GRADIENT DESCENT WITHOUT REPEATED BACKPROPAGATION VIA LAZY LOW-RANK JACOBIANS

Anonymous authors
Paper under double-blind review

ABSTRACT

Backpropagation is the standard and highly effective mechanism for computing gradient-descent updates in neural networks. We explore whether part of its repeated derivative computation can be amortized by exploiting structure that emerges in large, overparameterized models. Our approach builds on two such properties: the *lazy regime*, where the Jacobian changes only mildly during training, and substantial low-rank structure in the Jacobian. Together, these suggest that derivative information can be compressed once and reused across optimization. We develop practical tools for testing these properties without explicitly forming or storing the full Jacobian or Hessian, including a Hessian–Jacobian measure of laziness and a scalable rank-estimation procedure based on a small subset of model parameters. We further show that, under suitable parameterizations, parameter-efficient fine-tuning inherits the laziness of the base model. Building on these observations, we construct a training procedure that reuses a compact low-rank Jacobian representation to approximate standard gradient-descent updates without repeated network-wide backpropagation. The forward computation remains unchanged, while the costly network-wide backward pass is replaced by inexpensive updates over a compact Jacobian representation, making parameter updates nearly negligible compared with standard backpropagation. Our goal is not to replace backpropagation as a general-purpose training method, but to show that gradient-based learning can be reorganized around the mathematical structure of overparameterized networks, substantially reducing the need for repeated backpropagation. Experiments show that this alternative computational organization remains viable in practice on realistic architectures and datasets.

1 INTRODUCTION

Backpropagation is the standard and highly effective mechanism for training modern neural networks, providing an efficient way to compute parameter updates from the loss. As models become increasingly large and overparameterized, however, they can exhibit mathematical structure that is absent from generic optimization problems. This motivates exploring whether such structure can be used to reorganize parts of gradient-based training while preserving its conventional updates. In particular, it raises the possibility that a substantial fraction of the derivative computation normally repeated throughout training can instead be compressed and reused.

We focus on two properties that arise in overparameterized networks. Neural tangent kernel (NTK) theory identifies a *lazy regime* in which training remains close to the model's local linearization and the Jacobian changes only mildly (Jacot et al., 2018; Lee et al., 2019; Chizat et al., 2019; Liu et al., 2020). At the same time, neural-network Jacobians can exhibit substantial low-rank structure, with their effective dimension far smaller than their ambient size (Oymak et al., 2019). These properties are usually studied as descriptions of optimization dynamics. Here, we take a different perspective: a Jacobian that is both nearly constant and effectively low-dimensional contains derivative information that may not need to be recomputed from scratch throughout training.

For this perspective to be useful, however, these properties cannot simply be assumed from model size. A central contribution of this work is therefore to make them practically testable. We provide scalable diagnostics for determining whether the Jacobian changes sufficiently little and whether it admits a useful low-dimensional representation, without requiring the prohibitively large Jacobian or

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.

Hessian matrices to be explicitly formed or stored. This allows the structural assumptions underlying our approach to be verified in the setting where they are to be exploited.

The resulting viewpoint leads to a different way of carrying out gradient-based learning. Rather than replacing gradient descent with a new learning rule, we reuse the derivative structure underlying the conventional gradient and approximate subsequent parameter updates without repeated network-wide backpropagation. This has an important computational consequence. In standard backpropagation, the cost of producing an update is coupled to the computation through which parameters are repeatedly used in the model. Our approach separates these costs: the forward computation still scales with input size, while the repeated backward component is replaced by computation over a compact, reusable representation. This decoupling prevents the backward-update cost from growing with repeated parameter use, particularly in architectures with extensive parameter sharing such as convolutional networks and Transformers.

The same perspective becomes especially relevant for modern pretrained models, where even a low-rank Jacobian representation can remain expensive because the stored projection still scales with the number of trainable parameters. PEFT methods such as LoRA and BitFit (Hu et al., 2021; Zaken et al., 2022) reduce this dimension dramatically, potentially allowing us to exploit the laziness of a large base model while storing derivative information only for a small trainable parameter space. This requires laziness to survive the PEFT parameterization, which is not automatic. We introduce *laziness inheritance* and prove that specific PEFT parameterizations preserve the near-constant Jacobian behavior of the base model.

We view our method primarily as a conceptual demonstration that gradient-based learning need not always be organized around repeated backpropagation, rather than as a replacement for backpropagation as a general-purpose optimizer. Backpropagation remains extremely effective; our goal is to explore what becomes possible when laziness and low-rank structure are exploited explicitly. Our experiments test whether this perspective extends beyond theory, showing that these properties can be identified and exploited across realistic architectures and datasets. The results provide a proof of concept for reorganizing and amortizing substantial parts of the conventional training pipeline.

Our contributions are fourfold: (i) a practical way to determine whether a network operates sufficiently close to the lazy regime without explicitly constructing the large derivative matrices involved; (ii) a scalable way to identify and exploit the effective low-rank structure of its Jacobian; (iii) a laziness-inheritance result showing when PEFT preserves the lazy behavior of a large base model while restricting derivative storage and training to a much smaller parameter space; and (iv) we combine these properties into a proof-of-concept training procedure that approximates conventional gradient-descent updates without repeated network-wide backpropagation. Rather than seeking to replace backpropagation, this demonstrates that its repeated backward computation can be reorganized when the structure of overparameterized models is exploited, opening an alternative direction for gradient-based learning.

1.1 Notations

Let $D = \{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$ be a dataset of n input-output pairs, with $x^{(i)} \in \mathbb{R}^{d_x}$ and $y^{(i)} \in \mathbb{R}^{d_y}$, where d_x and d_y denote the input and output dimensions. Let $f(w; x) \in \mathbb{R}^{d_y}$ be a neural network with parameters $w \in \mathbb{R}^m$. Subscripts index entries or time, while superscripts index data instances. For a loss $\ell(y, f(w; x))$, we define its average over $S \subseteq D$ as

$$\ell(w; S) = \frac{1}{|S|} \sum_{(x^{(i)}, y^{(i)}) \in S} \ell\left(y^{(i)}, f(w; x^{(i)})\right). \tag{1}$$

We denote the model Jacobian at time t by $J_t \in \mathbb{R}^{n \times d_y \times m}$, with $[J_t]_{i,:,:} = \nabla_w f(w_t; x^{(i)}) \in \mathbb{R}^{d_y \times m}$. We denote the corresponding output-space loss gradients by $a_t \in \mathbb{R}^{n \times d_y}$, with $[a_t]_{i,:}^\top = \nabla_f \ell\left(y^{(i)}, f(w_t; x^{(i)})\right) \in \mathbb{R}^{d_y \times 1}$.

We use uppercase R for the reference-set size and lowercase r for the Jacobian rank.

2 Related Work

Neural tangent kernels and training regimes. The neural tangent kernel (NTK) establishes that sufficiently wide networks can evolve close to their linearization, with an approximately constant Jacobian throughout training (Jacot et al., 2018; Lee et al., 2019; Chizat et al., 2019). Subsequent work characterizes transitions between lazy and feature-learning regimes and how they depend on width, scaling, initialization, and parameterization (Geiger et al., 2020; Woodworth et al., 2020; Liu et al., 2020; Tu et al., 2024; Domine et al., 2025; Chen et al., 2025). Recent results further emphasize that the validity of the lazy or constant-NTK approximation can depend sensitively on the optimization regime, parameterization, and training objective (Yu et al., 2025). These works primarily characterize when lazy behavior emerges or breaks down. We instead treat laziness as an empirical property that can be verified before it is exploited. To our knowledge, we are the first to provide practical, scalable diagnostics for both near-constant Jacobian behavior and effective Jacobian low-rankness without explicitly forming the corresponding large derivative matrices.

Low-rank Jacobians and efficient derivative representations. Neural-network Jacobians and gradients often concentrate in low-dimensional subspaces, a structure used to study optimization and generalization to reduce training memory (Oymak et al., 2019; Gur-Ari et al., 2018; Larsen et al., 2021; Zhao et al., 2024; Jaiswal et al., 2024; Refael et al., 2025; Robert et al., 2025). In particular, Oymak et al. (2019) connect low-rank Jacobian structure with near-initialization training, but primarily as an analytical tool. Related work also accelerates empirical NTK computation through sketching, structured contractions, output-space approximations, and Jacobian projections (Zandieh et al., 2021; Novak et al., 2022; Mohamadi et al., 2023; Mahowald et al., 2026). These methods focus on constructing or applying the kernel $K = JJ^\top$, whereas we require a compact parameter-Jacobian representation that can be reused for parameter updates throughout optimization.

Linearized fine-tuning and PEFT. Pretrained models can be fine-tuned through their first-order linearization using tangent features or Jacobian–vector products rather than standard nonlinear training (Achille et al., 2021; Ortiz-Jimenez et al., 2023; Liu et al., 2024; Tang et al., 2024). These results demonstrate that the initialization Jacobian can support effective adaptation without being explicitly materialized. In parallel, parameter-efficient fine-tuning methods such as LoRA and BitFit restrict adaptation to a small trainable parameter space (Hu et al., 2021; Zaken et al., 2022), while recent theory connects PEFT to kernel and linearized training, including NTK behavior, LoRA optimization within the NTK regime, and departures from lazy dynamics (Malladi et al., 2023b; Jang et al., 2024; Dayi & Chen, 2025; Kim et al., 2025). We complement this literature by studying *laziness inheritance*: whether restricting an already-lazy model to a PEFT parameterization preserves the near-constant Jacobian property, and which parameterizations guarantee such preservation.

Backpropagation-free and forward-only training. Prior work avoids reverse-mode backpropagation using alternative update rules, forward-gradient or zeroth-order methods (Boopathy & Fiete, 2022; Baydin et al., 2022; Malladi et al., 2023a; Radhakrishnan et al., 2024). Recent approaches likewise replace repeated backward passes with forward-mode estimates or one-time calibration (Qin & Huang, 2026; Xia et al., 2026). Our method, instead, remains gradient descent: rather than replacing the conventional parameter update, we approximate it using a reusable low-rank representation of the parameter Jacobian, avoiding repeated network-wide backpropagation.

Prior work establishes lazy training, low-dimensional derivative structure, linearized optimization, and backpropagation-free learning. We connect these operationally: verify the required structure, compress the nearly constant Jacobian into a reusable low-rank representation, characterize its preservation under PEFT, and use it for gradient-descent updates without repeated backpropagation.

3 Understanding neural networks in the lazy regime

This section states the two assumptions behind our method, a low-rank Jacobian and laziness, and shows how to verify both at scale. We then build a gradient-descent algorithm that reuses the Jacobian instead of recomputing it, and establish when PEFT preserves laziness.

