step, and for every optimizer. What the main text needs in addition is that gradient descent does not settle below the cap.

Proposition 2 (the cap is attained in general position). If the joint law of (A, B) is absolutely continuous with respect to the Lebesgue measure on $\mathbb{R}^{d_{in} \times r} \times \mathbb{R}^{d_{out} \times r}$, then $\operatorname{rank}(AB^T) = r$ almost surely. A matrix A fails to have full column rank exactly when all of its $r \times r$ minors vanish, *i.e.* when it lies in the common zero set of finitely many non-trivial polynomials. That set is a proper algebraic subvariety of $\mathbb{R}^{d_{in} \times r}$ and therefore Lebesgue-null (Tao, 2023), so an absolutely continuous law assigns it probability zero; the same holds for B. Off this null set, Sylvester's rank inequality (Horn & Johnson, 2012) with shared inner dimension r gives

$$\operatorname{rank}(A) + \operatorname{rank}(B) - r \;\le\; \operatorname{rank}(AB^T), \tag{20}$$

i.e. $r \le \operatorname{rank}(AB^T)$, which together with Eq. (19) forces equality. □

What Proposition 2 does and does not cover. The hypothesis is a statement about a distribution, so it is worth being explicit about where it applies. It holds at step 0 under the symmetric residual initialization ($W_0 = W - K_\gamma(A, B)$), where both factors are Gaussian, so the update starts at rank exactly r. Under the standard asymmetric initialization the update starts at rank zero, since $B = 0$. And it does not transfer automatically to (A_t, B_t) after t optimizer steps, whose law is a deterministic image of the initialization and need not remain absolutely continuous. Proposition 2 therefore characterises the law at initialization; for the trajectory we rely on the two statements below, one algebraic and one measured.

First, the asymmetric case leaves the cap immediately. Writing $\nabla_{\Delta W} L = \partial L / \partial \Delta W$, back-propagation through $\Delta W = \alpha\gamma AB^T$ gives $\nabla_B L = \alpha\gamma\,(\nabla_{\Delta W} L)^T A$, so after one step $B_1 = -\eta\,\Psi\big(\alpha\gamma\,(\nabla_{\Delta W} L)^T A\big)$ with Ψ the optimizer map. Under plain SGD ($\Psi = \mathrm{id}$), $\operatorname{rank}(B_1) = \operatorname{rank}\big((\nabla_{\Delta W} L)^T A\big) \le \min\big(r, \operatorname{rank}(\nabla_{\Delta W} L)\big)$, and since $\nabla_{\Delta W} L$ for a linear layer is a sum of per-token outer products, $\operatorname{rank}(\nabla_{\Delta W} L) \le T$ for a batch of T tokens: the bound r is reached once $T \ge r$ and no degeneracy occurs, while a batch with fewer than r tokens can leave B_1 rank-deficient. Under AdamW the update is an elementwise normalization of the gradient, which does not preserve the product structure and generically restores full column rank.

Second, and more directly, the equality is measured rather than assumed: the LoRA-equivalent kernel of Fig. 3 in the main paper sits at exactly $32 = r$ on all 24 layer–module pairs and on every task after training. The claim the main text uses is the conjunction of the two, that Eq. (19) makes r an upper bound as a matter of algebra, and that the measurement shows the bound saturated rather than undershot.

Regardless of whether the downstream task calls for a high-rank or a low-rank update, then, the rank of a standard low-rank parameterization is confined to $[0, r]$ and observed at r; it has no mechanism for scaling itself either upward or downward. In contrast, ARK uses the kernel to extend the shared inner dimension implicitly, without changing the parameter budget $|\Delta V|$.

C.1.3 Why the Exact Definition Fails on Computed Matrices

Deep learning does not evaluate Eq. (14) in exact arithmetic. Rounding error perturbs every entry of a computed product, so singular values that are zero in theory are returned as small positive numbers, and Eq. (14) reports almost any computed matrix as full rank (Feng & Zhang, 2007). The exact definition is thus uninformative on empirical matrices, which is why we report Eq. (15) instead. Two directions of error have to be excluded, and they need different arguments.

A low reported rank is not an artifact of rounding. For a strictly parameter-bounded product such as AB^T, the spurious singular values stay below the tolerance for a quantitative reason. Writing $\widehat{\Delta W}$ for the computed product, Weyl's inequality gives $|\sigma_i(\widehat{\Delta W}) - \sigma_i(\Delta W)| \leq \|\widehat{\Delta W} - \Delta W\|_2$ for every i, and the backward-error analysis of matrix multiplication bounds the right-hand side by $O(r\epsilon)\|A\|_2\|B\|_2$ (Golub & Van Loan, 2013). The singular values that vanish in theory are therefore of order $\epsilon\|A\|_2\|B\|_2$, which exceeds the reference scale $\epsilon\sigma_1$ of Eq. (15) only by the alignment factor $\|A\|_2\|B\|_2/\sigma_1(AB^T)$. That factor is large only when the leading singular directions of the two factors are nearly orthogonal, and at our dimensions the $\max(m, n)$ term in the tolerance leaves ample margin for it; the measured spectra show a gap of several orders of magnitude there.

A high reported rank is not an artifact of the threshold. Here we do not appeal to results for *i.i.d.* random matrices, which do not apply to a trained ΔW. Instead we report the stable rank of Eq. (17) alongside the numerical rank. Being a ratio of spectral energies it cannot be inflated by ϵ-level singular values, and it is bounded above by the rank itself, so a high stable rank certifies genuinely independent directions without reference to any tolerance. Classical results for *i.i.d.* ensembles are consistent with this picture, in that the least singular value concentrates away from zero for aspect ratios bounded away from one (Bai et al., 1993; Vershynin, 2012; Edelman, 1988), but we do not rely on them.

C.2 The Geometry of the Induced Rank

In Sec. 4.1, we measured the rank a kernel induces from the *cross-kernel form* $K(A, B)$, which evaluates the kernel across two different sets of vectors and thereby matches the low-rank adaptation structure $\Delta W = AB^T$. To relate that criterion to traditional Mercer kernel theory, we consider the *self-kernel form* obtained by setting $A = B = X$, which is the object classical kernel theory works with (Schölkopf & Smola, 2002), and define the self-kernel Rank Scaling Density of K from $K(X, X)$.

Definition 4 (Self-kernel Rank Scaling Density ρ^{self})**.** *Let $X \in \mathbb{R}^{n_{\text{self}} \times r}$, with n_{self} the number of rows the self-kernel is formed over, have row vectors drawn i.i.d. from a Gaussian distribution $\mathcal{N}(0, \sigma^2 I)$, where $r \ll d$. The numerical rank of the resulting self-kernel matrix, $\operatorname{rank}(K(X, X))$ in the sense of Eq.* (15)*, is a random variable and the **Rank Scaling Density** $\rho^{\text{self}}(K, \sigma^2)$ is defined as the expected scaling ratio:*

$$\rho^{\text{self}}(K, \sigma^2) = \frac{\mathbb{E}[\operatorname{rank}(K(X, X))]}{r} \tag{21}$$

Similar to $\rho(K)$, we denote the standard $\rho^{\text{self}}(K, 1^2)$ as $\rho^{\text{self}}(K)$, and we call ρ^{self} the self-kernel Rank Scaling Density of K.

In the self-kernel form the matrix $K(X, X)$ is the object on which classical kernel theory operates, and the rank we measure has an exact geometric meaning. It is worth deriving that meaning without assuming positive semi-definiteness, because the kernels we retain are not PSD and an argument routed through Mercer's theorem would not cover them. We avoid calling $K(X, X)$ a Gram matrix here, since in App. C.8 that term carries its strict sense of a PSD matrix.

Let $S = K(X, X) \in \mathbb{R}^{n_{\text{self}} \times n_{\text{self}}}$ be symmetric, with spectral decomposition $S = U\Lambda U^T$, and let n_+ and n_- be the numbers of strictly positive and strictly negative eigenvalues. Writing $J = \operatorname{diag}(I_{n_+}, -I_{n_-})$ and $\Psi = |\Lambda|^{1/2}U^T$ restricted to the $n_+ + n_-$ non-null directions, we obtain

$$S_{ij} = \langle \Psi_i,\, J\,\Psi_j \rangle, \qquad \operatorname{rank}(S) = n_+ + n_-, \tag{22}$$

that is, an embedding of the n_{self} points into a pseudo-Euclidean space $\mathbb{R}^{(n_+, n_-)}$ whose bilinear form carries signs, and whose dimension is exactly the rank of S (Goldfarb, 1984; Haasdonk, 2005). The function-space counterpart of this construction is a reproducing kernel Krein space rather than a

Hilbert space (Ong et al., 2004; Loosli et al., 2016). Two consequences matter here. First, $\rho^{\text{self}}(K)\,r$ is the dimension of the smallest space in which the kernel can realise the similarities it produces, for *any* symmetric kernel, Mercer or not. Second, we obtain this reading without claiming that our kernels induce an RKHS, which they need not.

Recovering the Mercer case. If K is a valid Mercer kernel then $q = 0$, the signature is trivial, and Eq. (22) reduces to the familiar factorisation $K(X, X) = X'(X')^T$ with $X' = \Phi(X)$ for a feature map Φ into a Hilbert space $\mathcal{H}$. Let $D_{\mathcal{H}} = \dim \mathcal{H}$, which is infinite for the RBF kernel and a combinatorial number for polynomial kernels. Since $\operatorname{rank}(M) = \operatorname{rank}(MM^T)$ and a product cannot exceed the inner dimension of its factors,

$$\mathbb{E}[\operatorname{rank}_\tau(K(X, X))] \leq D_{\mathcal{H}}, \tag{23}$$

so the implicit feature dimension caps the rank from above. Read in the other direction, the measured $\rho^{\text{self}}(K)\,r$ is a certificate from below: a feature space of dimension smaller than the observed rank could not have produced $K(X, X)$ at all. Measuring ρ^{self} therefore probes the effective dimension of the mapping without requiring that mapping to exist in a Hilbert space in the first place.

Which ρ the bound is about. The certificate just derived is the one Sec. 4.1 appeals to, and it is stated for the self-kernel quantity of Def. 4: it is $\rho^{\text{self}}(K)\,r$, not the cross-kernel $\rho(K)\,r$ of Def. 2, that lower-bounds the feature dimension. The two coincide in what they are trying to capture but are measured on different objects, and the signature argument of Eq. (22) needs a symmetric matrix, which $K(A, B)$ with $A \neq B$ is not. For the cross-kernel ρ we therefore make no feature-space claim at all: there ρ is read purely as the rank the kernel induces at the shape the adaptation actually uses, which is the quantity the rest of the paper reports and fits. Readers who want a feature-space reading should take it through ρ^{self}.

Advantage of a Rank-Based Criterion over Mercer Validity for PEFT. While Mercer's theorem provides a theoretical guarantee that a high-dimensional mapping *exists* (which may suggest an easy linear separability in SVMs by Cover's theorem(Cover, 2006)), it does not quantify the *effective dimensionality* of that mapping for a specific data distribution. A valid Mercer kernel could theoretically map to an infinite-dimensional space, yet numerically collapse to a low-rank subspace on the data of interest. The distinction matters, and we state it precisely: a strictly positive definite kernel such as the RBF kernel keeps $K(X, X)$ of full algebraic rank n_{self} for almost every X, so the collapse we report is never an algebraic degeneracy, only a rapid decay of the spectrum that drives all but a few singular values below the tolerance of Eq. (15).

In the context of PEFT, we are less concerned with the strict positive semi-definiteness required in SVMs, and more concerned with the **actual numerical rank** of the generated update matrix. Selecting kernels by that quantity offers a pragmatic advantage:

- **Quantifiable Rank Expansion:** Instead of merely stating that a mapping exists, $\rho^{\text{self}}(K)$ explicitly measures how many linearly independent directions are generated from the random Gaussian basis X. A density of $\rho^{\text{self}}(K)$ means the kernel turns the rank-r input into a rank-$\rho^{\text{self}}(K)\,r$ update.
- **Statistical Stability:** For a rank-stable kernel (Def. 2, tested in Sec. C.6), the variance of this rank across different random seeds is negligible. This implies that while the specific mapping is stochastic (dependent on the initialization of X), the *expected rank* of the resulting update is deterministic and controllable.

Therefore, selecting kernels by induced rank can be viewed as a generalization of Mercer validity tailored to deep learning: it relaxes the PSD constraint, which is unnecessary for non-convex neural network training, in exchange for a direct and measurable handle on the rank expansion ratio.

C.3 ARK Formulation and Kernel Specifications

In this section, we provide the precise mathematical formulations for the Adaptive-Rank Kernels (ARK) used in our experiments.

C.3.1 General Formulation.

Table 5: Examples of common kernel functions expressed in our unified matrix representation. $\|A_i - B_j\|_2^2$ denotes the matrix of pairwise squared Euclidean distances. Here PSD means that every finite Gram matrix satisfies $c^\top Kc \geq 0$, whereas strict positive definiteness requires a strict inequality for every nonzero coefficient vector under the relevant distinct-point assumptions. The qualifications in the final column summarize standard forms and representative parameter regimes from kernel theory and SVM literature, including the power-series criterion for dot-product kernels, Gaussian positive definiteness, and the parameter-sensitive behavior of sigmoid kernels (Schölkopf & Smola, 2002; Hofmann et al., 2008; Burges, 1998; Lin & Lin, 2003; Micchelli, 1986). The rank-based criterion used in this work is independent of Mercer validity.

Kernel Name	$\mathbf{K}_{\text{base}}(\mathbf{A}, \mathbf{B})$	Act(z)	Formula	Conventional qualification
Linear	AB^T	z	$\alpha(\gamma AB^T + \delta) + \beta$	standard $x^\top y$: PSD
Polynomial	AB^T	z^d	$\alpha(\gamma AB^T + \delta)^d + \beta$	$d \in \mathbb{Z}_{\geq 0}$; nonnegative coefficients
Radial Basis Function (RBF)	$-\|A_i - B_j\|_2^2$	$\exp(z)$	$\alpha \cdot \exp(\gamma \cdot -\|A_i - B_j\|_2^2 + \delta) + \beta$	$\gamma > 0$: PSD; strict PD under standard assumptions
Logistic Sigmoid	AB^T	$\sigma(z)$	$\alpha \cdot \sigma(\gamma AB^T + \delta) + \beta$	parameter- and data-dependent
Tanh Sigmoid	AB^T	$\tanh(z)$	$\alpha \cdot \tanh(\gamma AB^T + \delta) + \beta$	$\gamma > 0$, sufficiently negative δ: CPD regime

The update matrix $\Delta W \in \mathbb{R}^{d_{in} \times d_{out}}$ in ARK is computed as:

$$\Delta W = K_\Theta(A, B) \qquad (24)$$

where $A \in \mathbb{R}^{d_{in} \times r}$ and $B \in \mathbb{R}^{d_{out} \times r}$ are low-rank matrices. Two initializations meet the degeneracy condition of App. B.1, and we report both. The **default**, used by every unmarked ARK row, is LoRA's own asymmetric scheme: Kaiming initialization (He et al., 2015) for A, $B = 0$, and the base weights left untouched at $W_0 = W_s$. Since $K(A, 0) = 0$ for the kernels we use, the update is exactly zero at step 0, which keeps the comparison against LoRA on the same footing. The variant marked † instead draws both factors and compensates in the base weights, $W_0 = W_s - K(A_0, B_0)$, following PiSSA (Meng et al., 2024) and LoRA-GA (Wang et al., 2024); we keep it because our kernel is symmetric in its two arguments, so zeroing one factor is not the only natural way to start from the pre-trained state (App. C.2). In both cases α is initialized to LoRA's scaling factor so that training starts from the pre-trained state.