Our starting point is the standard gradient-descent update

$$w_{t+1} = w_t - \eta \nabla_w \ell(w_t; S),$$

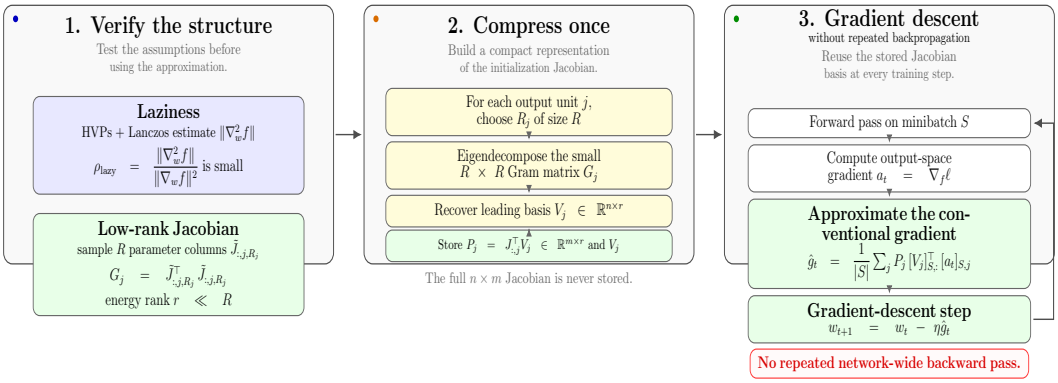

Figure 1: **Overview.** We first verify Jacobian low-rankness and laziness, then compress the Jacobian once and reuse it for gradient-descent updates without repeated backpropagation.

where η is the learning rate. Using the notation of Section 1.1, the chain rule gives

$$w_{t+1} = w_t - \frac{\eta}{|S|} \sum_{(x^{(i)}, y^{(i)}) \in S} [J_t]_{i,:,:}^\top [a_t]_{i,:}^\top. \tag{2}$$

In the lazy regime, the Jacobian changes only mildly during training, so $J_t \approx J_0$. Writing $J := J_0$, Equation (2) becomes

$$w_{t+1} \approx w_t - \frac{\eta}{|S|} \sum_{(x^{(i)}, y^{(i)}) \in S} [J]_{i,:,:}^\top [a_t]_{i,:}^\top. \tag{3}$$

Thus, the parameter Jacobian is approximately fixed and reusable across training steps, while the output-space gradient a_t remains time-varying—a factorization central to our approach.

3.1 Algebraic Perspective of Laziness: The Assumption of Low-Rank Jacobian

The lazy-regime assumption suggests a simple strategy: if the Jacobian remains approximately constant throughout training, one could compute it once at initialization, store it, and reuse it for subsequent gradient-descent updates. In practice, however, this is infeasible. For n examples, output dimension d_y, and m model parameters, the full Jacobian contains nd_ym entries, making its construction and storage prohibitively expensive.

This motivates exploiting additional structure in the Jacobian. The assumption that the model Jacobian J has a low-rank structure in large networks (Oymak et al., 2019) suggests that its parameter derivatives can instead be represented compactly. For each output unit j, we view $[J]_{:,j,k}$ as an n-dimensional representation of the k'th weight, describing its effect on the model output across all training examples. Under the low-rank assumption, these representations lie approximately in a much lower-dimensional subspace. We therefore seek to reduce their dimension from n to $r \ll n$ using the principal directions of the Jacobian covariance $\widetilde{J}_{:,j,:}\widetilde{J}_{:,j,:}^\top$, where $\widetilde{\cdot}$ denotes row-centering. Directly computing these directions, however, would itself require forming and decomposing an $n \times n$ matrix and is therefore impractical at scale.

To make this computation feasible, we approximate the same dominant subspace using only a small reference set $\mathcal{R}$ of R informative weights (one set $\mathcal{R}_j$ per output unit; we omit j when fixed). Since the Jacobian is assumed to be effectively low-rank, a sufficiently representative subset of its columns can preserve its dominant variation. In particular, for uniformly sampled columns, $\frac{1}{R}\widetilde{J}_{:,j,\mathcal{R}}\widetilde{J}_{:,j,\mathcal{R}}^\top$ is an unbiased sample approximation of $\frac{1}{m}\widetilde{J}_{:,j,:}\widetilde{J}_{:,j,:}^\top$, and approaches it with high probability as R grows. Equivalently, $\underbrace{\widetilde{J}_{:,j,\mathcal{R}}^\top \widetilde{J}_{:,j,\mathcal{R}}}_{R \times R} \overset{\text{same nonzero spectrum}}{\longleftrightarrow} \underbrace{\widetilde{J}_{:,j,\mathcal{R}}\widetilde{J}_{:,j,\mathcal{R}}^\top}_{n \times n} \approx \frac{R}{m}\widetilde{J}_{:,j,:}\widetilde{J}_{:,j,:}^\top$. The first relation is made precise by Proposition 3.1, while the factor R/m only rescales the eigenvalues. When the leading r directions are spectrally separated, the dominant eigenspace of the reference covariance therefore remains close to that of the full Jacobian covariance; the formal concentration and perturbation argument is given in Appendix H.1.

We therefore operate only on $\widetilde{J}_{:,j,\mathcal{R}} \in \mathbb{R}^{n\times R}$, with $R \ll m$. The following elementary linear-algebra property allows the relevant n-dimensional eigenspace to be recovered through an $R \times R$ eigendecomposition rather than an $n \times n$ one.

Proposition 3.1. *Let $A \in \mathbb{R}^{n\times R}$ with $n > R$. If $\phi \in \mathbb{R}^R$ is an eigenvector of $A^\top A$ with eigenvalue $\lambda > 0$, then $\psi := A\phi/\sqrt{\lambda} \in \mathbb{R}^n$ is an eigenvector of $AA^\top$ with the same eigenvalue λ.*

Applying Proposition 3.1 with $A = \widetilde{J}_{:,j,\mathcal{R}}$, the eigenvectors of the $n \times n$ covariance matrix $\widetilde{J}_{:,j,\mathcal{R}}\widetilde{J}^\top_{:,j,\mathcal{R}}$ can be recovered from the eigendecomposition of the much smaller $R \times R$ Gram matrix $\widetilde{J}^\top_{:,j,\mathcal{R}}\widetilde{J}_{:,j,\mathcal{R}}$, with $R \ll n$. We retain the r leading directions, yielding an $n\times r$ basis for the dominant sample-space variation of the Jacobian and an r-dimensional reduced representation of its parameter columns. This compact representation is developed into the training procedure in Section 3.3.

3.2 Empirical verification of the laziness and low-rank assumptions

The proposed framework relies on two mathematical assumptions regarding the Jacobian: (i) "laziness": the Jacobian remains nearly constant throughout training, and (ii) the Jacobian is low-rank. We now present methods to quantitatively measure the extent to which these assumptions are satisfied in a given setting consisting of a model and training data.

Laziness. Following Chizat et al. (2019), we quantify laziness by the ratio $\kappa_t := \frac{\|H_t\|\,\|a_t\|}{\|J_t\|^2}$, where H_t is the model Hessian, i.e., the derivative of the Jacobian with respect to the parameters. $\|H_t\|$ measures the sensitivity of the Jacobian to parameter changes, while $\|J_t\|^2$ controls how strongly a gradient step changes the model output under the local linear approximation. Thus, a small κ_t serves as an indicator of laziness: nonlinear curvature is weak relative to the local first-order behavior, so the Jacobian is expected to change only mildly during training. The precise relation between κ_t, the gradient-descent step size, and relative Jacobian drift is derived in Appendix F.

Since κ_t also depends on the current loss gradient through $\|a_t\|$, we remove this factor and define the model-side diagnostic $\rho_t := \frac{\|H_t\|}{\|J_t\|^2}$. For softmax cross-entropy, the per-example output gradient satisfies $\|a_t\|_2 \le \sqrt{2}$, so ρ_t controls κ_t up to a bounded factor.

We use the operator norm because it bounds the first-order change in the Jacobian caused by a parameter update, making it the natural quantity for measuring Jacobian drift. Computing the Hessian explicitly is infeasible, since even for a single scalar output it contains m^2 entries. For a fixed input x and output unit j, however, the Hessian $H_{j,t} := \nabla^2_w f_j(w_t; x)$ is symmetric, so its operator norm can be estimated from its dominant eigenvalue using a few Lanczos (Lanczos, 1950) iterations.

Lanczos requires only Hessian–vector products, which we approximate using finite differences: $H_{j,t}v \approx \frac{\nabla_w f_j(w_t+\epsilon v;x)-\nabla_w f_j(w_t-\epsilon v;x)}{2\epsilon}$. This yields a matrix-free estimate requiring only two gradient evaluations per Lanczos iteration. We repeat the procedure across several inputs and output units to obtain a network-level laziness measure. We also examine how ρ changes with model width. Prior wide-network analyses show that the Hessian norm decreases approximately as $\tilde{O}\Big(\frac{1}{\sqrt{\text{width}}}\Big)$, while the Jacobian norm remains $O(1)$ (Liu et al., 2020). Hence, ρ is expected to scale approximately as $\frac{1}{\sqrt{\text{width}}}$, which we use as an additional consistency check across model widths. For the denominator we use the same input and unit: the corresponding row of the Jacobian is the gradient $\nabla_w f_j(w_t; x)$, so $\rho_j(x) = \|H_{j,t}\|/\|\nabla_w f_j(w_t; x)\|^2$ (Appendix F).

Low-rank Jacobian. To measure whether the Jacobian admits a compact low-dimensional representation, we examine the decay of its singular values. For each output unit j, let $\widetilde{J}_{:,j,:} \in \mathbb{R}^{n\times m}$ denote the corresponding centered Jacobian slice, with singular values $\sigma_1 \ge \sigma_2 \ge \cdots$. We define the *energy rank* r_α as the smallest r such that $\frac{\sum_{i=1}^r \sigma_i^2}{\sum_i \sigma_i^2} \ge \alpha$. Thus, r_α is the number of singular directions required to retain an α fraction of the slice's total squared Frobenius energy. A small r_α relative to the maximal possible rank indicates that the Jacobian slice is effectively low-rank. In particular, $r_{0.95}$ is the smallest rank that retains 95% of this energy.

To evaluate this quantity efficiently, we use the much smaller reference slice $\widetilde{J}_{:,j,R}$. By Proposition 3.1, its nonzero spectrum can be obtained from the $R \times R$ matrix $\widetilde{J}_{:,j,R}^{\top}\widetilde{J}_{:,j,R}$, avoiding operations on the full Jacobian. Appendix H justifies when the reference slice preserves the dominant structure of the full slice. We compute $r_{0.95}$ on subsets of $n' \leq n$ samples and examine whether it stabilizes well below the maximal rank $\min(n', R)$. Such saturation indicates that most of the Jacobian energy remains concentrated in a small number of directions.

3.3 Gradient descent without repeated backpropagation

Our goal is an efficient and theoretically sound gradient-descent algorithm that exploits the laziness and low-rank structure of the Jacobian to avoid repeated backpropagation.

The preceding sections provide the two ingredients needed by our training procedure: a nearly constant Jacobian and a low-dimensional representation of its dominant directions. Explicitly constructing and storing the full Jacobian is infeasible at realistic scales, since it contains nd_ym entries and therefore requires $O(nd_ym)$ memory. We therefore compute only a compact low-rank representation during preprocessing and reuse it throughout training to approximate the conventional gradient-descent update without repeated network-wide backpropagation.

Algorithm 1 Gradient descent without repeated backpropagation

Require: An initialized neural network model f, a loss function ℓ, reference-set size R, rank r
  Pre-processing phase:
  Select a reference set $\mathcal{R}_j$ of R weights for each output unit j using Algorithm 2.
  for each output unit j **do**
    Compute the reference slice $[J]_{:,j,\mathcal{R}_j} \in \mathbb{R}^{n\times R}$, the derivatives with respect to the weights in $\mathcal{R}_j$ only, streaming over the data.
    Center the rows of $[J]_{:,j,\mathcal{R}_j}$.
    Obtain the leading r eigenvectors $V_j \in \mathbb{R}^{n\times r}$ of $[\tilde{J}]_{:,j,\mathcal{R}_j}[\tilde{J}]_{:,j,\mathcal{R}_j}^{\top}$ via eigendecomposition of the $R \times R$ matrix $[\tilde{J}]_{:,j,\mathcal{R}_j}^{\top}[\tilde{J}]_{:,j,\mathcal{R}_j}$ (Proposition 3.1).
    Store V_j, and accumulate the projection $P_j := [J]_{:,j,:}^{\top}V_j \in \mathbb{R}^{m\times r}$ by streaming over the data.
  end for
  Training phase:
  for each training batch S **do**
    Compute $[a_t]_{i,:}$ for each i s.t. $\left(x^{(i)}, y^{(i)}\right) \in S$.
    Update the network weights via (3), i.e., $w \leftarrow w - \frac{\eta}{|S|}\sum_j P_j[V_j]_{S,:}^{\top}[a_t]_{S,j}$.
  end for
  return trained model

Algorithm 1, illustrated in Figure 1, summarizes the resulting training procedure. For each output unit j, we select a separate reference set $\mathcal{R}_j$ of R parameter indices. Specifically, we accumulate the absolute loss-gradient magnitude over training examples whose label is j and retain the R parameters with the largest accumulated magnitude (Algorithm 2, Appendix B.1). The resulting set $\mathcal{R}_j$ is used to form the reference Jacobian $\widetilde{J}_{:,j,\mathcal{R}_j}$ and recover its leading basis $V_j \in \mathbb{R}^{n\times r}$. We then project these directions back to parameter space, $P_j := J_{:,j,:}^{\top}V_j \in \mathbb{R}^{m\times r}$, and store (V_j, P_j) for use during training. In practice, the sum defining P_j is estimated using a class-balanced subset of $n_{\text{pre}} \leq n$ training examples and rescaled by n/n_{pre}, avoiding a full-dataset preprocessing pass while preserving the scale of the projection.

At each training step, only the output-space loss gradient is recomputed. For minibatch S, we approximate the parameter gradient as $\widehat{g}_t = \frac{1}{|S|}\sum_j P_j[V_j]_{S,:}^{\top}[a_t]_{S,j}$, and update $w_{t+1} = w_t - \eta\widehat{g}_t$ (Appendix H.2 bounds the error of $\widehat{g}_t$). Thus, after preprocessing, no network-wide backward pass is required. In terms of complexity, for each output unit j, V_j has size $n \times r$ and P_j has size $m \times r$. Both can be constructed by streaming over the data, without materializing the full Jacobian. Treating d_y and r as constants, the resulting storage is $O(m + n)$. For a fixed minibatch size, the parameter update requires $O(m)$ time.

A key computational advantage of our method is that, after preprocessing, the parameter-update cost is decoupled from input length. In architectures with extensive parameter reuse, such as CNNs, RNNs, and Transformers, standard backpropagation becomes increasingly expensive as image resolution or sequence length grows, since gradients must be propagated through every parameter use. In contrast, our forward pass retains the usual input-size dependence, while the parameter update is computed directly from the stored (V_j, P_j) representation. Consequently, its backward-update cost does not grow with sequence length or spatial resolution, yielding increasing savings as inputs become larger; we demonstrate this empirically in Section 4.2 and Appendix C.

3.4 Laziness inheritance under parameter-efficient fine-tuning

For very large pretrained models, even a low-rank representation can remain costly because $P_j \in \mathbb{R}^{m\times r}$ still scales with the number of trainable parameters. PEFT offers a natural remedy: restricting training to $m_{\mathrm{PEFT}} \ll m$ parameters reduces this storage to $m_{\mathrm{PEFT}}r$. However, Algorithm 1 relies on reusing a nearly fixed Jacobian, so under PEFT we require the Jacobian with respect to the PEFT parameters to remain nearly constant.

We introduce *laziness inheritance*: a characterization of when a PEFT parameterization preserves the near-constant Jacobian structure of the base model. Importantly, base-model laziness alone is not sufficient, since the reparameterization itself may introduce additional Jacobian drift.

For LoRA, write $W = W_0 + sBA$, where A and B are the adapter factors and s is the LoRA scaling. We state the main results here; formal statements and proofs are in Appendix G.

Fixed-A LoRA inherits laziness. When $A = A_0$ is frozen and only B is trained (Zhang et al., 2023), the map from B to W is affine and introduces no additional reparameterization curvature; the adapter-Jacobian drift is controlled by the base-model drift up to the conditioning of A_0 (Theorem G.4).

Two-factor LoRA need not inherit laziness. When both A and B are trained, the bilinear map BA can itself induce Jacobian drift even when the base Jacobian is constant (Propositions G.1 and G.2).

BitFit adds no reparameterization curvature. As an affine restriction to the bias coordinates, BitFit introduces no additional reparameterization-induced curvature (Theorem G.7).

These results identify PEFT parameterizations that add no drift of their own. We therefore use fixed-A LoRA and BitFit, allowing the lazy structure of a large pretrained model to be exploited while storing and updating only a small PEFT-space Jacobian representation.

4 Experimental Results

Setup. We fine-tune CLIP ViT-B/L/H/g/G (Radford et al., 2021; Cherti et al., 2023) (widths 768–1664), and EVA02-E (4.4B) (Sun et al., 2023), with fixed-A LoRA plus a linear head, on EuroSAT (Helber et al., 2019), Stanford Dogs (Khosla et al., 2011), and CIFAR-100 (Krizhevsky et al., 2009). Our method uses reference sets of R=5000 and retained rank r=200. We compare against the stronger two-factor LoRA trained with backpropagation (BP), with the same data and schedule and a tuned learning rate for each method. Full experimental details are given in Appendix A, while reference-set selection is described in Appendix B.1.

4.1 Verifying the assumptions

Low rank. Figure 2(a) directly tests the low-rank assumption of Section 3.2 by measuring the 95% energy rank $r_{0.95}$ as the number of samples n' increases. While the maximal possible rank $\min(n', R)$ continues to grow, the measured energy rank quickly stabilizes far below it. At full data, we obtain $r_{0.95} = 107$ on EuroSAT and 366 on Stanford Dogs, compared with a maximal rank of 5000. Because the selected reference columns capture nearly all of the full slice's energy, the measured energy ranks closely reflect those of the full Jacobian slices (Appendix H.3). Together, these results provide strong empirical support for the low-rank structure exploited by our method: even as more data are added, most of the Jacobian energy remains concentrated in a small number of directions.

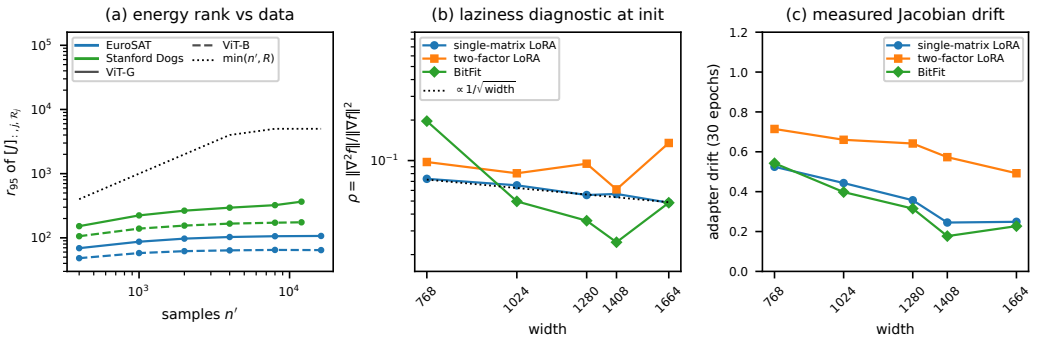

Figure 2: **Low-rank structure and laziness across model scales.** (a) 95% energy rank $r_{0.95}$ versus training samples n' for ViT-B (dashed) and ViT-G (solid); the dotted line is $\min(n', R)$. (b) Initialization laziness diagnostic ρ across widths for fixed-A LoRA, two-factor LoRA, and BitFit, with the predicted $1/\sqrt{\text{width}}$ scaling. (c) Relative adapter-Jacobian drift after 30 epochs of BP fine-tuning at learning rate 10^{-3}, well above the rates used for training, for the same configurations.

Laziness. Our goal is to test whether the initialization diagnostic ρ predicts the Jacobian drift that is later realized during training. The diagnostic of Section 3.2 requires two first-order gradient evaluations per Hessian–vector product and only a few Lanczos iterations, and runs on the full ViT-G in minutes without forming a Hessian or Jacobian.

Figure 2(b) shows that for fixed-A LoRA, ρ decreases from 0.073 on ViT-B to 0.049 on ViT-G. Fitting the measurements across model widths to a power law gives an exponent of -0.52, very close to the -0.5 predicted by the $1/\sqrt{\text{width}}$ scaling discussed in Section 3.2. The two-factor LoRA is 1.1–$2.8\times$ higher and does not exhibit the same width dependence, while BitFit approaches fixed-A LoRA at the largest width. This behavior is consistent with our theory: fixed-A LoRA inherits base-model laziness, two-factor LoRA need not, and BitFit introduces no additional reparameterization curvature (Appendix G).

Figure 2(c) directly tests the predictive value of ρ. For each of the five model widths and three parameterizations, we compare the initial value of ρ with the relative measured adapter-Jacobian drift. Across these 15 configurations, the two quantities have a Spearman rank correlation of 0.82: configurations with larger initial ρ tend to exhibit larger Jacobian drift. Moreover, at four of the five model widths, ρ correctly reproduces the ordering of the three parameterizations by their measured drift. Thus, the initialization diagnostic closely tracks the Jacobian drift realized during training. The same holds on ConvNeXt CLIP (Appendix E).

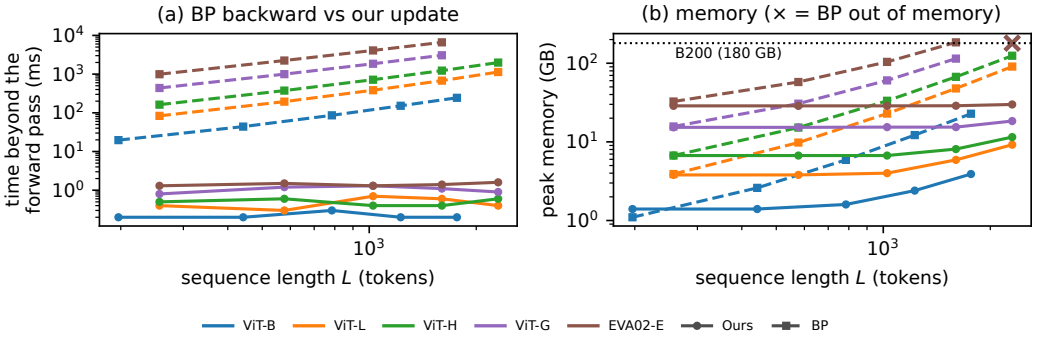

Figure 3: **Cost of the parameter update vs sequence length.** (a) Step time minus forward time: Ours (solid) is flat for all five backbones; BP (dashed) grows linearly with L. (b) Peak memory. BP uses the same fixed-A adapter as ours; two-factor LoRA would only be slower.

4.2 Complexity and Compute

Figure 3 demonstrates the main computational advantages of our method in both time and memory. For standard BP, the backward-pass cost grows approximately linearly with sequence length, whereas our parameter-update cost is nearly independent of it, remaining only 0.2–1.6 ms across all

tested backbones while BP grows to seconds. Consequently, the full training step approaches inference cost, requiring about one forward-pass equivalent (at most 1.01) compared with 2.06 for BP. Importantly, the practical advantage can be substantially larger than the roughly $2\times$ gap suggested by forward/backward operation counts: in mixed precision, standard BP costs 2.47–$3.65\times$ our step, while activation-checkpointed BP reaches 3.30–$4.55\times$. Appendix C analyzes these effects in detail.