Table 6: Kernel specifications. *Top:* kernels that require an alternative matrix representation. For Piecewise Linear, $A^{(p)}$ denotes the p-th slice of the feature dimension along the rank axis, and the scaling factors are $s_p = \gamma \cdot (-1)^p$. For Mercer-Tanh the nonlinearity is applied to the low-rank factors directly, so the implicit feature mapping Φ is trivially $\tanh$, a dimension-keeping map. *Bottom:* the scalar parameters of Eq. (25) and their roles; below we set $\beta = \delta = 0$ so that only the scale moves, not because the other two are inert.

Kernel Name	Formula	Scalar Parameters
Chi-Square (χ^2)	$\alpha \cdot \exp\left(-\gamma \sum_k \frac{(A_{ik} - B_{jk})^2}{A_{ik} + B_{jk} + \epsilon}\right)$	α, γ
Piecewise Linear	$\sum_{m=1}^{P} s_m \|A_i^{(m)} - B_j^{(m)}\|_k,\ s_m = \gamma \cdot (-1)^m$	γ
Mercer-Tanh	$\alpha\left(\tanh(\gamma A + \delta)\left[\tanh(\gamma B + \delta)\right]^T\right) + \beta$	$\alpha, \beta, \gamma, \delta$

Parameter	Description
γ (pre-activation scale)	Scales the kernel before the activation; the knob we learn (`scale_before`).
δ (pre-activation shift)	Changes the homogeneity, a rank route too (Apps. C.7, C.3.5).
α (post-activation scale)	Scales the output; the spectrum stays proportional, so the rank is untouched.
β (post-activation shift)	Adds a rank-one term, raising the attainable rank by one.

C.3.2 Unified Matrix-Form Kernel Representation.

A kernel function $k(u, v)$ is a similarity function that computes the inner product of two vectors u, v in a potentially high-dimensional feature space, $k(u, v) = \langle \phi(u), \phi(v) \rangle$. In our context, we extend this to matrices by computing the kernel value for every pair of row vectors from $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{n \times r}$. The resulting kernel matrix is denoted by $K(A, B) \in \mathbb{R}^{m \times n}$.

We use a unified representation that captures common kernel functions. This structure composes a base kernel with an activation function, modulated by several key parameters:

$$K_\Theta(A, B) = \alpha \cdot \text{Act}\big(\gamma \cdot K_{\text{base}}(A, B) + \delta\big) + \beta \qquad (25)$$

where $\Theta = \{\alpha, \beta, \gamma, \delta\}$ are scalar parameters, whose roles are summarized in Tab. 6. Based on our finding that γ moves the induced rank monotonically and predictably for kernels such as Swish(Ramachandran et al., 2018) and Tanh, we designate γ as the default learnable parameter, while treating the others as initialization constants. The RBF kernel is deliberately not in that list: its rank response to γ is non-monotonic, rising and then collapsing (Sec. 4.2, App. C.4), so a gradient on γ does not have a well-defined direction. The base kernel, K_{base}, is a simple similarity measure, typically the dot product (AB^T). Tab. 5 provides concrete examples of how this unified form instantiates several well-known kernels.

C.3.3 Alternative Matrix-Form Representations.

While the unified representation in formula 25 covers a broad class of kernels, certain specialized kernels require distinct matrix formulations that do not fit this pattern. For instance, the Chi-Square (χ^2) kernel involves element-wise operations that cannot be reduced to a simple base kernel followed by an activation. Similarly, the Piecewise Linear kernel(Shen et al., 2024) aggregates distances across feature slices. We also consider the Mercer Tanh kernel(Carrington et al., 2014), which applies non-linearity to the factors A and B individually before the dot product, ensuring the resulting matrix is always Positive Semi-Definite by construction. Tab. 6 details these alternative formulations. That kernel is worth pausing on, because it separates the two criteria as cleanly as any example could: it is PSD by construction, yet its feature map is applied element-wise to the factors and therefore keeps their dimension, so the product still passes through an inner dimension of r and $\rho \approx 1$. Mercer validity is thus no guarantee of rank expansion, which is why we select kernels by the rank they induce rather than by membership in the Mercer class.

C.3.4 Reinterpreting Kernel Parameters and Constraints.

In traditional SVM literature, the standard dual formulation is typically analyzed under the requirement that each finite Gram matrix be positive semidefinite (PSD), while strict positive definiteness is a stronger property and is not required for dual convexity (Burges, 1998; Schölkopf & Smola, 2002). The exact qualification is kernel- and domain-dependent: the Gaussian kernel has a standard global PSD regime, whereas sigmoid kernels can be non-PSD or conditionally positive definite for particular data and parameter ranges (Hofmann et al., 2008; Lin & Lin, 2003; Micchelli, 1986). For the fully parameterized forms in Tab. 5, nonnegative scaling and constant offsets are additionally needed whenever PSD preservation is intended. Our work reinterprets these scalar parameters through the lens of the **Rank Scaling Density**. In particular, the pre-activation scale γ serves as a control knob for the effective rank of the generated update matrix. As demonstrated in Sec. C.4, for stable kernels, the relationship between γ and the resulting rank is predictable and adaptive. We therefore do not impose Mercer PSD validity as a prerequisite for the rank-based analysis. What replaces it is a different question: how much rank a kernel induces, and whether that amount responds to the kernel's own parameters. App. C.9 answers it for the candidates in this table. This perspective permits exploration of hyperparameter regimes beyond conventional Mercer qualifications while keeping the PSD, strict-PD, and CPD distinctions explicit.

C.3.5 Formulas and Nature of the Swish Kernel

In this paper, we have identified that the Swish activation function (Ramachandran et al., 2018) can be repurposed as an Adaptive-Rank Kernel, as stated in Sec. 4.1. We are not aware of prior work that treats Swish as a kernel in this sense, so we provide a detailed exposition of its formulation, theoretical properties, and practical implications below.

Formulation The Swish kernel can be defined as formula 26, in the unified matrix-form representation we mentioned in C.3.2.

$$\begin{aligned} K_{\Theta}^{\text{Swish}}(A,B) &= \alpha \cdot \text{Swish}(\gamma AB^T + \delta) + \beta \\ &= \alpha \cdot \big((\gamma AB^T + \delta) \cdot \sigma(\gamma AB^T + \delta)\big) + \beta \end{aligned} \tag{26}$$

Here, $\sigma(z) = \frac{1}{1+e^{-z}}$ and $\odot$ denotes element-wise multiplication. The parameters $\Theta = \{\alpha, \beta, \gamma, \delta\}$ control the scaling and shifting, as detailed in Tab. 6.

Fig. 4 places the Swish kernel among the other kernels this paper uses, grouped by the argument each one consumes. The kernels we adopt are elementwise functions of the pre-activation $z = \gamma a^\top b + \delta$, so they share one axis; RBF and χ^2 instead act on how far apart the two rows are, and their decay toward zero is what makes the kernel matrix, and with it the rank, collapse at large σ (Sec. 4.2).

ReLU is Non-Linear Yet Never an ARK The grouping above also settles a question a reader may raise, namely why we do not simply take the most common non-linearity. ReLU is positively homogeneous, $\mathrm{ReLU}(\gamma Z) = \gamma\,\mathrm{ReLU}(Z)$ for $\gamma > 0$, so the scale factors out of the kernel matrix entirely. All three rank indicators we report are invariant to positive scaling: the numerical rank

uses a threshold relative to $\sigma_{\max}$, the stable rank is the ratio $\|\cdot\|_F^2/\|\cdot\|_2^2$, and the effective rank is a functional of the normalized spectrum. Hence $\rho(K_\gamma^{\mathrm{ReLU}}, \sigma^2)$ is exactly constant in γ, and by Def. 3 ReLU cannot be an ARK no matter how non-linear it is. Together with the polynomial kernel, whose controller is the integer degree (App. C.7), this gives the two ways a non-linearity fails the definition: the controller may be non-differentiable, or it may leave the rank untouched.

Theoretical Properties: Rank Behavior Under Parameter Limits We study how (γ, δ) regulates the *effective* and numerical rank of $K_\Theta^{\mathrm{Swish}}$, thereby clarifying their role as **rank scaling parameters**. Write $S = AB^T$ for the pre-activation similarity matrix, so that $K_\Theta^{\mathrm{Swish}} = \alpha\,\mathrm{Swish}(\gamma S + \delta) + \beta$ and $\mathrm{rank}(S) \le r$ by Eq. (19); the synthesized update is $\Delta W = K_\Theta^{\mathrm{Swish}}(A, B)$, which is what the bounds below constrain. Let $D = \min(d_{in}, d_{out})$ be the structural rank ceiling. The statements below are about the *tolerance rank* rank_τ of Eq. (16), not the algebraic rank: Swish has an infinite Taylor support, so Thm. 5 caps its algebraic rank only at D, and the low-rank behavior that the limits describe is what a fixed tolerance sees once the higher-order terms fall below it. Throughout, we assume a generic initialization of A, B drawn from a continuous distribution (e.g., Kaiming Initialization (He et al., 2015)), ensuring with probability one: (i) no accidental algebraic degeneracies, (ii) sign patterns of S_{ij} are non-pathological.

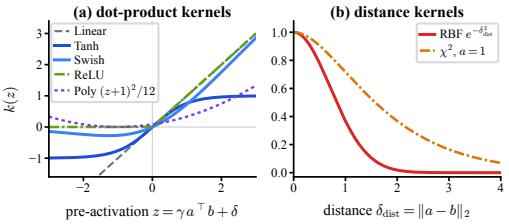


Figure 4: Kernels used in this paper, grouped by their argument. **(a)** Elementwise functions of the pre-activation z: the linear kernel is the identity, Tanh saturates, Swish saturates only on one side, ReLU is positively homogeneous, and the polynomial kernel grows without bound (drawn as $(z+1)^2/12$ to share the box). **(b)** Kernels of the separation between two rows, read on the scalar profile $a{=}1$, $b{=}1{+}\delta_{\mathrm{dist}}$ with $\gamma{=}1$, $\alpha{=}1$, $\delta{=}\beta{=}0$. RBF is a function of $\|a-b\|_2$ alone; the χ^2 kernel of Tab. 6 is not, since its denominator $A_{ik}{+}B_{jk}$ also carries the sum, which is why it is shown at a fixed anchor and why Tab. 6 lists it apart from the unified form. On this profile χ^2 reduces to $\exp\!\big(-\gamma\delta_{\mathrm{dist}}^2/(2{+}\delta_{\mathrm{dist}})\big)$, so it decays roughly like $e^{-\gamma d}$ rather than $e^{-\gamma d^2}$ and reaches zero more slowly than RBF.

Definition 5 (Rank Scaling Regime)**.** *We call (γ, δ) a Rank Scaling Regime if changing either induces a monotonic (or piecewise monotonic) shift in at least one complexity indicator (Numerical Rank, Stable Rank, Effective Rank) in expectation over random draws of (A, B).*

Theorem 1 (Small γ Limit: Low-Rank Preservation)**.** *Fix δ and a tolerance $\tau \in (0, 1)$. There exists $\gamma_0 > 0$ such that for all $0 < \gamma < \gamma_0$ the Swish kernel is a constant matrix perturbed by a scaled low-rank component, and*

$$\mathrm{rank}_\tau(K_\Theta^{Swish}) \le r + 1. \tag{27}$$

In the strict case $\gamma = 0$ the output is a constant matrix, whose algebraic rank is at most 1.

Proof. Consider the Taylor expansion of the element-wise function $f(x) = \mathrm{Swish}(x)$ around $x = \delta$. Let J be the all-ones matrix. For small γ:

$$\mathrm{Swish}(\gamma S + \delta J) \approx \mathrm{Swish}(\delta)J + \gamma\mathrm{Swish}'(\delta)S + O(\gamma^2). \tag{28}$$

Substituting this into the kernel definition:

$$K_\Theta^{\mathrm{Swish}} \approx (\alpha\mathrm{Swish}(\delta) + \beta)J + \alpha\gamma\mathrm{Swish}'(\delta)S. \tag{29}$$

Since $\mathrm{rank}(J) = 1$ and $\mathrm{rank}(S) \le r$, by the subadditivity of rank, $\mathrm{rank}(K_\Theta^{\mathrm{Swish}}) \le 1 + r$. As $\gamma \to 0$, the term involving S vanishes, leaving a rank-1 constant matrix. This implies that for small γ, the kernel preserves the low-rank structure of S. □

Theorem 2 (Large γ Limit: Full-Rank Inflation)**.** *As $\gamma \to +\infty$ (with δ fixed), the non-linearity of the Swish function destroys the low-rank structure: the kernel converges entrywise to the masked form below, whose tolerance rank saturates,*

$$\lim_{\gamma\to\infty} \mathrm{rank}_\tau(K_\Theta^{Swish}) = \mathrm{rank}_\tau(\alpha\,\mathrm{ReLU}(S) + \alpha\delta H(S) + \beta J)\,, \tag{30}$$

which our measurements find at the structural ceiling D (Fig. 6).

Table 7: Limiting behavior of K_Θ^{Swish} under extreme parameter regimes. $S = AB^T$ is the pre-activation similarity matrix, J the all-ones matrix, r the latent dimension and $D = \min(d_{in}, d_{out})$ the structural rank ceiling.

Limit	Approximate Form	Limiting rank_τ	Structural Effect
$\gamma \to 0$	$(\alpha\text{Swish}(\delta) + \beta)J + \alpha\gamma\,\text{Swish}'(\delta)\,S$	$\leq r + 1$	Low-Rank Preservation
$\gamma \to \infty$	$(\gamma S + \delta J) \odot H(S)$	$\to D$	Full-Rank Inflation
$\delta \to +\infty$	$\alpha\gamma S + (\alpha\delta + \beta)J$	$\leq r + 1$	Linearization
$\delta \to -\infty$	βJ	≤ 1	Collapse (Info Loss)

Proof. As $\gamma \to +\infty$, the Swish function behaves asymptotically like a ReLU function: $\text{Swish}(x) \approx \max(0, x)$. Specifically, for the matrix elements:

$$\text{Swish}(\gamma S_{ij} + \delta) \approx \begin{cases} \gamma S_{ij} + \delta & \text{if } S_{ij} > 0 \\ 0 & \text{if } S_{ij} < 0 \end{cases} \tag{31}$$

Let $H(S)$ be the Heaviside step matrix where $H_{ij} = 1$ if $S_{ij} > 0$ and 0 otherwise. The kernel can be approximated as:

$$K_\Theta^{\text{Swish}} \approx \alpha\left((\gamma S + \delta J) \odot H(S)\right) + \beta J. \tag{32}$$

The Hadamard product with the sign mask $H(S)$ effectively "turns off" entries where $S_{ij} < 0$. For a generic S, this operation removes linear dependencies between rows and columns, leaving a matrix of full rank D almost surely. □

Theorem 3 ($\delta \to +\infty$ Limit: Linearization)**.** *Fix γ and a tolerance $\tau \in (0, 1)$. There exists δ_0 such that for all $\delta > \delta_0$ the Swish kernel is in a linear regime and preserves the low-rank structure:*

$$\text{rank}_\tau(K_\Theta^{\textit{Swish}}) \leq r + 1. \tag{33}$$

Proof. For large positive inputs, $\text{Swish}(z) \approx z$. Thus, as $\delta \to +\infty$:

$$\text{Swish}(\gamma S + \delta J) \approx \gamma S + \delta J. \tag{34}$$

The kernel becomes:

$$K_\Theta^{\text{Swish}} \approx \alpha(\gamma S + \delta J) + \beta J = \alpha\gamma S + (\alpha\delta + \beta)J. \tag{35}$$

As in Theorem 1, this is a linear combination of S, whose rank is at most r, and the rank-1 matrix J. Thus, the rank is bounded by $r + 1$. □

Theorem 4 ($\delta \to -\infty$ Limit: Collapse)**.** *Fix γ and a tolerance $\tau \in (0, 1)$. There exists δ_1 such that for all $\delta < \delta_1$ the kernel collapses to a constant matrix, resulting in information loss:*

$$\text{rank}_\tau(K_\Theta^{\textit{Swish}}) \leq 1. \tag{36}$$

Proof. For large negative inputs, $\text{Swish}(z) \approx 0$. Thus, as $\delta \to -\infty$:

$$\text{Swish}(\gamma S + \delta J) \approx 0. \tag{37}$$

Consequently, $K_\Theta^{\text{Swish}} \approx \beta J$. The resulting matrix is a constant matrix with rank at most 1 (or 0 if $\beta = 0$). □

Summary of Limiting Regimes Tab. 7 summarizes the structural impact of the parameters on the kernel's rank.

C.4 Functional Form of the Rank Scaling Law: Candidate Families and Model Selection

This subsection supplies what Sec. 4.2 defers to the appendix: the candidate function families for a kernel's rank response, and the rule that selects among them. The object being characterised is the Rank Scaling Density ρ of Def. 2 as a function of the scale that drives it, so the statement supported here is that this response has a closed functional form, monotone for the kernels we adopt and non-monotonic for the ones we reject.

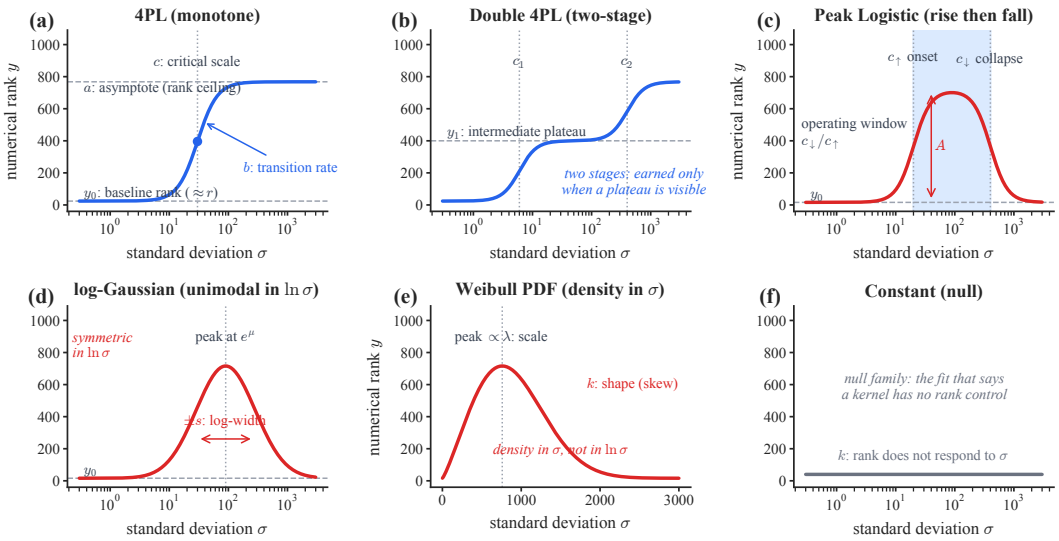


Figure 5: Parameter semantics of the candidate families, drawn from the same model functions the fitting code uses, with schematic parameter values chosen for legibility (these panels illustrate the parameters rather than report fits, so no R^2 is attached to any of them). Color follows the convention used throughout: blue for the monotone families selected on the rank-stable kernels, red for the unimodal families a collapsing response requires, gray for the null family. (a) 4PL: y_0 is the small-σ baseline ($\approx r$, where the kernel acts linearly), a the reachable ceiling ($\approx \min(d_{in}, d_{out})$), c the value of σ at which the transition reaches its midpoint, and b how abrupt it is. (b) Double 4PL adds a second transition (c_1, c_2) around an intermediate plateau y_1. (c) Peak Logistic: $c_\uparrow$ and $c_\downarrow$ bound the operating window, whose width is their ratio, and A is the rank gained inside it. (d) log-Gaussian: peak position e^μ and log-width s. (e) Weibull PDF is a density in σ itself, with scale λ fixing the peak and shape k the skew. (f) Constant: a single level k, that is, no response to σ.

C.4.1 Axis and Measured Quantity

The controlled variable is the one in Def. 2, the standard deviation σ of the Gaussian factors. As shown in Sec. 4.2, $\rho(K_\gamma, 1) = \rho(K_1, \sigma^2)$ with $\sigma^2 = \gamma$, so moving the kernel scale and moving the factor variance trace the same curve; we sweep the scale over six decades and report the response against σ. The fitted quantity y is the measured numerical rank of $K(A, B)$, averaged over the factor draws at each grid point, from which $\rho = y/r$ follows by Def. 2.

Each configuration is fitted on its own, that is, one fit per kernel, latent dimension r and layer size $D = \min(d_{in}, d_{out})$. This is where the dependence of ρ on r and on the layer dimensions, suppressed in the notation of Def. 2, is examined: the selected family is stable across the two latent dimensions we sweep at $D = 768$, with Tanh taking the single-transition form at both $r = 8$ ($R^2 = 0.9946$) and $r = 16$ ($R^2 = 0.9981$), and the second stage of the Swish response becoming resolvable, and therefore selected, only at $r = 16$.

C.4.2 Candidate Families and the Parsimony Rule

All families below are fitted to the same measurements, and we then apply a parsimony rule: among the families whose coefficient of determination is within 0.01 of the best achievable R^2, we retain the one with the fewest parameters. This favours the simpler family whenever the extra flexibility buys a negligible gain in fit: for Tanh, the seven-parameter Double-4PL improves R^2 by less than 0.002 over the four-parameter 4PL on a single-transition curve, which we read as overfitting rather than as evidence of a second stage. Since two of the candidates are parameterised in $\ln \sigma$, all families are scored on the same $\sigma > 0$ grid (one of the 768 grid points is discarded). Fig. 5 shows what each parameter means on the curve it belongs to.

1. **Four-Parameter Logistic (4PL) Model:** Selected for monotonic scaling behaviors (e.g., Tanh, Swish).

$$y = y_0 + \frac{a - y_0}{1 + (\sigma/c)^b} \tag{38}$$

Here y is the numerical rank and σ the input standard deviation. The parameters carry the following reading:

- a: The **asymptote**, representing the maximum reachable expected rank at high input scale. This is related to $\min(d_{in}, d_{out})$.
- y_0: The **baseline rank**, i.e., the numerical rank when σ is very small. This is generally nonzero: for the Tanh kernel, $\tanh(z) \approx z$ as $z \to 0$, so the operation is effectively linear and, unless the inputs are exactly zero everywhere, the attainable rank approaches r.
- c: The **critical scale**. This is the value of σ at which the rank transition reaches its midpoint, and it locates the operating point a practitioner has to reach before the rank starts to move.
- b: The **transition rate** (Hill slope), indicating how abruptly the rank increases around the critical scale c.

2. **Peak-Logistic Model:** Selected for non-monotonic behaviors (e.g., RBF), where excessive scale drives the rank back down. It is the product of a rising and a decaying logistic gate, so it stays within the same logistic vocabulary as the family above:

$$y = y_0 + A \cdot \frac{1}{1 + (c_\uparrow/\sigma)^{b_\uparrow}} \cdot \frac{1}{1 + (\sigma/c_\downarrow)^{b_\downarrow}} \tag{39}$$

   Here $c_\uparrow$ and $c_\downarrow$ locate the onset of expansion and the onset of collapse, and their ratio measures the width of the usable operating window; $b_\uparrow, b_\downarrow$ are the corresponding transition rates. Non-monotonicity is structural rather than inferred: the fitted curve necessarily rises and then falls. For the RBF kernel this family attains $R^2 = 0.9916$ with six parameters, whereas the seven-parameter Double-4PL reaches only 0.8538, so the parsimony rule prefers it on both criteria at once.

3. **Log-Gaussian Model:** A four-parameter alternative for the same unimodal regime, $y = y_0 + A\exp\big(-(\ln\sigma - \mu)^2/2s^2\big)$, whose parameters read directly as peak position e^μ and log-width s. It is fitted alongside the others but is not selected for any kernel reported here ($R^2 = 0.9528$ on RBF).

4. **Weibull Probability Density Function (PDF):** A unimodal density in σ itself rather than in $\ln\sigma$,

$$y = \text{amp} \cdot \frac{k}{\lambda}\left(\frac{\sigma}{\lambda}\right)^{k-1} \exp\left(-\left(\frac{\sigma}{\lambda}\right)^{k}\right) \tag{40}$$

   retained as a candidate for comparability with prior practice. Because the sweep spans six decades, a density in linear σ places its mass on one decade and cannot follow a narrow late peak sitting on a long low plateau; it is therefore never selected here (e.g., $R^2 = 0.6911$ on RBF).

5. **Constant Model:** A baseline model ($y = k$) for a response that does not move with σ. This family carries a selection decision rather than absorbing leftovers: the criterion of Sec. 4.2 admits a kernel as an Adaptive-Rank Kernel only if its ρ responds to the input standard deviation, so a kernel whose fit selects the constant family is one whose rank cannot be steered along this axis, and whose pre-activation scale is therefore unavailable as a rank controller at fixed budget. Thm. 7 says in advance which kernels land here: a positively homogeneous map factors the scale out of the kernel matrix, and every rank indicator we report is invariant to positive scaling. The linear kernel, that is LoRA itself, the piecewise-linear kernel of KLoRA (App. C.8) and ReLU are all of this type, and the constant fit is where that algebra becomes visible in the measurements. Selecting this family is a statement about the swept axis, not about the parameterization as a whole: another member of Θ may still move the rank, as the degree does for the polynomial family (App. C.7) and as δ does through the homogeneity itself.

Fig. 5 also carries the two-stage family (Double 4PL), since that is the family the parsimony rule compares against for Tanh and the one it selects for Swish at $r = 16$.

C.5 Empirical Attained-Rank Ranges at Fixed Budget

The analysis above characterizes the ordered rank response along the input scale σ. We next quantify the complementary notion of rank expressiveness: the numerical-rank range attained while the factor shapes, and hence the trainable-parameter budget, remain fixed. Let $D = \min(d_{in}, d_{out})$. For each (K, D, r) configuration, we sweep a nonnegative factor-scale grid and omit the degenerate zero-scale point. Within each of seven independent factor draws, we record the minimum and maximum numerical rank over the sampled scales; Fig. 6 reports the median of each endpoint, normalized by D. The ordered fits and endpoint ranges therefore characterize the response trajectory and its attained span, respectively.

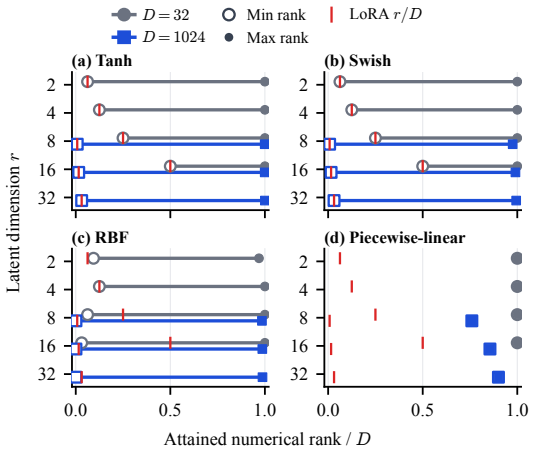


Figure 6: Attained numerical-rank ranges at fixed factor budget. Each segment joins the median per-draw minimum and maximum numerical rank over the sampled scale grid, normalized by D; open and filled markers are the two endpoints. Circles denote D=32, squares D=1024; red ticks mark the LoRA reference r/D.

For Tanh and Swish, the lower endpoint tracks the LoRA reference r/D, while the upper endpoint approaches the structural ceiling at both matrix sizes. Thus the same factor budget supports outcomes near the linear baseline and near full numerical rank through the sampled scale settings. The piecewise-linear endpoints coincide at high rank, confirming rank expansion without the non-constant scale response required by an ARK. RBF also spans a broad endpoint range, consistent with the fitted non-monotonic response above.

C.6 Statistical Validation of Rank Scaling Law Stability

Motivation. Because $\Delta W = K_\theta(A, B)$ uses random A, B, the rank we report is a random variable, and the main text's notion of rank stability (Def. 2) asks that it concentrate. This subsection makes that predicate operational and then tests it on the three computed metrics: Numerical Rank, Stable Rank, Effective Rank.

The predicate being tested. Fix a metric $\mathbf{R}$, a kernel K_θ, and a grid Γ of m pre-activation scales γ. We call K_θ (ε, δ)**-rank-stable** on Γ if

$$\sup_{\gamma \in \Gamma} \Pr_{A,B} \left[\left| \mathbf{R}(K_\gamma(A, B)) - \mathrm{med}_\gamma \right| > \varepsilon \, \mathrm{med}_\gamma \right] \;\leq\; \delta, \tag{41}$$

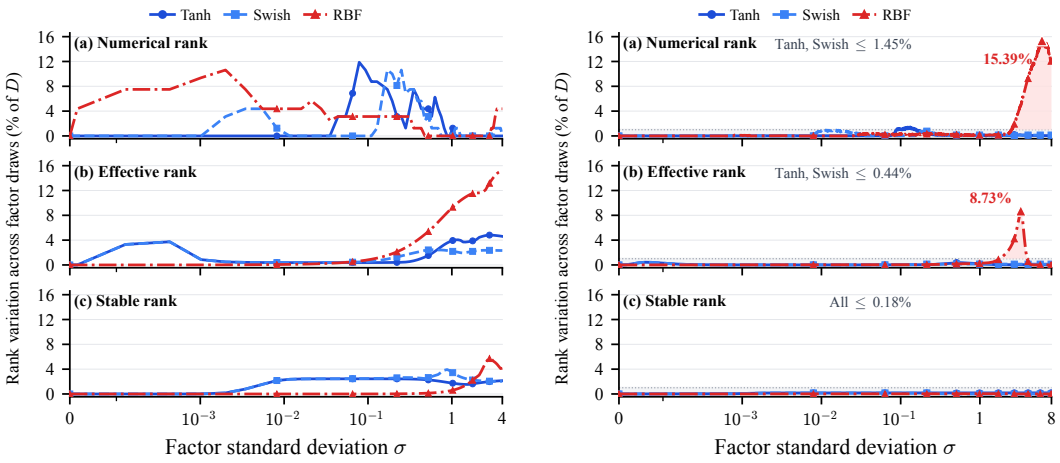
where med_γ is the median over factor draws at that γ. This is a statement with two knobs, a tolerated relative deviation ε and a tolerated failure probability δ: setting γ fixes the rank to within ε except on a δ-fraction of factor draws.