The memory advantage follows from avoiding the backward computation graph. As shown in Figure 3, BP requires increasingly large activation storage as sequence length grows and reaches out-of-memory in the largest settings, whereas our method maintains substantially lower peak memory. Appendix C further quantifies this advantage in terms of usable batch size, with 28–$33\times$ larger batches on ViT-G than standard BP. It also compares against activation checkpointing, which reduces memory by paying for additional forward recomputation; even then, it supports 1.8–$4.3\times$ smaller batches than our method on the tested Transformers. The appendix further extends the analysis to convolutional backbones. Our method avoids this memory–compute trade-off altogether: it requires neither backward activation storage nor checkpointing recomputation, while retaining a negligible parameter-update cost.

Table 1: **Test accuracy across training epochs on ViT-G.** Both methods use the same data and schedule, each with its own tuned learning rate. Columns report actual training epochs.

Method / Dataset	**15**	**20**	**25**	**30**
EuroSAT BP	$.988 \pm .004$	$.989 \pm .002$	$.990 \pm .001$	$.989 \pm .001$
EuroSAT Ours	$.975 \pm .006$	$.977 \pm .003$	$.977 \pm .006$	$.978 \pm .003$
Stanford Dogs BP	$.863 \pm .014$	$.866 \pm .009$	$.866 \pm .015$	$.867 \pm .011$
Stanford Dogs Ours	$.862 \pm .008$	$.867 \pm .007$	$.869 \pm .008$	$.871 \pm .008$
CIFAR-100 BP	$.914 \pm .007$	$.912 \pm .008$	$.913 \pm .007$	$.912 \pm .007$
CIFAR-100 Ours	$.895 \pm .009$	$.895 \pm .008$	$.896 \pm .010$	$.895 \pm .008$

4.3 Accuracy

We next ask how much accuracy is retained under the substantially cheaper training procedure established above. Table 1 reports test accuracy after the same number of training epochs for both methods, where each epoch of our method costs about half a BP epoch (Appendix C).

Our method stays close to BP on EuroSAT and CIFAR-100, and surpasses it on Dogs from 20 epochs onward. After 30 epochs, it reaches 0.978 versus 0.989 on EuroSAT, 0.895 versus 0.912 on CIFAR-100, and 0.871 versus 0.867 on Dogs. Thus, at about half the training compute per epoch, our method preserves most of BP's accuracy on all datasets, and can even outperform it. The accuracy cost is therefore modest relative to the reduction in backward computation achieved by our approach.

With BitFit, the comparison with BP is even more favorable (Appendix D).

5 Conclusions

In this work, we present a different view of gradient-based learning: rather than repeatedly recomputing network-wide derivatives through backpropagation, we ask when this information can be compressed once and reused throughout training. By exploiting Jacobian laziness and low-rank structure, we obtain parameter updates whose cost is nearly independent of sequence length and bring the overall training step close to inference cost. We further provide scalable diagnostics for verifying when this structure is present and show how it is preserved under parameter-efficient fine-tuning. Empirically, the resulting method substantially reduces backward computation while retaining strong accuracy on realistic models and datasets. More broadly, our results demonstrate that gradient-based training can be organized around reusable derivative structure, offering an alternative computational view of learning.

AI USE STATEMENT

We used generative AI tools to help draft parts of the paper and improve the writing, mainly for clarity, wording, grammar, and organization. We did not use AI tools for literature search, research ideation or execution, synthetic data generation, or mathematical proofs. All AI-assisted text was reviewed and edited by the authors, and we take full responsibility for the final content of the paper.

REPRODUCIBILITY STATEMENT

The training procedure is given in Algorithm 1 (Section 3.3), and the reference-set selection in Algorithm 2 (Appendix B.1). All theoretical results are stated with their assumptions and proved in full in Appendices F, G and H. All datasets are public, and all backbones are public pretrained checkpoints (Cherti et al., 2023; Sun et al., 2023; Wightman, 2019). Appendix A specifies the datasets, architectures, adapters and training protocol, and Table 2 lists every fine-tuning hyperparameter. The timing and memory measurements, including the hardware, precision and batch sizes, are described in Appendix C.

REFERENCES

Alessandro Achille, Aditya Golatkar, Avinash Ravichandran, Marzia Polito, and Stefano Soatto. Lqf: Linear quadratic fine-tuning. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15724–15734. IEEE, 2021.

Atılım Güneş Baydin, Barak A Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*, 2022.

Akhilan Boopathy and Ila Fiete. How to train your wide neural network without backprop: An input-weight alignment perspective. In *International Conference on Machine Learning*, pp. 2178–2205. PMLR, 2022.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Zixiang Chen, Greg Yang, Qingyue Zhao, and Quanquan Gu. Global convergence and rich feature learning in l-layer infinite-width neural networks under μp parametrization, 2025. URL https://arxiv.org/abs/2503.09565.

Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. Reproducible scaling laws for contrastive language-image learning. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2829. IEEE, 2023.

Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Advances in neural information processing systems*, 32, 2019.

Arif Kerem Dayi and Sitan Chen. Low-rank fine-tuning lies between lazy training and feature learning. In Nika Haghtalab and Ankur Moitra (eds.), *Proceedings of Thirty Eighth Conference on Learning Theory*, volume 291 of *Proceedings of Machine Learning Research*, pp. 1415–1471. PMLR, 30 Jun–04 Jul 2025. URL https://proceedings.mlr.press/v291/dayi25a.html.

Clementine Domine, Nicolas Anguita, Alexandra M Proca, Lukas Braun, Daniel Kunin, Pedro Mediano, and Andrew Saxe. From lazy to rich: Exact learning dynamics in deep linear networks. In *International Conference on Learning Representations*, volume 2025, pp. 102485–102536, 2025.

Mario Geiger, Stefano Spigler, Arthur Jacot, and Matthieu Wyart. Disentangling feature and lazy training in deep neural networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2020 (11):113301, 2020.

Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 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. *arXiv preprint arXiv:2106.09685*, 2021.

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

Ajay Jaiswal, Yifan Wang, Lu Yin, Shiwei Liu, Runjin Chen, Jiawei Zhao, Ananth Grama, Yuandong Tian, and Zhangyang Wang. From low rank gradient subspace stabilization to low-rank weights: Observations, theories, and applications. *arXiv preprint arXiv:2407.11239*, 2024.

Uijeong Jang, Jason D Lee, and Ernest K Ryu. Lora training in the ntk regime has no spurious local minima. *arXiv preprint arXiv:2402.11867*, 2024.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.

Junsu Kim, Jaeyeon Kim, and Ernest K Ryu. Lora training provably converges to a low-rank global minimum or it fails loudly (but it probably won't fail). *arXiv preprint arXiv:2502.09376*, 2025.

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

Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of research of the National Bureau of Standards*, 45(4): 255–282, 1950.

Brett W Larsen, Stanislav Fort, Nic Becker, and Surya Ganguli. How many degrees of freedom do we need to train deep networks: a loss landscape perspective. *arXiv preprint arXiv:2107.05802*, 2021.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

Chaoyue Liu, Libin Zhu, and Misha Belkin. On the linearity of large non-linear models: when and why the tangent kernel is constant. *Advances in Neural Information Processing Systems*, 33: 15954–15964, 2020.

Tian Yu Liu, Aditya Golatkar, and Stefano Soatto. Tangent transformers for composition, privacy and removal. In *International Conference on Learning Representations*, volume 2024, pp. 28508–28525, 2024.

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

Jamie Mahowald, Brian Bell, Alex Ho, and Michael Geyer. Efficient analysis of the distilled neural tangent kernel. *arXiv preprint arXiv:2602.11320*, 2026.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36:53038–53075, 2023a.

Sadhika Malladi, Alexander Wettig, Dingli Yu, Danqi Chen, and Sanjeev Arora. A kernel-based view of language model fine-tuning. In *International Conference on Machine Learning*, pp. 23610–23641. PMLR, 2023b.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Mohamad Amin Mohamadi, Wonho Bae, and Danica J Sutherland. A fast, well-founded approximation to the empirical neural tangent kernel. In *International conference on machine learning*, pp. 25061–25081. PMLR, 2023.

Roman Novak, Jascha Sohl-Dickstein, and Samuel S Schoenholz. Fast finite width neural tangent kernel. In *International Conference on Machine Learning*, pp. 17018–17044. PMLR, 2022.

Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. Task arithmetic in the tangent space: Improved editing of pre-trained models. *Advances in Neural Information Processing Systems*, 36:66727–66754, 2023.

Samet Oymak, Zalan Fabian, Mingchen Li, and Mahdi Soltanolkotabi. Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian. *arXiv preprint arXiv:1906.05392*, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Tian Qin and Wei-Min Huang. Backpropagation-free trunk training via the split forward gradients. *arXiv preprint arXiv:2607.16612*, 2026.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Adityanarayanan Radhakrishnan, Daniel Beaglehole, Parthe Pandit, and Mikhail Belkin. Mechanism for feature learning in neural networks and backpropagation-free machine learning models. *Science*, 383(6690):1461–1467, 2024.

Yehonathan Refael, Jonathan Svirsky, Boris Shustin, Wasim Huleihel, and Ofir Lindenbaum. Adarankgrad: Adaptive gradient rank and moments for memory-efficient llms training and fine-tuning. In *International Conference on Learning Representations*, volume 2025, pp. 102591–102612, 2025.

Thomas Robert, Mher Safaryan, Ionut-Vlad Modoranu, and Dan Alistarh. Ldadam: Adaptive optimization from low-dimensional gradient statistics. In *International Conference on Learning Representations*, volume 2025, pp. 4778–4814, 2025.

Quan Sun, Yuxin Fang, Ledell Wu, Xinlong Wang, and Yue Cao. Eva-clip: Improved training techniques for clip at scale, 2023. URL https://arxiv.org/abs/2303.15389.

Anke Tang, Li Shen, Yong Luo, Yibing Zhan, Han Hu, Bo Du, Yixin Chen, and Dacheng Tao. Parameter-efficient multi-task model fusion with partial linearization. In *International Conference on Learning Representations*, volume 2024, pp. 21148–21178, 2024.

Zhenfeng Tu, Santiago Aranguri, and Arthur Jacot. Mixed dynamics in linear networks: Unifying the lazy and active regimes. *Advances in Neural Information Processing Systems*, 37:106059–106104, 2024.

Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

Blake Woodworth, Suriya Gunasekar, Jason D Lee, Edward Moroshko, Pedro Savarese, Itay Golan, Daniel Soudry, and Nathan Srebro. Kernel and rich regimes in overparametrized models. In *Conference on Learning Theory*, pp. 3635–3673. PMLR, 2020.

Linhan Xia, Rui Liu, Zhaofeng Zhang, Yihao Wang, Binrui Shen, and Shengxin Zhu. Loca: Forward-only llm tuning after one-shot calibration with local credit assignment. *arXiv preprint arXiv:2608.03020*, 2026.

Zixiong Yu, Songtao Tian, and Guhan Chen. Divergence of neural tangent kernel in classification problems. In *International Conference on Learning Representations*, volume 2025, pp. 46521–46539, 2025.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–9, 2022.

Amir Zandieh, Insu Han, Haim Avron, Neta Shoham, Chaewon Kim, and Jinwoo Shin. Scaling neural tangent kernels via sketching and random features. *Advances in Neural Information Processing Systems*, 34:1062–1073, 2021.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. Lora-fa: Efficient and effective low rank representation fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

A Datasets, Architectures, and Training Protocols

A.1 Datasets

We fine-tune on three transfer tasks chosen to span the range over which a CLIP backbone is useful. EuroSAT (Helber et al., 2019) is Sentinel-2 satellite imagery in ten land-cover classes, and is the easy end: a linear probe on frozen features is already strong, so it measures whether our update loses anything when little has to be learned. Stanford Dogs (Khosla et al., 2011), with 120 breeds, is a fine-grained task and the hardest of the three, and CIFAR-100 (Krizhevsky et al., 2009) sits in between.