Scale-resolved concentration diagnostic. For each kernel, scale, and rank metric, we draw seven independent pairs of factor matrices (A, B) and compute the 10th–90th percentile rank spread across these draws. We divide the spread by $D = \min(d_{in}, d_{out})$, so the plotted percentage directly measures how much the rank changes when the random factor draw changes. Values near zero indicate reproducible ranks across draws. Tracking this variation along σ distinguishes a localized transition peak from persistent variation over a broad scale interval.

Table 8: Maximum rank variation across factor draws over the sampled factor-scale grid, measured as the normalized 10th–90th percentile spread (% of D).

	$d_{in} = d_{out} = 32, r = 4$			$d_{in} = d_{out} = 1024, r = 32$		
Kernel	Num	Eff	SR	Num	Eff	SR
Tanh	11.88	4.81	2.44	1.45	0.43	0.17
Swish	10.63	3.73	3.94	0.98	0.43	0.17
RBF	10.63	15.18	5.71	15.39	8.73	0.08

Empirical findings. In the small regime ($d_{in} = d_{out} = 32$, $r = 4$; Fig. 7a), the median Numerical Rank spread is zero for Tanh and Swish, with their maxima confined to their respective transition



(a) Small regime ($d_{in} = d_{out} = 32$, $r = 4$). (b) Large regime ($d_{in} = d_{out} = 1024$, $r = 32$).

Figure 7: Rank variation across factor draws. At each sampled factor standard deviation σ, each curve is the 10th–90th percentile rank spread across seven independent draws of the factor pair (A, B), divided by $D = \min(d_{in}, d_{out})$; values near zero mean that changing the random factor draw leaves the rank nearly unchanged. The gray band marks variation within 1%, and the pale red regions mark scales where RBF exceeds it. The horizontal axis uses a symmetric-log scale that retains $\sigma = 0$.

regions. RBF has a nonzero median spread and remains dispersed over a broader low-scale interval. Effective Rank gives the clearest separation at high scales: its maximum normalized spread is 4.81% for Tanh, 3.73% for Swish, and 15.18% for RBF. Stable Rank stays within 5.71% for all three kernels.

The separation strengthens in the large regime ($d_{in} = d_{out} = 1024$, $r = 32$; Fig. 7b). Tanh and Swish remain below 1.45% on Numerical Rank and below 0.44% on Effective Rank across the sampled scale grid. RBF reaches 15.39% and 8.73% on the same metrics, respectively, while Stable Rank variation stays below 0.18% for all three kernels. Thus the monotone Tanh and Swish responses are tightly concentrated in the large regime, while the non-monotonic RBF response retains broad cross-draw variation on the primary indicators.

Implication. Tanh and Swish combine predictable mean rank responses with concentrated trajectories across factor draws, particularly at the larger matrix size. RBF shows substantially broader dispersion in Numerical and Effective Rank, consistent with its rise-and-collapse response across the scale sweep.

C.7 Analysis of the Polynomial Kernel as an ARK Candidate

The polynomial kernel is a classic example in kernel methods, defined as $K(\mathbf{x}, \mathbf{y}) = (\gamma \mathbf{x}^T \mathbf{y} + \delta)^p$, where $p \in \mathbb{Z}_{\geq 0}$ is the polynomial degree. It is a well-known valid Mercer kernel, meaning it corresponds to an implicit mapping $\phi(\cdot)$ into a higher-dimensional feature space $\mathcal{H}$, such that $K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle_{\mathcal{H}}$. The dimension of this target feature space for an input vector $\mathbf{x} \in \mathbb{R}^r$ is given by the binomial coefficient:

$$\dim(\mathcal{H}) = \binom{r+p}{p} \tag{42}$$

This property makes the polynomial kernel an intriguing candidate for our Adaptive-Rank Kernel (ARK). Under our rank-scaling criterion (Sec. 4.1 of the main paper), the inhomogeneous polynomial kernel $K(A, B) = (\gamma AB^T + \delta)^p$ with $\delta \neq 0$ has a Rank Scaling Density of $\binom{r+p}{p}/r > 1$, since every degree up to p survives and $\sum_{k=0}^{p} \binom{r+k-1}{k} = \binom{r+p}{p}$ by Thm. 5. For the homogeneous case $\delta = 0$ only the top degree survives and the cap falls to $\binom{r+p-1}{p}$, which is the value our measurements reproduce. Either way the rung count is finite, so by Thm. 6 the cap does not move with γ: the numerical rank of $K(A, B)$ sits at the cap whenever the layer is large enough to accommodate it, and stays there.

Given that the feature space dimension, and consequently the numerical rank, is directly controlled by the degree p, the polynomial kernel appears to perfectly align with the motivation of ARK: by

scaling the parameter p, we can scale the expected rank of the update without changing the number of trainable parameters in A and B. This suggests that the polynomial family realizes the Iso-Budget Rank Scaling Law in principle.

However, it is not an ARK in the sense of Def. 3: for this family the parameter that moves ρ is the degree itself, $\Theta = \{p\}$, and p is a discrete integer hyperparameter. The Rank Scaling Density is therefore defined only at integer points and is neither continuous nor differentiable in Θ, leaving gradient descent nothing to turn. Making p learnable does not remove the obstacle: relaxing it to a real value, or routing it through a straight-through estimator (Bengio et al., 2013), destabilises training and drives the loss to `NaN`.

Another significant challenge arises when combining the polynomial kernel with the standard LoRA initialization, where the matrix B is initialized to zeros. For a polynomial kernel with degree $p > 1$ and a common setting of $\delta = 0$, the gradient with respect to the parameters in B vanishes at the start of training. To see this, consider the (i, j)-th element of the update matrix, $\Delta W_{ij} = (\gamma(AB^T)_{ij})^p$. The gradient of the loss $\mathcal{L}$ with respect to an element B_{jk} involves the term $\frac{\partial \Delta W_{ij}}{\partial B_{jk}}$, which is proportional to $(\gamma(AB^T)_{ij})^{p-1}$. Since B is initialized to zero, $(AB^T)_{ij}$ is zero. For any $p > 1$, the exponent $p - 1$ is positive, causing the gradient term to become zero. This "dead neuron" problem prevents the model from learning any updates for these layers.

The ablation of Sec. 5.3 therefore reports this family at its most favourable: a third-power polynomial added to a linear kernel, with the combination factor as the learnable parameter, listed as "Polynomial" in Tab. 4. Adding a linear term does not change the diagnosis: the union of two finite rung sets is finite, so Thm. 6 still applies and the cap remains independent of γ. Even given that advantage the polynomial family stays below the linear kernel.

Given these intractable optimization challenges, we conclude that despite its theoretical appeal as a Mercer kernel with predictable dimension scaling, the polynomial kernel is not a suitable candidate for an ARK. In contrast, kernels like Tanh and Swish offer a continuous, learnable parameter γ (or the pre-activation shift δ) that smoothly controls the numerical rank, which is what an ARK requires.

C.8 Proof that Piecewise Linear Function Cannot Serve as Valid Mercer Kernel

A function $k(x, z)$ is a valid Mercer kernel if and only if for any finite set of points $\{x_1, \ldots, x_N\}$, the corresponding Gram matrix K, where $K_{ij} = k(x_i, x_j)$, is positive semi-definite (PSD). A symmetric matrix K is PSD if for any non-zero vector $c \in \mathbb{R}^N$, the quadratic form $c^T K c \geq 0$.

The Piecewise-Linear kernel used in KLoRA (Shen et al., 2024) splits the r-dimensional inputs into P consecutive slices $x = (x^{(1)}, \ldots, x^{(P)})$ and aggregates the slice-wise Euclidean distances with alternating signs (Tab. 6):

$$k_{\mathrm{PWL}}(x, z) \;=\; \sum_{m=1}^{P} s_m \big\| x^{(m)} - z^{(m)} \big\|, \qquad s_m = \gamma\,(-1)^m. \tag{43}$$

We first show that its building block, the negative Euclidean distance, is not a Mercer kernel (Lemma), and then that the failure is inherited by k_{PWL} for *every* slice count $P \geq 1$ and every $\gamma \neq 0$ (Proposition), so it is not an artifact of a particular configuration such as the P=2 setting used by KLoRA.

Lemma (the building block is not PSD). Consider three points in a one-dimensional space ($\mathbb{R}^1$):

$$x_1 = 0, \quad x_2 = 1, \quad x_3 = 2$$

For $k(x, z) = -\|x - z\|$, the Gram matrix is

$$K = \begin{pmatrix} 0 & -1 & -2 \\ -1 & 0 & -1 \\ -2 & -1 & 0 \end{pmatrix},$$

and taking $c = [1, 1, 1]^T$ gives $c^T K c = -8 < 0$ (its spectrum is $\{-2.73, 0.73, 2\}$). Hence $-\|x - z\|$ is not PSD and not a valid Mercer kernel.

Proposition (Eq. (43) is not PSD for any P and $\gamma \neq 0$). Positive semi-definiteness is preserved under restriction: if k is PSD, then so is the Gram matrix of *any* finite subset of inputs. It therefore suffices to exhibit one such subset. Assume without loss of generality $\gamma > 0$ (the case $\gamma < 0$ follows by exchanging the roles of the odd and even slices), and choose $N = 3$ inputs that agree on all slices except the first,

$$x_i = (u_i,\, v,\, \ldots,\, v), \qquad u_1 = 0,\; u_2 = 1,\; u_3 = 2,$$

with v arbitrary. Every term with $m \geq 2$ contributes $s_m \|v - v\| = 0$, so Eq. (43) restricted to these inputs reduces to $-\gamma|u_i - u_j|$, *i.e.* γ times the matrix K of the Lemma. With the same c we obtain $c^T K c = -8\gamma < 0$. Consequently k_{PWL} is not PSD, for every $P \geq 1$ and every $\gamma \neq 0$. □

Remark (the failure is structural, not a matter of normalization). The negative Euclidean distance is a kernel *of negative type*: it is conditionally PSD, *i.e.* $c^T K c \geq 0$ holds for all c with $\sum_i c_i = 0$, which is why distance-based kernels can be rescued by the standard centering that maps them to a PSD kernel (Schölkopf & Smola, 2002). This escape route is unavailable here. For $P \geq 2$ the construction assigns a *positive* sign to the even slices, and picking two inputs that differ only in such a slice gives, with $c = [1, -1]^T$ (so $\sum_i c_i = 0$), the value $c^T K c = -2\gamma \|x^{(2)} - z^{(2)}\| < 0$: the alternating construction is not even conditionally PSD, so no shift or centering makes it a Mercer kernel.

Implication for KLoRA: Is it a Valid Kernel? The kernel that KLoRA reports as its best-performing configuration is therefore outside Mercer's framework, so the implicit feature mapping its rank-expansion argument appeals to is not guaranteed to exist, and the dimension-expanding property cannot be grounded in standard kernel theory.

Although the Piecewise-Linear kernel fails the Mercer condition, which leaves its mechanism unexplained under standard kernel theory, the rank it induces is straightforward to characterise. Measured at a fixed tolerance it does raise the numerical rank well above the parameter budget r, which is the property KLoRA relies on.

What it cannot do is move that rank, which is what separates it from the **Adaptive-Rank Kernel (ARK)** proposed in this work. Its scale enters as a single multiplicative factor, so the kernel is positively homogeneous in γ and Thm. 7 applies: the rank is not merely insensitive to γ, it is identical for every γ, in every realization and at every tolerance. The attainable rank is then a fixed value set by the parameter budget r, which is exactly the coupling that the Iso-Budget Rank Scaling Law requires a kernel to break, and which our Tanh and Swish kernels do break. Consequently, our framework also clarifies why KLoRA works: its rank expansion follows from the same mechanism, and what our Tanh and Swish kernels add is a learnable scale that decouples rank from the parameter budget.

C.9 The Rank Ladder: Two Necessary Conditions for a Rank Controller

This section answers a question that the selection criterion of Sec. 4.1 raises but does not settle: given an element-wise non-linearity, when can its scale γ move the rank of the synthesized update at all? We give two conditions, each of which is sufficient to make the attainable rank *independent* of γ, and therefore each of which excludes the non-linearity from serving as a rank controller. Both are stated for the exact (algebraic) rank; the scope of that choice is stated at the end.

Throughout, $A \in \mathbb{R}^{d_{in} \times r}$ and $B \in \mathbb{R}^{d_{out} \times r}$ have rows drawn *i.i.d.* from $\mathcal{N}(0, \sigma^2 I)$, we write $M = \gamma A B^T$ and $D = \min(d_{in}, d_{out})$, and Act acts element-wise so that $K_\gamma(A, B) = \mathrm{Act}(\gamma A B^T)$. We write $M^{\odot k}$ for the k-fold element-wise (Hadamard) power. Two notions of rank are kept apart: the **exact rank** $\mathrm{rank}(\cdot)$, the number of non-zero singular values, and the tolerance rank $\mathrm{rank}_\tau(\cdot)$ of Eq. (16). Statements below concern the exact rank unless the tolerance is named.

Theorem 5 (Rank ladder of an element-wise analytic map)**.** *Let* Act *be real analytic at the origin with* $\mathrm{Act}(z) = \sum_{k \geq 0} c_k z^k$ *on a neighbourhood containing the entries of* M*, and let* $S = \{k \geq 1 : c_k \neq 0\}$ *be its support. Then*

$$\mathrm{rank}\big(\mathrm{Act}(M)\big) \;\leq\; \min\Big(D,\; \mathbf{1}_{\{c_0 \neq 0\}} + \textstyle\sum_{k \in S} \binom{r+k-1}{k}\Big). \tag{44}$$

Proof. Element-wise evaluation turns the series into a sum of Hadamard powers, $\mathrm{Act}(M) = c_0 \mathbf{1}\mathbf{1}^T + \sum_{k \in S} c_k M^{\odot k}$, where $\mathbf{1}$ is the all-ones vector. For the k-th power, write $M^{\odot k} = \gamma^k\, \Phi_k(A) \Phi_k(B)^T$,

where Φ_k maps each row of a factor to its k-th symmetric tensor power. The image of Φ_k lies in a space of dimension $\binom{r+k-1}{k}$, hence $\operatorname{rank}(M^{\odot k}) \le \binom{r+k-1}{k}$. The rank of a sum is at most the sum of the ranks, and no matrix of this shape can exceed D. □

We call the summands the *rungs* of the ladder: each surviving Taylor coefficient contributes one rung, and rung k is $\binom{r+k-1}{k}$ high. The scale γ does not create or remove rungs, it only reweights them by γ^k.

Theorem 6 (A finite ladder caps the rank independently of γ)**.** *If S is finite, then the bound of Thm. 5 is a constant D_S that does not depend on γ. Consequently ρ cannot be moved past D_S/r by any choice of γ, and the family $\{K_\gamma\}$ is not an Adaptive-Rank Kernel.*

Proof. Immediate from Thm. 5: with S finite the right-hand side is a finite sum of binomial coefficients in r and k alone. □

This is the mechanism behind the polynomial family. For a homogeneous cubic, $S = \{3\}$ gives $D_S = \binom{r+2}{3}$, which is 4 at $r = 2$ and 20 at $r = 4$; for the third Chebyshev polynomial, $S = \{1, 3\}$ gives $D_S = r + \binom{r+2}{3}$, which is 6 and 24 respectively. Both predictions are met exactly by measurement, with zero spread across random draws. The polynomial kernel of Sec. C.7 is the inhomogeneous case: when $\delta \neq 0$ every degree up to p survives and $\sum_{k=0}^{p} \binom{r+k-1}{k} = \binom{r+p}{p}$, which recovers the feature-space dimension quoted there; when $\delta = 0$ only the top degree survives and the cap falls to $\binom{r+p-1}{p}$. The distinction matters numerically and we state it explicitly to avoid quoting the inhomogeneous count for a homogeneous kernel.

Theorem 7 (Positive homogeneity makes the rank invariant to γ)**.** *Suppose* $\mathrm{Act}(cz) = c^p \mathrm{Act}(z)$ *for all $c > 0$ and some $p > 0$. Then for every realization of (A, B), every $\gamma > 0$ and every $\tau \in [0, 1)$,*

$$\operatorname{rank}_\tau\big(K_\gamma(A, B)\big) = \operatorname{rank}_\tau\big(K_1(A, B)\big), \tag{45}$$

so ρ is constant in γ and $\{K_\gamma\}$ is not an Adaptive-Rank Kernel.

Proof. Element-wise homogeneity gives $K_\gamma(A, B) = \gamma^p K_1(A, B)$. Singular values are positively homogeneous of degree one, so $\sigma_i(K_\gamma) = \gamma^p \sigma_i(K_1)$ for every i. The test $\sigma_i > \tau\sigma_1$ is invariant under multiplying both sides by $\gamma^p > 0$, hence the counted set, and with it the exact rank obtained at $\tau = 0$, is unchanged. □

Thm. 7 covers the linear kernel ($p = 1$), homogeneous monomials z^p, ReLU ($p = 1$ on $\gamma > 0$), and the piecewise-linear kernel of Sec. C.8, whose scale enters as a single multiplicative factor. Measurement agrees and is stronger than the qualitative statement made there: the rank is not merely insensitive to γ, it is numerically identical across the sweep under three different tolerance conventions.

The two conditions are independent. ReLU's Taylor support, equivalently its Hermite support under the Gaussian input measure, is infinite, so ReLU is excluded by Thm. 7 alone. A homogeneous cubic, by contrast, is excluded by both. Neither condition subsumes the other, and a candidate must fail both in order to be admitted.

Table 9: Kernels excluded by each of the two conditions, and the rank response $\rho(\gamma)$ each would give. Every verdict is read off Thms. 6 and 7, with no free parameters.

Kernel	Excluded by	$\rho(\gamma)$
Linear	Thm. 7, $p = 1$	flat at 1
Monomial z^p	Thms. 6, 7	flat at $\binom{r+p-1}{p}/r$
Chebyshev T_p	Thm. 6	flat at D_S/r
Polynomial, learnable p	Thm. 6	flat between degrees
ReLU	Thm. 7, $p = 1$	flat, height set by τ
Piecewise linear	Thm. 7	flat
Tanh, Swish	neither	moves with γ

Scope. Both theorems constrain the exact rank, so we use them only to *exclude* candidates, and we report the tolerance alongside every measured rank. When the tolerance rank tracks the exact rank is a question about the spectrum near the threshold rather than about Act, and is treated in Sec. C.6.

Summary of the resulting exclusions. The verdicts of Tab. 9 are read off the two theorems, with no free parameters.

The last row of Tab. 9 is the point of the exercise: Tanh and Swish survive both conditions, which is what licenses treating their scale as a controller rather than as a normalization constant. Surviving these conditions is necessary and not sufficient, and the additional requirement that the induced rank concentrate across draws is what Sec. C.6 tests.

Two readings of the table are worth keeping apart. The verdicts of the middle column are exact: they follow from the theorems alone, so a kernel listed there cannot be turned into a rank controller by any choice of γ. The right column records what is lost when such a kernel is used anyway: a flat $\rho(\gamma)$ means the scale moves the spectrum without moving the rank, and the two surviving kernels are the only rows for which ρ is a function of γ rather than a constant.

C.10 Algebraic Bounds on the Attainable Rank

In classical kernel theory, Mercer kernels enjoy closure properties (the sum or the product of two Mercer kernels is again a Mercer kernel), which provide a calculus for building complex feature spaces. Our criterion asks about rank rather than about membership in a class, so the corresponding calculus bounds the rank that a combination induces instead of asserting closure. Having bounded the rank of a single element-wise map in Sec. C.9, we now bound ρ under the three operations we use to build composite kernels.

We treat scalar multiplication, addition, and the Hadamard product. For brevity we fix $\sigma^2 = 1$ and let $D = \min(d_{in}, d_{out})$ be the structural rank ceiling. Each composite below is again an element-wise map of the same form, so Thm. 5 applies to it without change. The linear-algebra inequalities we invoke are statements about the *algebraic* rank, whereas ρ is defined on the numerical rank of Eq. (15); since the numerical rank never exceeds the algebraic one, we use them as upper bounds only, in the same spirit as the scope note of Sec. C.9.

Theorem 8 (Scalar Multiplication Invariance)**.** *Let K be a matrix kernel and $c \in \mathbb{R} \setminus \{0\}$ a non-zero scalar. The scaled kernel $K_c(A, B) = c \cdot K(A, B)$ has the same Rank Scaling Density:*

$$\rho(cK) = \rho(K) \tag{46}$$

Proof. In linear algebra, the rank of a matrix is invariant under multiplication by any non-zero scalar, meaning $\mathrm{rank}(cM) = \mathrm{rank}(M)$ for $c \neq 0$. Consequently, for any random realization of A and B, the resulting matrix rank remains identical: $\mathbf{R}_{cK} = \mathbf{R}_K$. Taking the expectation and dividing by r directly yields $\rho(cK) = \rho(K)$. This theoretically validates our empirical decision in Sec. 4.1 to discard the post-activation scale α when searching for rank controllers. □

Theorem 9 (Addition and Subadditivity Bounds)**.** *Let K_1 and K_2 be matrix kernels. The Rank Scaling Density of their sum $K_{add}(A, B) = K_1(A, B) + K_2(A, B)$ satisfies*

$$|\rho(K_1) - \rho(K_2)| \leq \rho(K_1 + K_2) \leq \min\left(\rho(K_1) + \rho(K_2), \frac{D}{r}\right) \tag{47}$$

Proof. For any two matrices M_1 and M_2 of the same dimensions, the rank of their sum is bounded by the subadditivity property and the triangle inequality of rank (Horn & Johnson, 2012):

$$|\mathrm{rank}(M_1) - \mathrm{rank}(M_2)| \leq \mathrm{rank}(M_1 + M_2) \leq \mathrm{rank}(M_1) + \mathrm{rank}(M_2) \tag{48}$$

Substituting $M_1 = K_1(A, B)$ and $M_2 = K_2(A, B)$, taking the mathematical expectation over the random matrices A, B, and dividing the entire inequality by r yields:

$$\frac{1}{r}\mathbb{E}\big[|\mathbf{R}_{K_1} - \mathbf{R}_{K_2}|\big] \leq \rho(K_1 + K_2) \leq \rho(K_1) + \rho(K_2) \tag{49}$$

Applying Jensen's inequality to the convex absolute value function provides the lower bound: $|\mathbb{E}[\mathbf{R}_{K_1}] - \mathbb{E}[\mathbf{R}_{K_2}]|/r \leq \mathbb{E}\big[|\mathbf{R}_{K_1} - \mathbf{R}_{K_2}|\big]/r$. Additionally, since the rank cannot exceed the structural ceiling D, the upper bound is capped at D/r. □

Theorem 10 (Hadamard Product and Multiplicative Bounds)**.** *Let K_1 and K_2 be matrix kernels. The expected rank of their Hadamard (element-wise) product $K_{had}(A, B) = K_1(A, B) \odot K_2(A, B)$ is upper-bounded by:*

$$\mathbb{E}[\mathbf{R}_{K_1 \odot K_2}] \leq \min\left(\mathbb{E}[\mathbf{R}_{K_1} \cdot \mathbf{R}_{K_2}],\ D\right) \tag{50}$$

Proof. The rank of a Hadamard product is upper-bounded by the product of the individual ranks (Styan, 1973):

$$\operatorname{rank}(M_1 \odot M_2) \leq \operatorname{rank}(M_1) \cdot \operatorname{rank}(M_2) \tag{51}$$

Taking the expectation yields $\mathbb{E}[\mathbf{R}_{K_{had}}] \leq \mathbb{E}[\mathbf{R}_{K_1}\mathbf{R}_{K_2}]$. Notably, unlike addition, the expected product $\mathbb{E}[\mathbf{R}_{K_1}\mathbf{R}_{K_2}]$ cannot be naively decoupled into $\rho(K_1) \cdot \rho(K_2) \cdot r^2$ unless the rank variables of K_1 and K_2 are strictly independent, which is generally untrue given they share the same input matrices A and B. Nevertheless, this bound confirms that element-wise multiplication possesses a highly aggressive, combinatorial potential for rank expansion, albeit strictly bottlenecked by the structural rank ceiling D. □

Discussion. Thm. 9 bounds an additive composite from both sides, so adding a linear residual to a non-linear kernel can neither cancel the rank the non-linear part induces nor inflate it beyond the sum of the two. Thm. 10 accounts for Swish, which is the Hadamard product of the linear kernel with a logistic factor. Neither bound is a statement about downstream accuracy, which the ablations address separately.

D Experimental Details and Additional Results

D.1 Details on Datasets and Evaluation Protocols

To comprehensively evaluate our proposed methods, we conduct experiments across Natural Language Processing (NLP) and Large Language Model (LLM) domains.

The reference implementation of the proposed adapter, the test suite that checks the properties the construction claims, the measurement files and scripts behind the efficiency and kernel-curve figures, and a training entry point that reruns the ARK-versus-LoRA comparison at a fixed parameter budget accompany this submission as the Code and Data Supplement.

Natural Language Processing (GLUE). For standard NLP tasks, we evaluate our methods on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019), which consists of 9 diverse sequence classification tasks (CoLA, SST-2, MRPC, STS-B, QQP, MNLI, QNLI, RTE, and WNLI). Following Devlin et al. (2019) and Ben Zaken et al. (2022), we exclude WNLI and report the remaining eight tasks: the GLUE authors note that the construction of this dataset is problematic, and no submitted system outperforms the 65.1 majority-class baseline, so the task carries no usable signal for comparing adaptation methods. The same eight-task protocol is adopted by the LoRA-family baselines we compare against (Hu et al., 2022; Zhang et al., 2023b; Valipour et al., 2023; Liu et al., 2024b). We implement our training and evaluation pipelines based on the standard `run_glue.py` script provided by the Hugging Face `transformers` library. The encoders are the Hugging Face checkpoints `bert-base-cased` (referred to as BERT-Base-Cased throughout, including Tab. 1) and, for the backbone-transfer check, `roberta-base` (RoBERTa-Base, Tab. 12); no uncased variant is used anywhere in the paper. Following standard practice in the PEFT literature, since the GLUE test set labels are not publicly available and require limited server submissions, all reported GLUE results are evaluated on the standard validation (dev) sets.

Large Language Models (Mathematical Reasoning). For LLM experiments, we use two widely adopted mathematical reasoning datasets: MetaMathQA (Yu et al., 2024) and GSM8K (Cobbe et al., 2021). The MetaMathQA dataset is used as the primary training set, while GSM8K is employed for validation and testing.

To ensure data quality and control input length during training, we filter out samples from MetaMathQA whose concatenated query and response exceed 1300 characters, retaining approximately 93.8% of the original data. For GSM8K, no filtering is necessary as all samples are sufficiently short. For validation during training, we randomly select 50 samples from the GSM8K training split. This

practice is adopted because evaluation is performed multiple times during training, making a smaller validation set preferable for efficiency. The official GSM8K test split is strictly used as the test set and is evaluated only once after the conclusion of training.

Crucially, following the standard evaluation protocols established by Hugging Face PEFT method comparisons, we strictly utilize the *Base* foundation models rather than their chat-tuned *Instruct* or *Thinking* counterparts. This ensures that the reasoning capabilities are genuinely acquired through the PEFT optimization process rather than relying on prior instruction tuning. To elicit step-by-step reasoning from the base model, we format all inputs using a consistent zero-shot chain-of-thought template:

```
Question: {query} Think step by step.\nAnswer:
```

All datasets are preprocessed and tokenized using this consistent template to ensure a strictly fair comparison across all baselines and model variants.

Generation and Scoring Protocol (GSM8k). Because generative accuracy depends on how an answer is decoded and matched, we state the full protocol. Decoding is **greedy** (`do_sample=False`, single beam, no temperature/top-p/top-k and no custom stopping criteria), capped at 300 newly generated tokens with a total sequence budget of 800 tokens; prompts are left-padded and generation runs under `inference_mode()` with the model in `eval()` mode. A prediction is scored by anchoring on the first delimiter among ``` ``The answer is: '' ```, ``` ``The answer is '' ```, ``` ``The final answer is: '' ```, ``` ``The final answer is '' ``` and ``` ``#### '' ```; the remaining text up to the first newline is stripped of trailing punctuation and parsed as a decimal (simple fractions `a/b` are accepted). Gold answers are parsed identically, and an item counts as correct only when the two parsed values are numerically equal; a generation in which no delimiter appears or the parse fails counts as incorrect. Accuracy is therefore a strict exact-match rate and not a "last number in the string" heuristic. Test accuracy is computed once, after the final optimization step, over the complete official GSM8K `main` test split (1,319 problems); the 50-sample validation subset is used only for the periodic checks during training.

Optimization Protocol (GSM8k). All runs use `bfloat16` with mixed precision and SDPA attention, a maximum sequence length of 768, AdamW (weight decay 0.1, gradient-norm clipping 1.0) and a cosine schedule with a 10% warmup, for 5,000 optimization steps without gradient accumulation. Following the memory limits of a single GPU, the per-step batch size is 8 for Qwen2.5-3B and 1 for Qwen2.5-7B; every method in Tab. 2 is trained under the identical setting for a given backbone. Each configuration is run with seeds $\{0, 1, 2\}$ and we report the median, as elsewhere in the paper. Learning rates are selected per method by the grid search described above. Adapters are inserted in the attention W_q and W_v projections, matching the GLUE protocol.