A.2 Architectures

The width ladder is the LAION-2B CLIP family of vision transformers (Cherti et al., 2023): ViT-B/16, ViT-L/14, ViT-H/14, ViT-g/14 and ViT-G/14, with embedding widths 768, 1024, 1280, 1408 and 1664. Holding the pretraining corpus and objective fixed across the ladder means a trend in the laziness diagnostic reflects width rather than a change of recipe. ViT-G/14 is the default backbone for the transformer accuracy experiments. The timing and memory measurements of Section 4.2 additionally use EVA02-E (Sun et al., 2023), width 1792.

To test whether the analysis is specific to attention, we repeat the ladder on three ConvNeXt CLIP models (Liu et al., 2022) from the same pretraining corpus, with 88M, 199M and 843M parameters, final-stage widths 1024, 1536 and 3072, and a 256-pixel input. ConvNeXt-L's checkpoint carries an MLP projection between the pooled features and the classifier; we freeze it, so that across every backbone the trainable set is exactly the adapter plus a linear readout.

A.3 Adapters

LoRA. Our method uses fixed-A LoRA, $W_0 + sBA$ with rank 2, inserted at the fused query–key–value projection of every attention block (transformers) or at the first linear layer of every block MLP (ConvNeXt). In the ConvNeXt-XXL accuracy experiments, our method uses only the last eight blocks: measured as in Table 11, ρ falls from 0.17 with every block to 0.019, 0.015 and 0.013 with the last 16, 8 and 3, below $1/\sqrt{\text{width}} = 0.018$ from eight blocks on; backpropagation keeps every block. A is frozen at its random initialization and B starts at zero, so the adapter is linear in its trainable parameters. For backpropagation we use the more expressive two-factor LoRA, the standard form of Hu et al. (2021), in which A and B are both trained. Since it contains fixed-A LoRA as the special case of a constant A, this is the stronger of the two baselines one could choose.

BitFit. BitFit trains the biases of the last k blocks. We tried $k \in \{2, 4, 8\}$ and all blocks, and each comparison reports the depth that works best for each method, chosen separately for ours and for backpropagation.

Readout. In both cases the linear readout is trained as well. In our method its gradient is computed in closed form rather than replayed, which is exact because the readout is linear in its own weights. For BitFit we first warm-start the readout and then freeze its weight, keeping the bias trainable; this removes the bilinear head curvature that would otherwise dominate the measured drift.

A.4 Protocol

Our method uses a reference set of $R{=}5000$ columns per output unit, chosen by accumulated gradient magnitude (Appendix B.1), and retains rank $r{=}200$. The projection P_j is accumulated over a class-balanced subset of $n_{\text{pre}} = \max(2000,\, 20d_y)$ training images and rescaled by n/n_{pre}. Both methods are trained with AdamW for 30 epochs, and each is given its own learning rate, selected once from a small grid and then held fixed across runs. Backpropagation runs in full fp32, the more precise setting, and our method in TF32; running our method in fp32 as well gives identical accuracy (.9780 on EuroSAT and .8950 on CIFAR-100), so the difference in precision does not favor it. Table 1 reports test accuracy at fixed epochs, as mean and standard deviation over ten runs; the ablations in the appendix report the best accuracy over the epoch checkpoints of a single run. Table 2 lists the remaining settings.

Experiments ran on a mix of NVIDIA A100, H200 and B200 GPUs; the timing and memory measurements of Appendix C were all taken on a single B200 so that the curves are comparable.

A.5 FINE-TUNING HYPERPARAMETERS

Table 2 collects the settings used for fine-tuning. Unless a table or figure states otherwise, every experiment in the paper uses them.

Table 2: **Fine-tuning hyperparameters.** Where the two methods differ, the entry for ours is given first.

Shared	
Optimizer	AdamW, β=$(0.9, 0.999)$, weight decay 0.01
Batch size	32
Epochs	30, evaluated every 5
Data augmentation	none
Readout initialization	truncated normal, std 0.02
Readout learning rate	the adapter's; on ConvNeXt also 10^{-3} for both, best kept
LoRA	
Rank, scaling	2, s=$\alpha/$rank=2, no dropout
Initialization	A Kaiming-uniform; B=0 / $B \sim \mathcal{N}(0, 10^{-4})$
Trained factors	B only / A and B
Learning rate grid	10^{-5}–4×10^{-4} / 10^{-4}–2×10^{-3}
BitFit	
Depth	last $k \in \{2, 4, 8\}$ blocks or all, best per method
Readout warm start	30 epochs at 10^{-3}, then weight frozen
Learning rate grid	10^{-6}–3×10^{-3}
Ours only	
Reference set	R=5000 per output unit, gradient-magnitude rule
Retained rank	r=200
Projection subset	$n_{\mathrm{pre}} = \max(2000,\ 20d_y)$, class-balanced
Precision	
Training	TF32 / fp32

Table 3 gives the learning rate, and for BitFit the depth, behind every accuracy table.

Table 3: **Learning rates and BitFit depths used.** k: BitFit on the last k blocks. On ConvNeXt-XXL our LoRA adapter is in the last eight blocks.

		Ours	BP
ViT-G, LoRA (Table 1)	EuroSAT	1.5×10^{-4}	2.5×10^{-4}
	Stanford Dogs	3×10^{-5}	10^{-4}
	CIFAR-100	5×10^{-5}	5×10^{-4}
ViT-G, BitFit (Table 10)	EuroSAT	3×10^{-5}, k=4	3×10^{-4}, all blocks
	Stanford Dogs	3×10^{-4}, k=4	3×10^{-4}, k=8
	CIFAR-100	10^{-4}, k=4	3×10^{-4}, all blocks
ConvNeXt-XXL, LoRA (Table 12)	EuroSAT	3×10^{-4}	2.5×10^{-4}
	Stanford Dogs	3×10^{-5}	2.5×10^{-4}
	CIFAR-100	10^{-4}	5×10^{-4}
ConvNeXt-XXL, BitFit (Table 12)	EuroSAT	10^{-4}, k=4	3×10^{-4}, all blocks
	Stanford Dogs	10^{-4}, k=4	3×10^{-4}, all blocks
	CIFAR-100	10^{-4}, k=4	3×10^{-4}, all blocks

B REFERENCE SET: SELECTION, SIZE, AND RANK

Our update is built from two approximations of the Jacobian: we keep only the R parameter directions that carry the most gradient energy for each output unit, and we retain only the top r singular directions within that set. Three choices are therefore open: whether the reference sets are per output unit or shared, how large R and r are, and which score fills the sets. The first two affect accuracy; the third does not.

B.1 THE SELECTION PROCEDURE

Algorithm 2 expands the selection step of Algorithm 1. One pass over the training set accumulates the absolute loss gradient into a separate bucket per class, and each output unit then takes the R coordinates of largest accumulated magnitude in its own bucket. Because the bucket is chosen by the label rather than by differentiating each logit, the pass costs one backward pass per image in total, not d_y of them; the price of going per unit is the d_y accumulators, not extra gradient computation.

Algorithm 2 Per-Unit Gradient-Magnitude Reference-Set Selection

Require: Model f, samples $\mathcal{B}$, reference set size R
Ensure: Reference sets $\mathcal{R}_1, \ldots, \mathcal{R}_{d_y}$ of R parameter indices each
1: Initialize accumulators $g_j \leftarrow \mathbf{0}$ for $j = 1, \ldots, d_y$
2: **for** each (x, y) in $\mathcal{B}$ **do**
3: Compute loss $L = \ell(f(x), y)$ and gradient $\nabla_\theta L$
4: Accumulate $g_y \leftarrow g_y + |\nabla_\theta L|$
5: **end for**
6: For each j, select the top-R parameters by magnitude in g_j as $\mathcal{R}_j$
7: **return** $\{\mathcal{R}_j\}_{j=1}^{d_y}$

Selection costs one backward pass per sample plus $\mathcal{O}(d_y m)$ memory for the accumulators and $\mathcal{O}(d_y R \log R)$ for sorting, and is independent of the rank r.

The three subsections that follow ask, in turn, whether the per-unit partition in line 6 is what matters, whether R and r matter, and whether the magnitude score in line 4 matters.

B.2 WHERE THE COORDINATES COME FROM

There are three natural ways to choose the R reference columns. *Per unit*, the choice of Algorithm 2, gives every output unit j its own set: the R parameters whose gradient magnitude, accumulated over the training images of class j, is largest. A task with d_y classes therefore has d_y sets. *Shared* uses a single set for all units, the R parameters with the largest score pooled over all classes. *Readout barred* is the shared set with the readout weights excluded from the candidates, so that all R columns come from the adapter.

The distinction matters because of the readout. Its weights have the largest gradients, so they fill most of a shared set, yet logit j has exactly zero derivative with respect to the readout row of any other class. In the slice of unit j, a shared set drawn mostly from the readout is therefore mostly identically zero columns, which leaves roughly R/d_y live ones and caps the slice at that rank. With R=5000 the cap is 500 at ten classes, comfortably above the retained rank r=200, but 50 at a hundred classes and 41 at 120, well below it. Barring the readout removes the dead columns, but still asks one subspace to serve every unit. Per-unit sets avoid both problems, and cost no extra backward passes, only the d_y accumulators of Algorithm 2.

Table 4 bears this out. Per-unit sets are the most accurate on all three tasks. At ten classes, where the cap is slack, per-unit sets lead the shared set by only .001, and all three choices lie within .0035 of each other. At a hundred and more classes, where it binds, per-unit sets lead the shared set by .0035 on both CIFAR-100 and Stanford Dogs, and the barred set by .005 and .009.

Table 4: **Test accuracy of our method for three ways of choosing the R=5000 reference columns, with r=200.** d_y is the number of output units.

	d_y	shared	readout barred	per unit
EuroSAT	10	.9775	.9750	**.9785**
Stanford Dogs	120	.8685	.8630	**.8720**
CIFAR-100	100	.8920	.8905	**.8955**

Table 5: **Test accuracy on Stanford Dogs as the reference-set size R and the retained rank r vary.**

	r=50	r=100	r=200	r=400
R=500	.8580	.8615	.8640	.8660
R=1000	.8620	.8655	.8670	.8655
R=2000	.8645	.8680	.8690	.8700
R=5000	.8660	.8690	**.8720**	.8695
R=10000	.8655	.8685	.8715	.8710

B.3 How large R and r are

Table 5 sweeps R over a twentyfold range and r over an eightfold range on Stanford Dogs, the task with the most output units and a Jacobian energy rank of $r_{0.95}$=366 (Section 4.1). In every row, accuracy increases with r up to r=200, and for $r \leq 200$ it increases with R up to R=5000. Past that point the gains stop: doubling R from 5000 to 10000 changes accuracy by at most .0015, and for $R \geq 1000$ doubling r to 400 changes it by at most .0025. The default R=5000, r=200 is the best cell of the grid, which spans 1.4 points in total, down to .8580 at the smallest setting.

The two knobs set the cost of the method: R fixes the size of each reference slice and of its eigenproblem, and r the size of the stored factors V_j and P_j. Near the default the grid is shallow, and every cell with $R \geq 2000$ and $r \geq 100$ is within .004 of the best, less than the run-to-run spread on this task in Table 1. A tighter budget can therefore take R=2000, r=100, which halves the stored factors, at little cost in accuracy, while larger values than the default only add cost.