D.2 Details on Compared Methods

Tab. 11 organizes representative PEFT methods, including every baseline reported in the main text, by tuning space and re-parameterization capability. To ensure a fair and rigorous comparison, we adhered to the following principles:

- **Fidelity of Implementation:** For all baseline methods, we used their official implementations provided by the HuggingFace PEFT library (Mangrulkar et al., 2022) or other highly-cited public repositories. This ensures that we are comparing against faithful and well-optimized versions of these methods. Method names follow the PEFT library, which is the stable reference when a preprint renames itself across revisions. Our own method, ARK, was implemented within the OpenDelta framework (Hu et al., 2023).
- **Comprehensive Hyperparameter Tuning:** We performed a systematic hyperparameter search for most methods, including the baselines, to find their optimal configuration for each task. We adopted recommended hyperparameter configurations from official documentation where available and expanded upon them to ensure each method was given a fair opportunity to perform at its best. The detailed search spaces and final chosen parameters are provided in Sec. D.3.
- **Scope of the kernelized-LoRA baseline:** KLoRA (Shen et al., 2024) denotes the kernelized-LoRA component of SNELL. We implement it in the same OpenDelta framework as ARK,

Table 11: A taxonomy of representative PEFT methods categorized by their tuning space and re-parameterization capability. Parameter-space methods are inherently mergeable (feature-parameter equivalent in App. B.3), while we further highlight those whose attainable rank is adaptive. KLoRA is the kernelized-LoRA component of SNELL; the non-mergeability of bottleneck adapters follows Luo et al. (2023). References, in table order: LoRA (Hu et al., 2022), DoRA (Liu et al., 2024a), PiSSA (Meng et al., 2024), LoHa (Yeh et al., 2024; Hyeon-Woo et al., 2022), LoKr (Yeh et al., 2024; Edalati et al., 2025), KLoRA (Shen et al., 2024), VeRA (Kopiczko et al., 2024), VB-LoRA (Li et al., 2024), Bone (Kang & Yin, 2025), RandLoRA (Albert et al., 2025), OFT (Qiu et al., 2023), BOFT (Liu et al., 2024b), OSF (Nayak et al., 2026), FourierFT (Gao et al., 2024), WaveFT (Bilican et al., 2025), GraLoRA (Jung et al., 2025), DeLoRA (Bini et al., 2025), AdaLoRA (Zhang et al., 2023b), DyLoRA (Valipour et al., 2023), AutoLoRA (Zhang et al., 2024b), $(IA)^3$ (Liu et al., 2022), SSF (Lian et al., 2022), RepAdapter (Luo et al., 2023), Adapter (Houlsby et al., 2019), Prompt-Tuning (Lester et al., 2021), P-Tuning (Liu et al., 2024c), Prefix-Tuning (Li & Liang, 2021), LLaMA-Adapter (Zhang et al., 2024a).

Category	Method	Core Idea	Key Mechanism / Property
Baseline	Full Fine-Tuning	Updates all parameters.	Performance and cost upper bound.
Parameter-Space Adaptation	LoRA	Low-rank factorization of the update.	$\Delta W = \alpha\gamma AB^T$.
	DoRA	Splits W into magnitude and direction.	Trainable magnitude, LoRA on direction.
	PiSSA	Adapter starts at the top singular pairs.	Frozen weight is the SVD residual.
	LoHa	Hadamard product of two low-rank pairs.	$\Delta W = (AB^T) \odot (CD^T)$, orig. FedPara.
	LoKr	Kronecker product with a low-rank block.	$\Delta W = C \otimes (AB^T)$, orig. KronA.
	KLoRA	Non-linear kernel on the LoRA factors.	The kernel raises the rank of ΔW.
	VeRA	One frozen random pair, shared by layers.	Learns per-layer scaling vectors.
	VB-LoRA	Updates drawn from a global vector bank.	"Divide-and-share" paradigm.
	Bone	Updates from replicated shared shards.	Matrix shard sharing.
	RandLoRA	Combines fixed random low-rank bases.	Full-rank update, low parameter count.
	OFT	Left-multiplies W by a rotation.	$W = RW_0$, keeps hyperspherical energy.
	BOFT	Butterfly factorization of OFT's R.	Sparse butterfly orthogonal factors.
	OSF	Constrained full fine-tuning of W.	Gradients projected off the top subspace.
	FourierFT	Learns the update in the frequency domain.	Sparse Fourier coefficients.
	WaveFT	Learns a sparse update in the wavelet domain.	$\Delta W = \mathrm{IDWT}(C)$, C sparse.
	GraLoRA	Partitions W into sub-blocks.	Per-block adapters, less entanglement.
	DeLoRA	Decouples angle from adaptation strength.	Normalized, rescaled low-rank factors.
Parameter-Space Adaptation (Adaptive)	AdaLoRA	Allocates budget across weight matrices.	Prunes singular values of a $P\Lambda Q^T$ form.
	DyLoRA	Trains a range of ranks jointly.	Representation sorting, search-free.
	AutoLoRA	Finds a rank per layer.	Meta-learned, thresholded selectors.
	ARK (Ours)	**Realizes the Iso-Budget Rank Scaling Law.**	**Learnable rank controller γ at fixed budget.**
Feature-Space Adaptation (w/ Equivalence)	$(IA)^3$	Rescales activations by learned vectors.	Mergeable for single-task use.
	SSF	Scales and shifts deep features.	Affine on activations, mergeable.
	RepAdapter	Adapters built for re-parameterization.	Zero-cost inference once merged.
Feature-Space Adaptation (w/o Equivalence)	Adapter	Inserts bottleneck feed-forward layers.	Non-linearity blocks merging into W.
	Prompt-Tuning	Learns a soft prompt on the input.	Changes inputs, not weights.
	P-Tuning	Inserts trainable input embeddings.	A prompt encoder (MLP or LSTM) emits them.
	Prefix-Tuning	Prepends prefix activations per layer.	Changes attention activations.
	LLaMA-Adapter	Prompts with zero-init gating.	Stable initialization by construction.

keep the piecewise-linear kernel with the slice count $P{=}2$ adopted in the original work, match its trainable-parameter count to LoRA, and search its learning rate over the same grid as every other method. The sparse-tuning mechanism that SNELL combines with this kernel selects *which entries* of the update are trained rather than the rank of the update, and is therefore orthogonal to the question studied here.

Two further routes to a higher rank per parameter appear in the taxonomy. One shares the trainable factors across layers, modules, or blocks of a single matrix, as in VeRA (Kopiczko et al., 2024), VB-LoRA (Li et al., 2024) and Bone (Kang & Yin, 2025); the other moves the update into a transform domain and learns a sparse set of coefficients there, as in FourierFT (Gao et al., 2024) and WaveFT (Bilican et al., 2025). Both change how much rank a given budget buys, yet in each case the attainable rank follows from the parameterization and the budget rather than from a quantity the task can move, which is the distinction Def. 1 draws. On the feature side the taxonomy further separates methods that learn continuous prompts directly from those that generate them through a small encoder, as P-Tuning (Liu et al., 2024c) does with an MLP or LSTM.

Table 10: Hyperparameter search space and final values for the GLUE benchmark. Learning rates were tuned individually for each dataset.

Method	Hyperparameter	Search Space	Final Value
All PEFT Methods	Learning Rate	{2e-5, 2e-4, 2e-3, 2e-2}	Dataset-specific
LoRA	`lora_alpha` (α)	{8, 16, 32, 64, 128}	64
ARK-Tanh (Ours)	`lora_alpha` (α)	{8, 16, 32, 64, 128}	16
	γ Initialization	{0.5, 0.67, 1.0}	0.5

D.3 Implementation and Hyperparameter Details

As stated previously in Sec. 5.1, we conducted a rigorous hyperparameter search to ensure a strictly fair comparison across all baselines. Tab. 10 details the hyperparameter search space and the selected values used for the main results on the GLUE benchmark.

Software and Hardware Environment. All experiments run on Ubuntu 22.04.4 LTS (Linux kernel 5.15) with NVIDIA L20 GPUs (one GPU per run) and CUDA 12.8. The software stack is Python 3.11, PyTorch 2.10, HuggingFace `transformers` 4.53, `datasets` 4.5, NumPy 2.4, and a local fork of `peft` 0.19 in which ARK and all reproduced baselines are implemented as adapter layers, so that every method shares one training loop (`run_glue.py` for GLUE, a matched script for GSM8k) and one evaluation path.

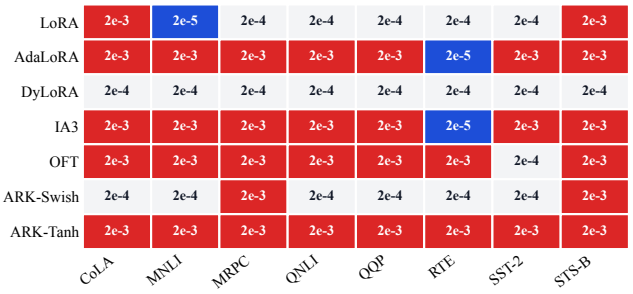


Figure 8: **Optimal learning rate per method and GLUE task.** Each cell reports the rate selected from {2e-5, 2e-4, 2e-3, 2e-2}; blue, gray, and red denote the low, middle, and high selected levels. Values come from the same BERT-Base-Cased runs as Tab. 10.

It is important to note that due to the diverse data characteristics and scale variations across tasks, the optimal learning rate naturally fluctuates. Therefore, we performed extensive grid searches to determine the task-specific optimal learning rates for all PEFT methods and their ablations. For the GLUE benchmark, the search space was $\{2 \times 10^{-5}, 2 \times 10^{-4}, 2 \times 10^{-3}, 2 \times 10^{-2}\}$. **The selected optimal learning rates across key methods and GLUE tasks are visually mapped in Fig. 8.** For the generative mathematical reasoning experiments (LLM on GSM8k), the learning rate search space was bounded within $\{1 \times 10^{-5}, 1 \times 10^{-4}, 9 \times 10^{-4}, 1 \times 10^{-3}\}$.

Crucially, to eliminate discrepancies caused by differing codebases or training frameworks, all baseline results reported in this paper are derived from our own local reproductions, rather than being transcribed from prior literature. For every method and task combination, we evaluated the model using its independently searched optimal learning rate and reported the median performance across at least three independent random seeds. For structural hyperparameters, preliminary experiments established robust default values across all datasets: we set `lora_alpha` to 64 for standard LoRA and 16 for ARK (selected from $\{8, 16, 32, 64, 128\}$), with the pre-activation scale γ strictly initialized to 0.5 (selected from $\{0.5, 2/3, 1.0\}$). For AdaLoRA we additionally searched `lora_alpha` over $\{8, 16, 64\}$ and report its best setting, since this baseline prunes an initially over-provisioned rank and is therefore more sensitive to the effective update scale α/r than the fixed-rank methods; the § row of Tab. 1 uses the resulting best value.

D.4 Backbone Transfer: GLUE on RoBERTa-Base

Tab. 12 reports the per-task numbers behind the backbone-transfer check summarized in Sec. 5.2. The protocol is identical to the BERT-Base-Cased setting of Tab. 1 (eight tasks, WNLI omitted, same per-method rank configuration and learning-rate grid, median over seeds); only the pre-trained encoder differs. We compare against the rank-control family, which shares the parameter budget of Tab. 1.

On this backbone AdaLoRA is provisioned above the budget shared by the other rank-control methods (see the RPC column) yet lands 0.23 below LoRA; as in Tab. 1, its pruning-based adaptivity does not pay for itself at a comparable provisioned cost, and overtakes LoRA only once the provisioned budget is doubled (Tab. 1, §). We also repeated the analysis of Sec. 5.2 on this table: the Friedman omnibus rejects equal mean positions ($\chi^2_{(4)} = 14.04$, $p = 0.0072$), ARK-Swish holds the best mean position, and the Holm-corrected post-hoc separates it from DyLoRA.

Fig. 9 adds the grid behind those numbers: on this backbone every method was run at all four rates of one fixed grid, so the panels record how much of the grid each method can be trained at. Both ARK kernels stay within a few points of their own task best across the grid, while the dynamic-rank and pruning baselines lose more than five points on part of it.

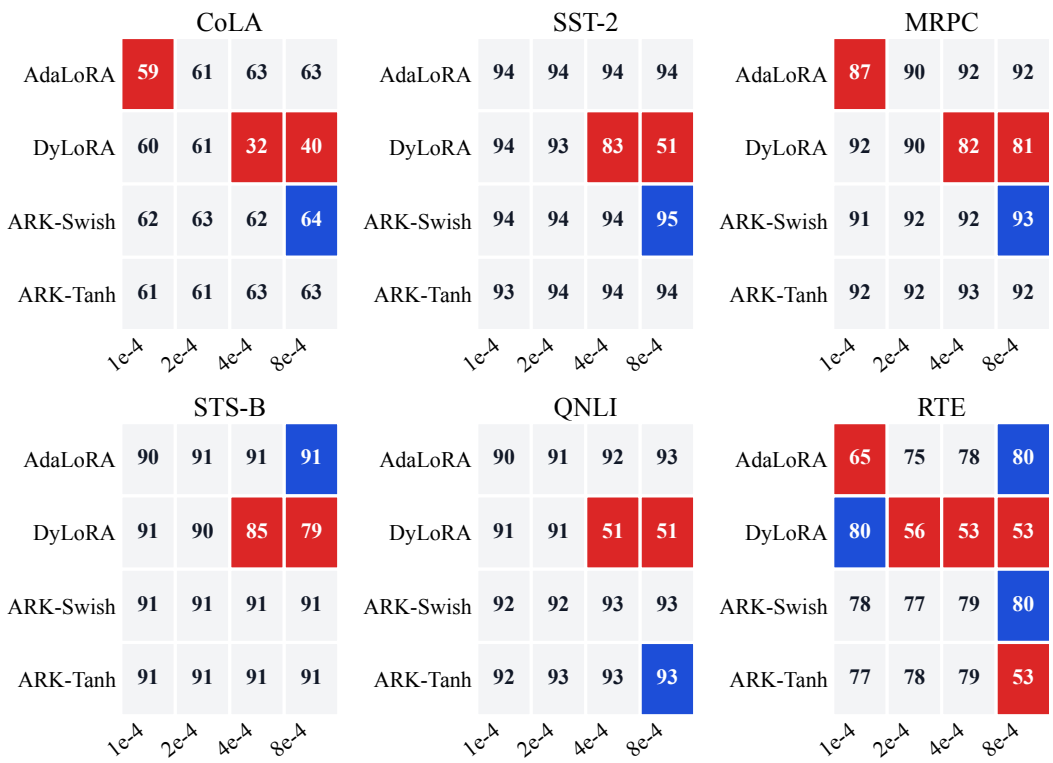


Figure 9: **Learning-rate robustness on RoBERTa-Base.** Median dev score for each method at four learning rates, one panel per task, with the value printed in every cell. Blue marks the best cell of a task and red a cell more than five points below it. This sweep is separate from the selected rates of Fig. 8, which are BERT-Base-Cased: here the grid is $\{1, 2, 4, 8\} \times 10^{-4}$ and every method is run at all four rates, so the panels show how much of the grid each method can be trained at rather than which rate was chosen.

Table 12: Backbone transfer: performance on the GLUE benchmark using RoBERTa-Base. We report standard metrics for each task (Matthews correlation for CoLA, Spearman correlation for STS-B, and accuracy for others), taking the median over seeds. **Bold** numbers indicate the best results among the compared methods. WNLI is omitted following Tab. 1.

Method (Year)	MRPC	STS-B	RTE	CoLA	MNLI	QNLI	QQP	SST-2	Avg.	RPC
LoRA'21	92.09	91.12	79.42	62.34	87.03	93.21	**87.05**	94.72	85.87	7.12e-3
AdaLoRA'22	91.23	91.08	79.42	62.33	86.87	**93.46**	86.15	94.61	85.64	8.30e-3
DyLoRA'23	92.12	90.73	77.98	61.36	86.58	92.29	85.20	94.50	85.10	7.12e-3
ARK-Swish	**92.55**	**91.15**	**80.14**	63.16	86.84	93.06	86.62	**94.84**	**86.05**	7.12e-3
ARK-Tanh	**92.55**	91.10	77.62	**64.58**	**87.11**	93.03	86.98	**94.84**	85.98	7.12e-3

D.5 More Ablation Studies

The ablations of Sec. 5.3 compare kernel families and structural extensions at a single parameter budget, and report one number per configuration: the score reached at the end of training. This section adds the measurements that a single number cannot carry. The subsection below follows the numerical rank of ΔW across training on a dense logging grid, which turns the post-training rank of Fig. 3 into a trajectory and separates two things that the main table conflates, namely the rank a kernel can reach and the path by which it gets there. All runs here share the budget of the main tables (r=32, adapters on W_q and W_v, BERT-Base-Cased), so any difference between curves is attributable to the kernel and its initialization rather than to the parameter count.

D.6 Rank Trajectories under a Dense Evaluation Grid

The main text reports which numerical ranks one budget can reside at (Fig. 3). Here we record the whole trajectory, on a logging grid of 5 to 20 optimizer steps, which gives tens to thousands of measurement points per run. All runs share one budget (r=32, adapters on W_q and W_v, 24 layer-

module pairs per run, BERT-Base-Cased), so any difference between curves is a property of the kernel and its initialization, not of the parameter count.

The attained rank tracks what the task asks for. Fig. 10 follows the Tanh kernel, the configuration behind the main GLUE row of Tab. 1, on all eight tasks of the benchmark. With the standard asymmetric init ($B{=}0$) the update starts at rank 0 and never has to leave the rank $r{=}32$ at which its inner dimension would cap a linear kernel: on the four small tasks it resides just above that level (median 32 on RTE and STS-B, 33 on MRPC and CoLA). On the data-rich tasks the same kernel resides higher, at a level several times that one: median 48 on SST-2 (67k pairs, maximum 144 over modules), 52 on QNLI (105k, max 110), 100 on QQP (364k, max 256) and 66 on MNLI (393k, max 279). The level is reached late there, at 61 to 75 percent of training on those four tasks, against the first few percent on the small ones. A linear kernel under the same budget is pinned at exactly 32 everywhere. What the kernel provides is therefore rank head-room that the optimizer draws on in proportion to the task, within the head-room the Rank Scaling Law of Sec. C.4 quantifies and the range Fig. 3 reports.

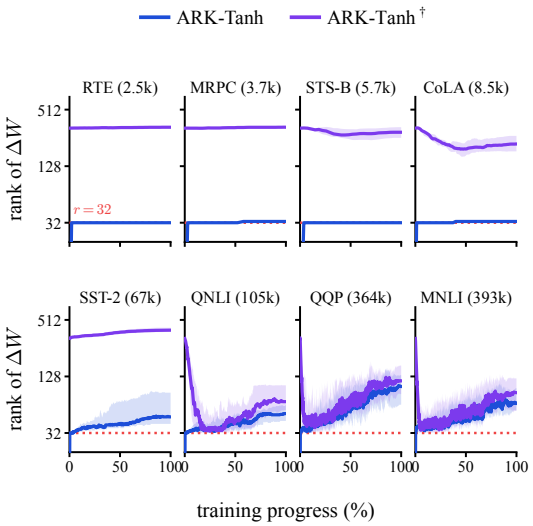


Figure 10: **Numerical rank of ΔW over training on all eight GLUE tasks (BERT-Base, $r{=}32$).** Lines are medians over the 24 layer-module pairs of a run, bands the interquartile range, the vertical axis $\log_2$, and the dotted line marks $r{=}32$, the rank at which the inner dimension caps a linear (LoRA) kernel for the entire run. Panels are ordered by training-set size. With the standard asymmetric init the Tanh kernel resides at that level on the four small tasks and several times higher on the data-rich ones; the symmetric residual variant starts high and either holds its level or moves down to meet it.

The symmetric residual variant ARK-Tanh† enters from the opposite side: it begins at rank $\approx$325 by construction and then moves to where the task wants it, holding its level on MRPC, RTE and SST-2 (final medians 332, 332, 398) and descending on CoLA, QNLI, QQP and MNLI (to 220, 69, 115, 86). Rank is thus adjusted in both directions under a fixed budget, and the two initializations meet at comparable levels on the tasks that support the highest ranks.

Kernel and initialization set the shape of the curve. Fig. 11 contrasts four kernel and initialization combinations on the three ablation tasks. Four behaviors coexist.

Residing at r. With the Tanh kernel and asymmetric init the rank reaches 32 within the first 5 to 20 steps and stays at 32 to 33 on these three tasks. Under a pre-activation scale $\gamma \approx 0.5$ the kernel operates in its near-linear regime, and Fig. 10 shows the same configuration leaving that regime once the task provides more data.

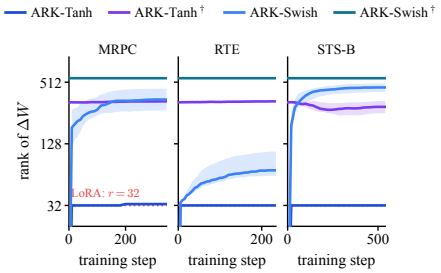


Figure 11: **Four coexisting rank behaviors at one budget (BERT-Base, $r{=}32$).** Medians over 24 layer-module pairs with interquartile bands, $\log_2$ axis, dotted line at $r{=}32$. ARK-Tanh resides near r on these three tasks and ARK-Swish rises over the first quarter to half of training, while the symmetric residual variants start high: ARK-Swish† holds a plateau at $\approx$560 and ARK-Tanh† can move downward (STS-B).

A level reached late. With the Swish kernel the rank rises monotonically over a substantial part of training: the median reaches 90 percent of its final value only after 41 percent of the steps on MRPC ($0 \to 346$), 51 percent on RTE ($0 \to 70$) and 26 percent on STS-B ($0 \to 455$).

Relaxation from a high start. ARK-Tanh† begins at rank $\approx$325 by construction. On MRPC and RTE it stays there, while on STS-B it descends to

$\approx$275 around step 240 before recovering to $\approx$294, that is the rank is reduced during training rather than grown.

A high plateau. ARK-Swish† starts at $\approx$560, the highest of the four, and holds that value on all three tasks while remaining competitive (F1 89.0 on MRPC, accuracy 66.4 on RTE, Pearson 87.8 on STS-B). A high-rank update is therefore not something the optimizer has to reach gradually: it can also be the starting point.

Two points are worth making explicit. First, every behavior above occurs at an identical parameter count, so what differs is the kernel, not the budget; the r=32 line in both figures is where a linear kernel would remain for the entire run. Second, the ordering of these behaviors follows the analytic RSL response of each kernel (Sec. C.4): the rank a module reaches is a property of the kernel the budget is spent through.

D.7 What the Kernel Costs, and Why

Tab. 3 reports that our method trains in $1.01\times$ the time and $1.08\times$ the memory of LoRA, and this subsection derives both before measuring them. Almost none of that cost is the kernel. Two implementations of the same LoRA update scale along different axes, and they do so in opposite directions in the forward and the backward pass, so most of what separates them cancels over a tr aining step; what is left is the elementwise activation itself, which costs a few tenths of a percent of one projection. Time therefore lands inside run-to-run variance, and the only term that stays visible at model scale is the one tensor the derivative of the activation has to retain.

Throughout this subsection an adapted projection maps d_{in} inputs to d_{out} outputs, the input it receives in one optimizer step is $X \in \mathbb{R}^{N\times d_{in}}$ over N tokens, and r is the latent dimension of Eq. (3); we write d only for a square projection, meaning $d_{in} = d_{out} = d$. We first separate the kernel from the implementation it runs inside, then write both cost laws down, then check them against time and against memory in turn, and finally compose them back to the two reported ratios.

Isolating the kernel from the implementation. The update LoRA defines is bilinear, and the original work fixes no order in which to apply it: $X(AB^T)$ and $(XA)B^T$ produce the same matrix, so either order is a faithful implementation of it. Frameworks exploit that freedom, choosing among forward and backward computation graphs by estimated cost and gaining between ten and twenty eight percent for free (Cherniuk et al., 2025); the consequence that matters here is that the factored order never has to form the $d_{in} \times d_{out}$ matrix at all. The elementwise Act of Eq. (3) does not commute with that reordering, since every entry of $\mathrm{Act}(\gamma AB^T)$ needs the corresponding entry of AB^T, so our kernel requires the built order. The two scalars of Eq. (3) split the same way: for a linear update both α and γ

	PEFT LoRA	**ARK-Linear**	**ARK-Tanh**
ΔX	$\alpha\gamma\,(XA)B^T$	$X\big(\alpha\gamma AB^T\big)$	$X\big(\alpha\tanh(\gamma AB^T)\big)$
forms ΔW	no	yes	yes
fwd FLOPs	$2Nr(d_{in}{+}d_{out})$	$d_{in}d_{out}(2r{+}1)$	$d_{in}d_{out}(2r{+}2)$
bwd FLOPs	$4Nr(d_{in}{+}d_{out})$	$2d_{in}d_{out}(N{+}2r)$	$d_{in}d_{out}(2N{+}4r{+}1)$
mem.	$2Nd_{out}$	$2d_{in}d_{out}$	$2d_{in}d_{out}$

Table 13: **The two multiplication orders are both LoRA; which one is cheaper depends on the batch, and the nonlinearity is not what decides it.** $X \in \mathbb{R}^{N\times d_{in}}$ is the layer input and ΔX the increment each configuration adds to its output, with α the constant scaling factor and γ the pre-activation scale of Eq. (3). Every row is counted per adapted projection against the same frozen projection $XW^{\top}$ carrying no adapter, charging a multiply-add as two operations and an elementwise pass as one per entry. A LoRA update is bilinear, so $X(AB^T)$ and $(XA)B^T$ are algebraically the same update and either order is a faithful implementation of it; the reference libraries pick the factored order and never build ΔW Cherniuk et al. (2025), while an elementwise activation needs each entry of AB^T in place and so requires the other order. We implement LoRA in the order our kernel requires, which leaves the kernel as the only difference between the reference row and ours. Neither order dominates. In the forward pass the factored one costs $2Nr(d_{in}{+}d_{out})$ and grows with the token count while the built one costs $d_{in}d_{out}(2r{+}1)$ and depends on the layer shape alone, so they cross at $N^{\star} = \frac{2r+1}{2r}\,\frac{d_{in}d_{out}}{d_{in}+d_{out}}$, within two percent of $d/2$ for a square projection at r=32; by the largest batch measured here the built order is an order of magnitude cheaper. The backward pass narrows that gap, since a built update accumulates a gradient of the shape of ΔW before contracting back into the two factors, so there both orders grow with N. The nonlinearity is one further pass over the same matrix, which raises the arithmetic by $(2r{+}2)/(2r{+}1)$, 1.5% at r=32, plus the one activation its derivative consumes; ARK-Tanh and ARK-Linear differ by exactly that pass and that tensor. Memory splits with the roles exchanged: the factored order materialises its own $N \times d_{out}$ output and then a sum, two activations that grow with the batch, while the built order folds the update into W and holds a pair of $d_{in} \times d_{out}$ buffers that does not.

sit outside every product and may be placed wherever is cheapest, while under $\tanh$ only α keeps that freedom.

Comparing our method against the factored implementation alone would charge the kernel with the choice of order as well. We therefore run LoRA in the built order too, as ARK-Linear, which computes exactly the LoRA update through our code path and applies no nonlinearity: the step from the factored implementation to ARK-Linear is the order, and the step from ARK-Linear to ARK-Tanh is the kernel. Tab. 3 normalizes to ARK-Linear, which holds the data pipeline, the optimizer, the injection sites and the code path fixed and leaves the kernel as the only difference between the reference row and ours; a ratio taken across two implementations would carry their engineering differences as well, and those are of the same order as the effect being reported. Neither order is privileged on cost, and both are measured here side by side (Tabs. 14 and 15): at the smallest batch the two adapter terms are indistinguishable, both below the dispatch floor, while past the crossover derived below the built order is an order of magnitude cheaper in the forward pass. Tab. 13 sets the three updates and their cost laws side by side.

N	adapter op alone, forward (μs) PEFT	Linear	Tanh	ratio $\frac{\text{Tanh}}{\text{PEFT}}$	$\frac{\text{Lin.}}{\text{PEFT}}$	$\frac{\text{Tanh}}{\text{Lin.}}$
d_{in}=d_{out}=768, the width of BERT-base and RoBERTa-base						
128	36.3	19.3	37.3	1.03	0.53	1.93
4096	40.5	19.3	25.1	0.62	0.48	1.30
16384	42.8	19.2	25.2	0.59	0.45	1.31
d_{in}=d_{out}=2048, the width of Qwen2.5-3B						
128	26.0	25.1	25.8	0.99	0.96	1.03
4096	52.0	20.5	31.0	0.60	0.39	1.51
16384	172.5	20.0	26.0	0.15	0.12	1.30
d_{in}=d_{out}=3584, the width of Qwen2.5-7B						
128	38.0	37.4	50.7	1.33	0.98	1.35
4096	48.7	36.3	52.4	1.08	0.75	1.44
16384	366.2	36.6	49.2	0.13	0.10	1.35

Table 14: **Building ΔW becomes an order of magnitude cheaper than applying the two factors one at a time as the batch grows, and the nonlinearity adds a constant third on top.** Each row is one synthetic projection of the stated square shape, timed in isolation rather than inside a model; the widths are those of the backbones named beside them. The adapter operation is timed on its own, with no frozen projection present and no subtraction involved. This table asks how the two orders scale with the token count; Tab. 15 asks how large the kernel is beside the projection it sits in and measures an increment over that projection rather than an operation by itself, so the two sets of numbers answer different questions and are not comparable. One L20, bf16, r=32, medians of 200 iterations with the operation repeated inside each timing window and the measurement order shuffled across four rounds. The PEFT term grows with the token count while both of ours are flat in it, which is the $2Nr(d_{in}+d_{out})$ against $d_{in}d_{out}(2r+1)$ of Tab. 13; at the largest batch our kernel is between six and eight times cheaper than the factored order on the two Qwen widths. The last column prices the nonlinearity on its own against the same products without it, 1.30 and 1.35 on those two rows. Any timing below roughly 60 μs is set by framework dispatch rather than by arithmetic, so at the two smaller batches the three configurations are not separated by their cost laws.

Cost laws, written before measuring. All counts are taken against the same frozen projection $XW^\top$ carrying no adapter, charging a multiply-add as two operations and an elementwise pass as one per entry. In the forward pass the factored order costs $2Nr(d_{in}+d_{out})$ for a step over N tokens, while building ΔW costs $2d_{in}d_{out}r$ for the product and one pass of $d_{in}d_{out}$ to apply γ, so $d_{in}d_{out}(2r+1)$ in total and nothing at all in N; the nonlinearity is one further pass, giving $d_{in}d_{out}(2r+2)$. The backward pass is not symmetric between the two. The factored order differentiates through two thin products, $4Nr(d_{in}+d_{out})$, while the built order first accumulates a gradient of the shape of the update, $\nabla_{W+\Delta W} = \nabla_Y^\top X$ at $2Nd_{in}d_{out}$, and only then contracts it back into the two factors at $4d_{in}d_{out}r$; the nonlinearity adds one elementwise pass there and retains the activation its derivative consumes.