B.4 Which score fills a per-unit set

Table 6: **Test accuracy for three rules that fill each per-unit reference set (R=5000, r=200).**

	gradient magnitude	column energy	random
EuroSAT	**.9785**	.9760	.9735
Stanford Dogs	**.8720**	.8710	.8695
CIFAR-100	**.8955**	.8950	.8950

Table 6 replaces the gradient-magnitude score of Algorithm 2 with two alternatives: the column energy of the Jacobian at initialization, and a uniform random draw among the eligible columns. Gradient magnitude is the most accurate rule on every task, and it is also the cheapest informative one: it needs one backward pass per image, whereas column energy needs the per-sample Jacobian, one backward pass per output unit.

The margins, however, are small, at most .005, and this follows from the low rank of the Jacobian. Each per-unit slice is dominated by a few directions, and Appendix H.1 shows that almost any sufficiently large sample of its columns spans them; the score only decides which of many adequate samples is used. The partition of Appendix B.2 is different in kind, since it decides which subspace each unit can see at all.

C Cost of an Update: Memory and Scaling

Section 4.2 reports the time a step costs beyond its forward pass. This appendix adds the memory it costs, the two ratios that summarize both, and the behaviour as the input grows.

C.1 What the factor of two undercounts

With the backbone frozen, a backpropagation step costs about two forward passes by operation count. The backward pass carries the error back to the adapters but forms no gradients for the frozen weights, so it does roughly the arithmetic of one more forward pass; the usual estimate of three forward passes per training step (Kaplan et al., 2020) counts those weight gradients. Our step is one forward pass plus an update whose cost does not depend on the input (Appendix C.3). The count therefore gives a factor of two over backpropagation, and of three over backpropagation with activation checkpointing (Chen et al., 2016), which runs the forward pass twice. This appendix measures what the count leaves out: the extra arithmetic of attention at long inputs, the time a backward FLOP takes compared with a forward FLOP, and memory. Table 7 sets the count against the measured time for every backbone.

Setup. All measurements use fixed-A LoRA of rank 2 with d_y=10 and r=200, on one GPU, in PyTorch (Paszke et al., 2019) without compilation. A backpropagation step is the forward pass, the backward pass and an AdamW step. Our step is a forward pass without gradient tracking, the update of Algorithm 1, and the same AdamW step. The batch is 32 on ViT-B and 8 on ViT-G and ConvNeXt-XXL. FLOPs are counted per operator with PyTorch's FLOP counter; by that count our step is one forward pass to within 1%. TensorFloat-32 is disabled except where a column says TF32, and bf16 means autocast mixed precision.

Table 7: **Cost of a training step relative to ours on one B200, counted in FLOPs and measured in time.** OOM: out of memory at this batch size.

	backpropagation / ours				checkpointed / ours			
	FLOPs	fp32	TF32	bf16	FLOPs	fp32	TF32	bf16
ViT-B, L=197	2.01	2.00	2.17	2.72	3.01	2.71	2.96	3.54
ViT-B, L=1765	2.26	2.36	2.26	2.81	3.26	3.10	3.20	3.77
ViT-G, L=257	2.02	1.99	2.18	2.47	3.02	2.66	2.92	3.30
ViT-G, L=2305	2.16	OOM	OOM	OOM	3.16	2.94	3.26	3.53
ConvNeXt-XXL, L=64	1.99	1.95	2.02	3.18	2.99	2.92	2.95	4.08
ConvNeXt-XXL, L=1024	1.99	2.34	1.99	3.65	2.99	3.32	2.94	4.55

The count grows with the input on transformers. On ConvNeXt-XXL, the backward-to-forward FLOP ratio remains 0.99 across resolutions. On transformers it starts at 1.01–1.02 and grows with the sequence length, to 1.26 at 1,765 tokens on ViT-B and 1.16 at 2,305 tokens on ViT-G. Attention multiplies two activations, so its backward pass needs the gradient with respect to both, where a frozen linear layer needs only the gradient with respect to its input. As attention takes a larger share of the arithmetic, the backward pass outgrows the forward pass: by count alone (the FLOPs columns of Table 7), a backpropagation step on ViT-B at 1,765 tokens is 2.26 of our steps and a checkpointed step 3.26.

Table 8: **Time of one backward FLOP relative to one forward FLOP on one B200.** A value of one means time follows the FLOP count.

	fp32	TF32	bf16
ViT-B, L=197	0.99	1.18	1.73
ViT-B, L=1765	1.08	1.00	1.44
ViT-G, L=257	0.98	1.20	1.49
ConvNeXt-XXL, L=64	0.97	1.10	2.34
ConvNeXt-XXL, L=1024	1.36	1.01	2.70

A backward FLOP takes longer than a forward FLOP. Dividing the time of the backward pass by the time of the forward pass, and that by their FLOP ratio, gives the time a backward FLOP takes relative to a forward FLOP (Table 8). In fp32 it is 0.97–1.08, and the time ratio of backpropagation follows its count.[1] In TF32 it is 1.00–1.20. In bf16 it is 1.44–1.73 on transformers and 2.34–2.70 on ConvNeXt-XXL. Mixed precision (Micikevicius et al., 2017), now usually in bf16 (Kalamkar et al., 2019), is how large models are trained on this hardware, and here it runs the forward pass 2.3 to 9.0 times faster than fp32; the backward pass gains less. The effect is not confined to short inputs, where the time spent launching kernels could account for part of a step: at 1,765 tokens on ViT-B and 1,024 positions on ConvNeXt-XXL, where a bf16 forward pass takes 305 and 236 ms, a backward FLOP takes 1.44 and 2.70 forward FLOPs.

Where the extra backward time goes. The measurements above show that, particularly in bf16, the backward pass takes more time than its FLOP count alone would suggest. To identify the source of this overhead, we profile the GPU kernels executed during the backward pass. The additional cost is concentrated in the matrix multiplications used to propagate gradients through the low-rank adapter factors. These operations read large activation matrices but produce very narrow outputs, and therefore achieve substantially lower GPU throughput than the wide matrix products of the forward pass. Our method avoids these per-layer gradient computations entirely, replacing them with multiplication by the stored P_j representation.

To distinguish this computational cost from CPU-side kernel-launch and scheduling overhead, we also compare GPU kernel time alone with the end-to-end wall-clock measurements. The ratios remain nearly unchanged: a step costs 2.52 forward passes in kernel time on ViT-G at 257 tokens and 3.31 and 3.66 on ConvNeXt-XXL at 64 and $1{,}024$ positions, compared with 2.47, 3.18, and 3.65 by wall clock. Thus, the excess backward time is already present in the GPU computation itself rather than arising primarily from kernel-launch overhead.

On ConvNeXt-XXL at $1{,}024$ positions, the profiled adapter-gradient matrix multiplications account for at least 41% of the step time while contributing under 0.2% of its FLOPs; the corresponding fractions are 53% at 64 positions and 27% on ViT-G. The largest such product takes 168 ms and reads about 1.6 GB of activations. Thus, a small fraction of the nominal FLOPs can dominate the backward runtime because these narrow products operate far below the throughput reached by the wide forward-pass matrix multiplications. Specialized kernels for these shapes could narrow this gap; the measurements here reflect the kernels dispatched in the current implementation for both methods.

Depthwise convolutions account for only 5–8% of the step, and changing the memory format to channels-last has no measurable effect. Thus, the convolution layout does not explain the bf16 backward overhead (Liu et al., 2022; Wightman, 2019).

This backward inefficiency is reflected directly in the end-to-end step cost (Table 7). In bf16, backpropagation costs 2.47–$3.65\times$ our step and checkpointed backpropagation 3.30–$4.55\times$, compared with FLOP-count ratios of 1.99–2.26 and 2.99–3.26, respectively. None of this additional backward cost enters our step. The two methods use the same forward computation, and enabling gradient tracking increases its cost by at most 0.6%. Everything in our step beyond the forward pass—the update and optimizer step—takes only 0.2–1.2 ms in bf16, at most 3.8% of the full step and under 2% on every transformer. Backpropagation runs on the library's tuned kernels, while our update is plain, unfused PyTorch. Our advantage therefore comes without any engineering on our side: the update already takes about a millisecond, and the FLOP counts of Table 7 do not depend on implementation at all.

We report these measurements on the B200, which is also used for the other timing and memory experiments in the paper. The wall-clock ratios therefore describe the realized savings on this hardware, while the FLOP ratios are hardware independent. The precision comparison concerns computational cost only: the accuracy experiments use TF32 for our method and fp32 for backpropagation (Appendix A.4), and we make no claim about accuracy in bf16.

[1] The exception is ConvNeXt-XXL at 1,024 positions (1.36). In the fp32 sweep of Figure 5, the forward pass there takes 24% less time per position than at 576 positions, while the backward pass takes the same time per position; it is the forward pass that departs from the trend.

Table 9: **Largest batch that fits on one 180 GB B200, in bf16 except the last column.** $\geq$: stopped at a PyTorch tensor-size limit before memory ran out.

	ours	backprop.	checkpointed	ours / backprop.	ours / backprop. (fp32)
ViT-B, L=197	19,059	2,439	10,433	7.8×	10.3×
ViT-B, L=1765	488	70	275	7.0×	7.4×
ViT-G, L=257	7,188	215	1,658	33×	42×
ViT-G, L=2305	198	7	84	28×	29×
ConvNeXt-XXL, L=64	$\geq$ 1,365	612	$\geq$ 1,365	$\geq$ 2.2×	$\geq$ 3.7×
ConvNeXt-XXL, L=1024	$\geq$ 85	38	$\geq$ 85	$\geq$ 2.2×	$\geq$ 3.9×

Memory. Backpropagation has to keep each layer's activations from the forward pass until the backward pass has used them, whereas a forward pass alone releases them as soon as the next layer has read them. Our step keeps no activations, only the stored representation (V_j, P_j). Table 9 gives the largest batch that fits on a 180 GB B200. In bf16, our step fits 7.0–7.8× the batch of backpropagation on ViT-B and 28–33× on ViT-G, and more in fp32; at 2,305 tokens on ViT-G, backpropagation fits 7 images and our step 198. Checkpointed backpropagation recovers part of this, at the cost of the second forward pass counted above, and still fits 1.8–4.3× fewer images than our step on transformers. On ConvNeXt-XXL our step fits at least 2.2× the batch of backpropagation in bf16 and 3.7× in fp32. There our step and the checkpointed step both stopped at a 32-bit tensor-indexing limit before running out of memory, so those entries are lower bounds. P_j grows linearly with d_y, so with more classes the batch our step fits is smaller than here.

C.2 Activation checkpointing

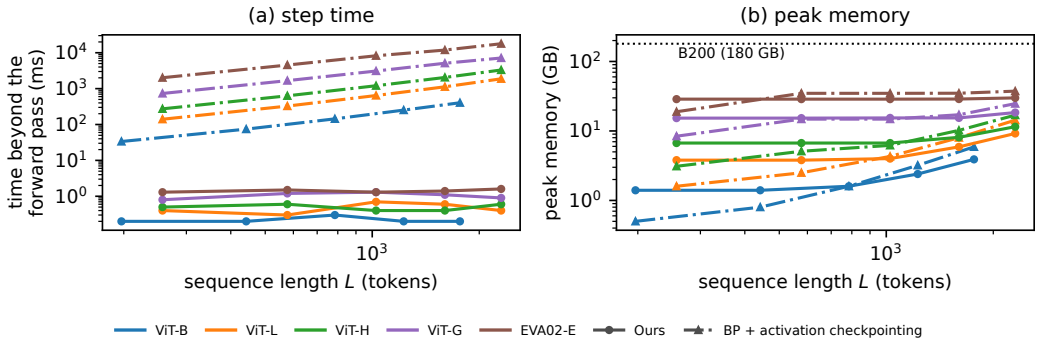

Figure 4: **Our update against backpropagation with activation checkpointing.** (a) Time beyond the forward pass, (b) peak memory.