Three predictions follow, and the third is the one that ends up mattering. In the forward pass the configurations that build ΔW should be flat in the token count and should be the cheaper option once N exceeds $N^\star = \frac{2r+1}{2r}\,\frac{d_{in}d_{out}}{d_{in}+d_{out}}$, essentially half the harmonic mean of the two widths since the leading factor is $65/64$ at r=32. The two projections adapted in Qwen2.5-7B have different shapes and so different crossovers, 1820 tokens for the square 3584×3584 query projection and 455 for the GQA-narrow 3584×512 value projection; solving against the $\tanh$ law rather than the linear one replaces the leading factor by $(r+1)/r$ and moves the two to 1848 and 462. In the backward pass both orders grow with N, so what the forward pass saves the backward pass gives back, and over a full step the gap between the two implementations should be far smaller than either law is on its own. Memory should show the asymmetry reversed: the factored order has to materialise its own $N \times d_{out}$ output and then a sum, two activations that grow with the batch, while the built order folds the update into W and holds a fixed pair of $d_{in} \times d_{out}$ buffers.

What the kernel costs, in time. Tab. 14 times the extra term on its own. Both configurations that build ΔW are flat in N to within noise, 0.037 and 0.049 ms at $d = 3584$ across a hundredfold range of batch size, while the factored term rises from 0.038 to 0.366 ms, so by $N = 16384$ building the product is an order of magnitude cheaper on both Qwen widths, ratios of 0.10 and 0.12. That is the forward prediction. The paired decomposition of Tab. 15 and Fig. 12 adds the backward pass and confirms the second one: forward, building ΔW *saves* time against the factored order, $+0.090$ ms against $+0.877$, while with the backward pass included it costs $+3.946$ against $+1.979$, since the gradient has to cross the $d_{in} \times d_{out}$ matrix before contracting back into the two factors. The two directions cancel to the extent the laws say they should, which is why the choice of order is not what an end-to-end ratio measures.

What it does measure is the kernel, and on the same paired basis that accounts for 0.2 to 1.7% of a projection, 0.075 ms out of 11.5 at $d = 3584$ with the backward pass included. Against building the product alone the nonlinearity costs a factor of 1.3 to 1.4 on the two Qwen widths at the largest batch, where the adapter terms clear the dispatch floor. Its arithmetic is one pass, $(2r{+}2)/(2r{+}1)$ or 1.5% at $r{=}32$, so that factor is the traffic of the pass rather than its multiply-adds: the elementwise kernel reads ΔW and writes it back, roughly doubling the bytes moved over a matrix the product itself only writes once.

Model	frozen (ms)	extra time (ms), paired PEFT	ΔW	kernel	kernel share	extra MiB
forward						
768	0.19	+0.078	+0.009	**+0.002**	**1.3%**	3
2048	1.33	+0.423	+0.037	**+0.004**	**0.3%**	16
3584	3.95	+0.877	+0.090	**+0.009**	**0.2%**	49
forward + backward						
768	0.76	+0.261	+0.226	**+0.013**	**1.7%**	3
2048	4.78	+0.992	+1.317	**+0.025**	**0.5%**	16
3584	11.53	+1.979	+3.946	**+0.075**	**0.6%**	49

Table 15: **The kernel is at most 1.7% of a projection; building ΔW is what the batch pays for, and forward it pays negatively.** One synthetic adapted projection of the stated square width at $N = 16384$, measured on a standalone projection rather than inside a real model. Unlike Tab. 14, which times the adapter operation by itself, every time here is how much more the projection as a whole costs once the adapter is attached. Each extra time is a *paired* median: within one iteration the frozen projection and the variant are timed back to back and the difference recorded, because on a shared GPU drift reaches milliseconds while the quantity of interest is tens of microseconds. The ΔW column is ARK-Linear, the cost of building the update; **kernel** is the further cost of the elementwise $\tanh$ on top of it and **share** expresses it against the frozen projection. Forward, building ΔW costs less than the PEFTś two separate products ($+0.090$ against $+0.877$ ms at width 3584); with the backward pass included the order reverses, since gradients must cross the $d_{in} \times d_{out}$ matrix. The memory column is the extra peak allocation, fixed in N and equal to two bf16 copies of ΔW, against the two extra $N \times d_{out}$ activations of the PEFT adapter branch, which reach 48, 128 and 224 MiB at this batch for square widths 768, 2048 and 3584.

What the kernel costs, in memory. The last column of Tab. 15 confirms the reversal. Our extra allocation is fixed in N and equals the predicted two bf16 copies of the update, 49 MiB at $d = 3584$ against a predicted 49, while the two extra $N \times d_{out}$ activations of the factored order reach 224 MiB at the same batch, which is $2Nd_{out}$ entries in bf16. Between the two kernels the difference lies in how long the extra allocation stays resident rather than in how many bytes it occupies at its peak. Both form ΔW and add it into W, so both reach the same peak, at the moment when W and ΔW coexist, and that instantaneous peak is what a single-projection measurement records. Afterwards the linear kernel can release ΔW, since scaling and addition need nothing of it to differentiate, whereas $\tanh$ has to keep it, ΔW being the output its derivative consumes. One $d_{in} \times d_{out}$ tensor per adapted projection therefore stays resident through the backward pass under $\tanh$ and not under the linear kernel, and that is the term which accumulates over a whole model.

Composing back to the reported numbers. The laws now say which of the two reported ratios should show an effect at all. Qwen2.5-7B uses grouped query attention with four key-value heads, so a layer performs $2d^2 + 2d\,d_{kv}$ multiply-adds of attention projection with $d_{kv} = 512$, and $3d\,d_{\text{ff}}$ of feed forward with d_{ff} the feed-forward width, about 29 and 204 million per token; the two adapted projections contribute 3584^2 and 3584×512, together 6.3% of the layer. At 0.2 to 0.6% of a projection the kernel is under a tenth of a percent of a layer, so the end-to-end time cost is expected to sit inside run-to-run variance, which is where the reported $1.01\times$ sits. Memory is the term the laws say should show: the retained activation is 28 MiB per layer across the query and value projections and of the order of a gigabyte over the twenty eight layers of this backbone, against an 18.4 GiB baseline, and that is what the reported $1.08\times$ records. Both figures follow from timings taken on a single projection, with no quantity fitted to the end-to-end runs.

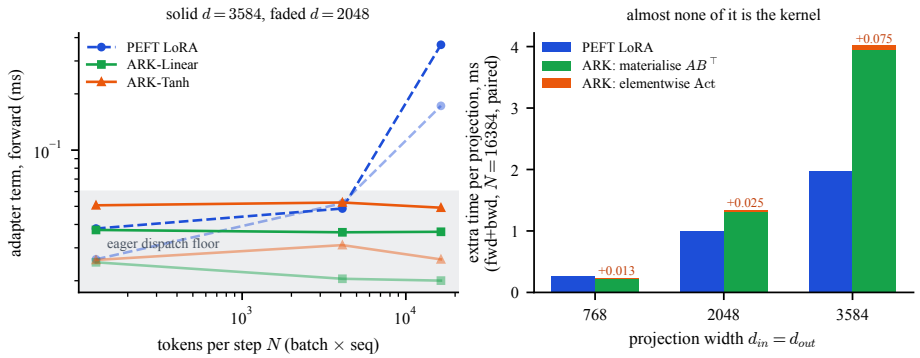


Figure 12: **Almost none of the overhead is the kernel.** Left, the adapter term timed on its own: LoRA's factor-by-factor product grows with the token count, while both configurations that build ΔW depend only on the layer shape and stay flat, so they become an order of magnitude cheaper by training scale. The shaded band marks timings below 0.06 ms, where framework dispatch rather than arithmetic sets the time. Right, the paired decomposition of one adapted projection with its backward pass: the green bar is the cost of building the update, the orange cap is the further cost of the elementwise $\tanh$, annotated in milliseconds. One L20, bf16, r=32, medians of two hundred iterations. Each configuration is one synthetic projection of the stated square width, timed in isolation rather than inside a model; 768, 2048 and 3584 are the widths of BERT-base and RoBERTa-base, of Qwen2.5-3B and of Qwen2.5-7B.

Protocol of the measurement. The measurements run on one L20 in bf16 at r=32, with the eager backend and the update recomputed on each forward call, matching the configuration used for Tab. 3. Timings are medians of two hundred iterations, with the operation repeated inside each timing window so that framework dispatch is amortised and the order of measurements shuffled across four rounds so that thermal drift cannot be charged to one configuration. Rows whose timings all fall below 0.06 ms are set by kernel launch rather than by arithmetic and are marked as such. Every extra term is a paired difference taken inside a single iteration, since subtracting two independently measured medians does not work at this scale: run-to-run drift on a shared accelerator reaches milliseconds while the quantity of interest is tens of microseconds.

E Positioning and Clarifications

E.1 Relation to Kernelized LoRA and Other Adaptive PEFT

Tab. 11 situates ARK among adaptive PEFT methods. Two families deserve explicit separation from ours.

Pruning from a Provisioned Budget. One branch provisions a large rank budget up front and then reduces it, whether by sensitivity-based pruning of singular values, by meta-learned selection over candidate rank-one components, or by rank-sorted truncation after training. What these share is adherence to the Budget-Coupled Rank Scaling Law: obtaining a high-rank update requires paying for it in physical parameters throughout training, and their adaptivity is a downward truncation from that predefined ceiling. ARK instead starts from a small latent dimension and raises the attainable numerical rank of the update at an unchanged parameter count.

Kernelized LoRA (SNELL, also KLoRA). The closest prior work inserts a kernel into the low-rank product (Shen et al., 2024). Its kernel is the piecewise-linear construction analyzed in App. C.8, where we show that it fails positive semi-definiteness for every P and every $\gamma \neq 0$; we therefore do not repeat that argument here. Two differences remain worth stating. First, the kernel serves a different purpose there: the method targets sparse tuning at low memory, and the kernel compensates for the capacity that sparsity removes. Second, its rank expansion is held in place by a competition-based sparsification mechanism added alongside the task loss, whereas in ARK the attainable rank follows from the kernel's own response to the pre-activation scale (App. C.4), with no auxiliary regulariser or mask.

ARK sets aside Mercer validity in favour of a rank-based criterion (App. C.2), which asks how much numerical rank a kernel induces and whether that amount moves with the kernel's own scale. This admits non-PSD choices such as Tanh and Swish, and makes the attainable rank a property of the

kernel that can be fitted and checked (Apps. C.4 and C.6) rather than something enforced by an external constraint.

E.2 Disambiguating "Kernel" in Computer Science

Within the broad context of computer science, the term "Kernel" carries distinct semantics depending on the specific subfield. To prevent conceptual ambiguity, we first differentiate three common homonymous concepts from the definition employed in this paper.

First, in Computer Vision, a kernel in Convolutional Neural Networks (CNN) typically refers to a weight tensor with fixed spatial dimensions (e.g., 3×3 or 5×5), which performs convolution operations via a sliding window to extract local features. Second, in the context of High-Performance Computing (HPC) and hardware programming (such as CUDA or Triton), a kernel denotes a specific computational routine or function executed in parallel on accelerators like GPUs; this concept pertains to the instruction set and parallel programming level. Third, in the domain of Operating Systems, the kernel refers to the core program responsible for managing hardware resources and system calls.

Distinct from the aforementioned definitions, the Adaptive-Rank Kernels proposed in this paper align with the definitions found in machine learning theory and functional analysis. Here a kernel is a binary function representing a generalized inner product, designed to construct high-dimensional feature spaces via implicit mappings. This usage is consistent with literature on SVM and NTK(Jacot et al., 2018).

Kernel Methods and the Kernel Trick In machine learning theory, Kernel Methods constitute a class of algorithms that analyze patterns by mapping data into high-dimensional feature spaces. The core mechanism driving these methods is the "Kernel Trick." Traditional linear algorithms rely on the Euclidean inner product $\langle \mathbf{x}, \mathbf{y} \rangle$ within the original space. The kernel trick directly substitutes this inner product operation with a non-linear kernel function $K(\mathbf{x}, \mathbf{y})$. This substitution enables the generalization of linear algorithms to non-linear scenarios without altering the logical structure of the algorithm itself.

Theoretically, a kernel function corresponds to a non-linear mapping $\phi : \mathcal{X} \to \mathcal{H}$ that projects data from the input space $\mathcal{X}$ to a high-dimensional (potentially infinite-dimensional) feature space. Directly computing this explicit high-dimensional mapping vector $\phi(\mathbf{x})$ is computationally prohibitive or even infeasible. However, the kernel trick uses the equality $K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$ to transform a complex, invisible, implicit high-dimensional mapping into a concrete, efficiently computable function in the low-dimensional input space. This mechanism allows models to exploit the geometric properties of high-dimensional spaces for learning without explicitly constructing high-dimensional features. In this paper, we also use the Kernel Trick to avoid an explicit cost scaling, but we recast the selection criterion in terms of the rank a kernel induces and define the ARK family accordingly in Sec. 4.1.

E.3 Activation Functions in Neural Architectures

There exists a fascinating functional duality in deep learning: certain element-wise non-linearities can serve fundamentally different purposes depending on where they are injected into the architectural hierarchy.

Activations on Features vs. Parameters. In standard neural networks, non-linear activation functions (e.g., ReLU, GeLU) are strictly applied to the forward-pass feature activations $H_{out} = \sigma(H_{in}W)$. This paradigm extends to standard PEFT methods like the **Adapter** (Houlsby et al., 2019). Adapters insert small bottleneck feed-forward modules ($\Delta H = \sigma(HW_{down})W_{up}$) into the network. While the non-linearity is crucial for learning complex representations, applying it to the *features* (H) strictly prevents the adapter from being structurally merged (re-parameterized) into the original frozen weights, resulting in unavoidable inference latency.

In stark contrast, ARK completely shifts the operational domain of the activation function from the feature space (H) to the *parameter space* (W). By applying the non-linearity to the weight decomposition itself ($\Delta W = \text{Act}(\gamma AB^T)$), ARK uses it not to activate individual neurons, but to smoothly warp a low-rank parameter geometry into a high-rank structure. This preserves feature-

parameter equivalence (App. B.3), so ARK can be merged for zero-cost inference, which distinguishes it from feature-space Adapters.

The S-Shaped Curve: From Normalization to Rank Expansion. Classical functions with S-shaped saturation curves, such as the hyperbolic tangent ($\tanh$) and the logistic sigmoid, have long served as "smooth switches." A recent, highly relevant work is **DyT (Dynamic Tanh)** (Zhu et al., 2025). DyT introduces an element-wise operation $\mathrm{DyT}(x) = \tanh(\alpha_{\mathrm{DyT}}\, x)$ as a drop-in replacement for Layer Normalization (LN) applied to feature activations in Transformers. Their core motivation stems from the observation that LN inherently produces an S-shaped input-output mapping; thus, modulating the feature scale α_{DyT} allows Transformers to achieve stable training without explicit normalization layers.

While the mathematical form of ARK-Tanh ($\tanh(\gamma AB^T)$) shares a superficial resemblance with DyT, their **functional goals, operational domains, and mechanisms are entirely distinct**. First, DyT is a *feature-space* operation designed to replace normalization layers for training stability, whereas ARK is a *parameter-space* operation designed for parameter-efficient fine-tuning. Second, the learnable scale in DyT (α_{DyT}, which is unrelated to our post-activation scale α) modulates the variance of forward-pass activations to prevent gradient explosion/vanishing. In ARK, the same scalar sits on the parameter side: γ scales the inner product before the non-linearity, so that the standard deviation of the pre-activation entries, and with it the attainable numerical rank of ΔW, moves along the kernel's response curve. Ultimately, ARK uses the S-shaped saturation property not for feature normalization, but to obtain a fittable relation between that scale and the expected numerical rank under a strict parameter budget (App. C.4).