Checkpointing is the standard answer to the memory growth of Figure 3(b): discard the activations and recompute them during the backward pass. Figure 4 compares it against our update on both axes at once. It does flatten the memory curve, to at most 37.4 GB over the whole sweep, and it keeps the largest backbones inside the device where plain backpropagation does not fit. But it pays for that in time: a checkpointed step costs 2.83× our step in the median, against 2.05× for plain backpropagation, because the forward pass is walked twice. Our update needs neither the stored activations nor the second forward pass, so it is the cheaper of the two on time at every point measured. On peak memory at a fixed batch it is cheaper from 1,601 tokens on for every backbone and dearer at short inputs, while in the largest batch that fits (Table 9) it still fits 1.8–4.3× more images than checkpointing.

C.3 Independence from sequence length and resolution

Section 3.3 claims that our update costs $\mathcal{O}(d_y mr)$, with no dependence on how many tokens the model reads, while a backward pass grows with the input just as the forward pass does. The trans-

former sweeps above confirm it. From L=197 to L=2305, our step costs no more than its forward pass, up to measurement noise, while the backward pass grows from 20 ms to seconds.

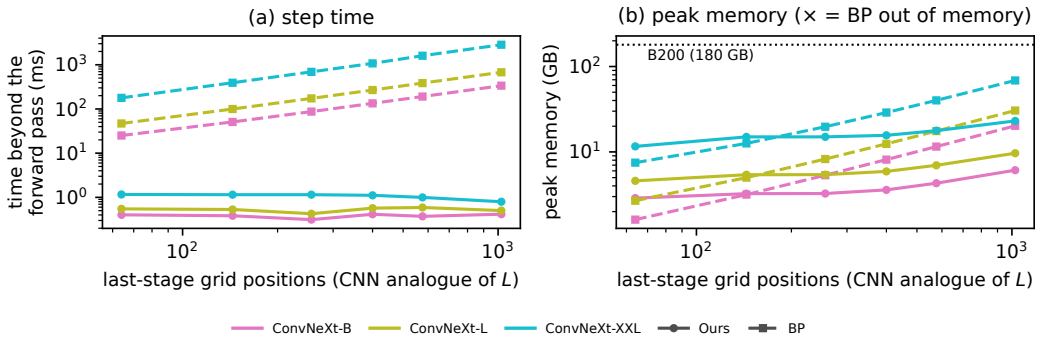

Figure 5: **Figure 3 on ConvNeXt CLIP.** (a) Time beyond the forward pass and (b) peak memory against the final-stage grid (input 256 to 1024 pixels, batch 8, fp32).

For a CNN the spatial grid takes the place of the token count. Figures 5 and 6 repeat the sweep on ConvNeXt at input resolutions from 256 to 1024 pixels, a final-stage grid of 64 to 1024 positions. What our update adds to the forward pass stays between 0.3 and 1.2 ms at every resolution and for all three widths; on ConvNeXt-XXL it is 0.8 ms at the largest grid. What backpropagation adds grows with the grid instead, from 178 ms to 2.8 s on ConvNeXt-XXL, and to 4.9 s with checkpointing. A backpropagation step therefore costs 1.94–2.34 of our steps and a checkpointed step 2.92–3.32, with the largest gap at the largest grid.

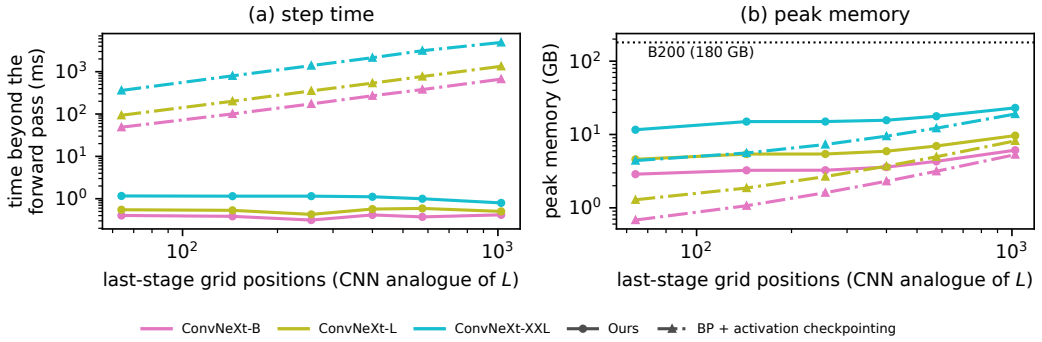

Figure 6: **Figure 4 on ConvNeXt CLIP.** Same sweep as Figure 5.

Memory splits the same way. Every method has to run the forward pass, and its footprint grows with the input. Beyond it, our step holds only the stored (V_j, P_j), whose size does not depend on the resolution: on ConvNeXt-XXL it adds 4 to 8 GB at every grid. Backpropagation adds its stored activations, which grow with the grid from 3.4 to 49.7 GB. ConvNeXt activations are small, so at the lowest resolution backpropagation still needs less memory than we do, but it overtakes us at about 150 positions, and at the largest grid it needs 68.7 GB against our 23.1 GB, three times as much (Figure 5b). Checkpointing brings backpropagation down to the footprint of the forward pass, just below ours (Figure 6), but only by running the forward pass twice, which makes each step about three times as costly as ours. Our update keeps its extra memory fixed without paying for it in time.

D BitFit on ViT-G

The accuracy experiments of Section 4.3 use LoRA. BitFit is the other parameterization our theory covers, and by Theorem G.7 it adds no curvature of its own, so replaying a fixed Jacobian should suit it well. Table 10 compares the two methods on ViT-G, each at its best depth and learning

Table 10: **Test accuracy of BitFit on ViT-G.**

	Ours	BP
EuroSAT	.9790	**.9825**
Stanford Dogs	**.8465**	.8455
CIFAR-100	.9040	**.9065**

rate. Without repeated backpropagation, our update closely matches backpropagation: it is ahead on Stanford Dogs and within 0.35 points on EuroSAT and CIFAR-100, gaps several times smaller than with LoRA (Table 1). It does so while training fewer parameters: our best setting updates the biases of the last four blocks, while backpropagation reaches its best with eight on Stanford Dogs and all of them on the other two tasks (Table 3).

E Beyond Transformers: Convolutional Backbones

Every accuracy result so far is measured on vision transformers; only the time and memory measurements of Appendix C include a convolutional backbone. Nothing in the argument of Sections 3.2 and 3.4 depends on attention: the diagnostic requires a Jacobian and a Hessian–vector product, and the inheritance statements require only the chain rule through the adapter. This appendix tests whether the empirical claims hold on ConvNeXt CLIP, which shares its pretraining corpus with the transformer ladder.

E.1 The assumptions

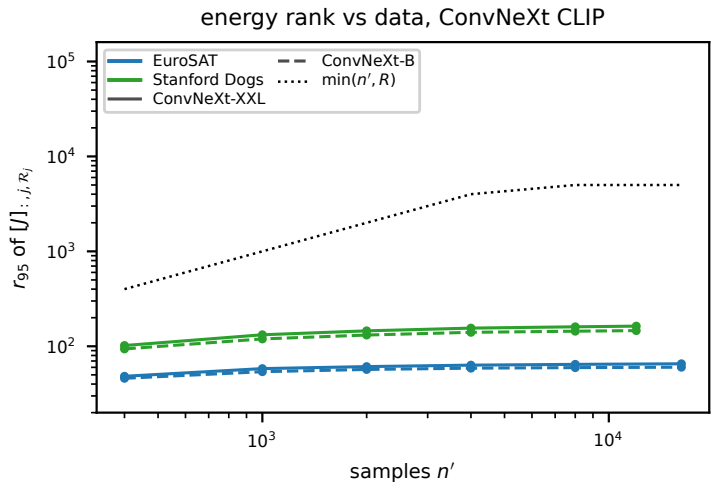

Figure 7: **Energy rank vs data on ConvNeXt CLIP.** As Figure 2(a), for ConvNeXt-B (dashed) and ConvNeXt-XXL (solid).

Table 11: **Laziness on ConvNeXt CLIP.** The diagnostic ρ at initialization and the relative adapter-Jacobian drift realized during training.

	fixed-A LoRA		two-factor LoRA		BitFit	
	ρ	drift	ρ	drift	ρ	drift
ConvNeXt-B	0.51	0.60	0.54	0.64	0.55	1.70
ConvNeXt-XXL	0.17	0.29	0.22	0.47	0.35	1.17

Table 11 repeats the verification of Section 3.2 on two ConvNeXt widths. The pattern is similar to that observed on transformers. The reference slices are low rank: as on transformers, the energy

rank levels off far below the maximal rank as data grow, at 60 to 65 on EuroSAT and 146 to 163 on Stanford Dogs with the full training set. (Figure 7). Moreover, ρ, measured once at initialization, predicts how far the Jacobian will move during training. On both backbones it orders the three parameterizations in the same way as the drift: fixed-A LoRA is the laziest, followed by two-factor LoRA and then BitFit. It also tracks width. From ConvNeXt-B to ConvNeXt-XXL, ρ falls for every parameterization, by a factor of 1.6 to 3, and the drift falls with it. Across all six cells, the rank correlation between ρ and the drift is 0.83, compared with 0.82 for the 15 transformer cells in Section 4.1.

This is what makes the diagnostic useful in practice. It takes a few minutes and requires no training, whereas the drift it predicts is only visible after 30 epochs. It also applies to an architecture without attention. One can therefore check whether our update applies to a new backbone before committing to it.

E.2 Accuracy

Table 12: **Test accuracy on ConvNeXt-XXL.**

	BitFit		LoRA	
	Ours	BP	Ours	BP
EuroSAT	.9790	.9815	.9780	.9895
Stanford Dogs	**.8225**	.8195	.8485	.8620
CIFAR-100	.8865	.8945	.8920	.9200

Table 12 compares the two methods on ConvNeXt-XXL. On this backbone, a step of our method costs about half a backpropagation step, and under a third in mixed precision (Table 7, Appendix C). At that cost, BitFit with our update closely matches backpropagation: it beats it on Stanford Dogs and stays within a point on the other two tasks. With LoRA, the baseline has an advantage beyond compute, because it trains the more expressive two-factor adapter; our update still comes within 1.4 points of it on EuroSAT and Stanford Dogs, and 2.8 on CIFAR-100.

F Laziness Diagnostic

This appendix shows why a small ρ means a slowly changing Jacobian, and how we compute ρ in practice.

From curvature to drift. Stack the outputs of all n training examples, so that $J_t \in \mathbb{R}^{nd_y \times m}$ is the Jacobian at w_t, and let $a_t \in \mathbb{R}^{nd_y}$ collect the output gradients $[a_t]_i = \nabla_f \ell(y^{(i)}, f(w_t; x^{(i)}))$. A full-batch gradient step is then $\Delta w_t = -\eta J_t^\top a_t$, where for the average loss η is the learning rate divided by n. Write $H_t = D_w J(w_t)$ for the derivative of the Jacobian, a map from a parameter direction v to the matrix $H_t[v]$, and $\|H_t\| = \sup_{\|v\|=1} \|H_t[v]\|$ for its norm. Row r of $H_t[v]$ is $\nabla_w^2 f_r(w_t)\, v$, the Hessian of output r applied to v, so for a single output $\|H_t\|$ is simply the spectral norm of its Hessian. A first-order expansion gives $J_{t+1} - J_t = H_t[\Delta w_t] + O(\|\Delta w_t\|^2)$, and since $\|\Delta w_t\| \le \eta \|J_t\|\, \|a_t\|$,

$$\frac{\|J_{t+1} - J_t\|}{\|J_t\|} \lesssim \frac{\eta \|H_t\|\, \|J_t\|\, \|a_t\|}{\|J_t\|} = \left(\eta \|J_t\|^2\right) \kappa_t, \qquad \kappa_t := \frac{\|H_t\|\, \|a_t\|}{\|J_t\|^2}. \tag{4}$$

The step-size factor. The bound (4) has two factors: κ_t, which depends on the model, and $\eta\|J_t\|^2$, the step size in the units of the linearized model. The second cannot be large if training is to make progress. For the square loss, one step of the linearized model multiplies the residual by $I - \eta J_t J_t^\top$, so the residual shrinks only if $\eta\|J_t\|^2 < 2$. For softmax cross-entropy, which we use, the Hessian of the loss with respect to the logits, $\mathrm{diag}(p) - pp^\top$ with p the predicted class probabilities, has eigenvalues at most $\frac{1}{2}$ by Gershgorin's theorem, so a step decreases the linearized loss whenever $\eta\|J_t\|^2 < 4$. With either loss, learning rates in this range keep $\eta\|J_t\|^2$ below a small constant, and the drift per step is governed by κ_t, the curvature-to-gradient ratio that Chizat et al. (2019) use to define lazy training.

Removing the loss. κ_t still depends on the loss through $\|a_t\|$, which changes as training progresses. Writing $\kappa_t = \rho_t \|a_t\|$ isolates the part that depends on the model alone, $\rho_t = \|H_t\|/\|J_t\|^2$. For cross-entropy the loss factor is bounded: the output gradient of example i is $p_i - e_{y^{(i)}}$, where p_i is the vector of predicted class probabilities and $e_{y^{(i)}}$ the one-hot vector of its label, and $\|p_i - e_{y^{(i)}}\|^2 = \|p_i\|^2 - 2p_{i,y^{(i)}} + 1 \le 2$ since $\|p_i\|^2 \le 1$. Hence, for a single example, which is how we evaluate the diagnostic below, $\kappa_t \le \sqrt{2}\,\rho_t$, and a small ρ bounds the drift at any value of the loss.

What we compute. We evaluate ρ one input x and one output unit j at a time. For a single pair the Jacobian is the gradient $g = \nabla_w f_j(x)$ and its derivative is the Hessian $\nabla_w^2 f_j(x)$, so

$$\rho_j(x) = \frac{\|\nabla_w^2 f_j(x)\|}{\|\nabla_w f_j(x)\|^2}$$

is the definition above applied to that pair: the denominator is the squared norm of one row of the Jacobian, and the numerator the norm of the Hessian of the same output at the same input. We take eight inputs and five output units and report the median of the forty values. The Hessian is symmetric, so its norm is its largest eigenvalue in absolute value, which we find with 25 Lanczos iterations (Lanczos, 1950). Lanczos only needs Hessian–vector products, and we form each from two ordinary gradients,

$$\nabla_w^2 f_j(x)\, v \approx \frac{\nabla_w f_j(w + \epsilon v; x) - \nabla_w f_j(w - \epsilon v; x)}{2\epsilon}, \qquad \epsilon = 10^{-3}.$$

Nothing larger than a gradient is ever stored. On ViT-B and ViT-G the finite differences give the same ρ as exact second-order autodiff to three decimals (0.073 and 0.049), and the whole measurement takes minutes.

Width. For wide networks the Hessian norm shrinks roughly as $\mathrm{width}^{-1/2}$ while the Jacobian stays of order one (Liu et al., 2020), so ρ should fall at the same rate. We draw this rate as a reference line in Figure 2(b); it is a check on the trend across widths, not a threshold for any single model.

G Laziness Inheritance under PEFT: Statements and Proofs

This appendix contains the full statements and proofs behind Section 3.4. Throughout, $\|\cdot\|$ is the spectral norm, $\|\cdot\|_F$ the Frobenius norm, and $\sigma_{\min}^{+}(\cdot)$ the smallest *nonzero* singular value. A network $f(\theta)$ is lazy when its Jacobian $J(\theta) = \partial f/\partial\theta$ barely changes along training; we quantify this by the relative drift $\Delta := \sup_t \|J_t - J_0\|/\|J_0\|$. When the Jacobian at initialization is rank–deficient, as it is for LoRA at the standard initialization, the relevant baseline is $\sigma_{\min}^{+}(J_0)$ rather than $\|J_0\|$. A replayed factorization must remain accurate in every direction it encodes, so the drift must be small relative to the *weakest* response. The two baselines agree up to the condition number for a well-conditioned Jacobian. We specify below which baseline each bound uses.

G.1 Low-rank adaptation

Consider a weight matrix $W_0 \in \mathbb{R}^{d\times k}$ adapted as $W = W_0 + s\,BA$ with $B \in \mathbb{R}^{d\times\rho}$, $A \in \mathbb{R}^{\rho\times k}$ and scale $s = \alpha/\rho$ (Hu et al., 2021). Denote $\phi = (A, B)$ for the trainable parameters, $J_\phi = \partial f/\partial\phi$ for the LoRA Jacobian and $J_W = \partial f/\partial W$ for the base-weight Jacobian. By the chain rule J_ϕ factors through J_W:

$$J_\phi = J_W \cdot \Pi, \qquad \Pi := \frac{\partial W}{\partial \phi}, \qquad \Pi[(dA, dB)] = s\,(B\,dA + dB\,A). \tag{5}$$

The key observation is that Π depends on (A, B): as the factors move, Π moves too. Subtracting the initial Jacobian and introducing a cross term gives the exact decomposition

$$J_{\phi,t} - J_{\phi,0} \;=\; \underbrace{\big(J_{W,t} - J_{W,0}\big)\,\Pi_t}_{\text{(A) base drift}} \;+\; \underbrace{J_{W,0}\,\big(\Pi_t - \Pi_0\big)}_{\text{(B) factor-map drift}}. \tag{6}$$

Term (A) vanishes if the base model is lazy. Term (B) is present *even if the base Jacobian never moves*, purely because Π_t varies with the factors in $B_t A_t$. Since Π is linear in (A, B) with coefficient s, its change is controlled by how far the factors travel: $\|\Pi_t - \Pi_0\| \le s\,(\|A_t - A_0\| + \|B_t - B_0\|) \le s\,D$, where D denotes the distance the factors travel, so $\|\text{Term (B)}\| \lesssim s\,\|J_{W,0}\|\,D$.

The baseline. At the standard initialization $B_0 = 0$, $\Pi_0[(dA, dB)] = s\,dB\,A_0$, so the dA-block lies in $\ker \Pi_0$ and the smallest nonzero singular value of $J_{\phi,0}$ is attained on the dB-block alone:

$$\begin{aligned}\sigma^{+}_{\min}(J_{\phi,0}) \;&=\; s \min_{dB \neq 0} \frac{\|J_{W,0}(dB\,A_0)\|}{\|dB\|_F} \\ &=\; s \min_{dB \neq 0} \underbrace{\frac{\|J_{W,0} M\|}{\|M\|_F}}_{\ge \gamma} \times \underbrace{\frac{\|dB\,A_0\|_F}{\|dB\|_F}}_{\ge \sigma_{\min}(A_0)}, \\ &\qquad M := dB\,A_0 \in \operatorname{span}(A_0).\end{aligned}$$

The first factor is bounded below by the smallest gain of $J_{W,0}$ over the adapter subspace,

$$\gamma \;:=\; \min_{0 \neq M \in \operatorname{span}(A_0)} \frac{\|J_{W,0} M\|}{\|M\|_F} \qquad \text{(the alignment)}, \tag{7}$$

and the second by $\sigma_{\min}(A_0) > 0$ since A_0 has full row rank. Hence,

$$\sigma^{+}_{\min}(J_{\phi,0}) \;\ge\; s\,\sigma_{\min}(A_0)\,\gamma, \tag{8}$$

with equality when the two minimizing directions coincide (in particular, when A_0 has orthonormal rows). The alignment γ is large when the adapter subspace consists entirely of directions to which the base Jacobian responds strongly. It is small when the subspace contains at least one direction to which the base Jacobian responds weakly. Dividing Term (B) by the baseline, cancels s and gives the relative drift contributed by the product:

$$\Delta^{(B)} \;\approx\; \frac{\|J_{W,0}\|\,D}{\sigma_{\min}(A_0)\,\gamma}. \tag{9}$$

Proposition G.1 (Sufficient condition for the LoRA Jacobian to stay frozen)**.** *Suppose the base model is lazy, so Term (A) in (6) is negligible. Then the two-factor LoRA Jacobian remains frozen ($\Delta \ll 1$) if the factor-map drift (9) is small, i.e.*

$$\|J_{W,0}\|\,D \;\ll\; \sigma_{\min}(A_0)\,\gamma. \tag{10}$$

The converse fails: every step leading to (10) is tight up to a constant except the bound $\|J_{W,0} M\| \le \|J_{W,0}\|\,\|M\|_F$, which bounds Term (B) using the largest gain of $J_{W,0}$ over all directions, whereas the drift depends only on its gain over the range of $\Pi_t - \Pi_0$. Condition (10) therefore certifies laziness, but its violation does not rule out laziness; Proposition G.2 establishes the loss of laziness by computing Δ directly.

Reading the condition: the left side grows with the base-Jacobian scale and with the distance the factors must travel to fit the task (D is dominated by $\|B_t\|$ since $B_0 = 0$); the right side is the frozen factor's scale multiplied by the alignment. In over-parameterized models, the alignment is typically very small, so the certificate is unavailable, and laziness fails in this regime.

A case where laziness is lost. We consider a linear base model with an exactly constant Jacobian, and show that rank-one LoRA still loses laziness as the model grows. Let $x \in \mathbb{R}^{d_x}$ with $\|x\| = 1$, and let the base model be the linear map $f(W) = Wx$ with a single weight row $W \in \mathbb{R}^{1 \times d_x}$; its Jacobian $\partial_W f = x^\top$ is constant, so Term (A) is identically zero. Apply rank-one LoRA, $W = W_0 + s\,b\,a^\top$ with trainable scalar $b \in \mathbb{R}$ and vector $a \in \mathbb{R}^{d_x}$, using the standard initialization $b_0 = 0$, $\|a_0\| = 1$. Fit a single example with target $y = 1$ under the squared loss, taking $W_0 x = 0$. Writing $p := a^\top x$, the output and the LoRA Jacobian are

$$f = s\,b\,p, \qquad J_\phi = \big[\,\partial_a f,\ \partial_b f\,\big] = \big[\,s\,b\,x^\top,\ s\,p\,\big]. \tag{11}$$

Proposition G.2 (Drift grows with dimension)**.** *Let $p_0 := a_0^\top x$ and assume $0 < p_0^2 \le 1/s$. Under gradient flow from $(a_0, b_0 = 0)$ the loss converges to zero, the trained factor converges to b_∞ satisfying*

$$b_\infty^2 \;=\; \tfrac{1}{2}\Big(\sqrt{p_0^4 + 4/s^2} - p_0^2\Big) \;\ge\; \frac{1}{2s}, \tag{12}$$

and the relative drift of the LoRA Jacobian satisfies

$$\Delta \;\ge\; \frac{b_\infty}{|p_0|} \;\ge\; \frac{1}{|p_0|\sqrt{2s}}. \tag{13}$$

For the standard random initialization $a_0 \sim \mathrm{Unif}(\mathbb{S}^{d_x-1})$, $|p_0| = \Theta(1/\sqrt{d_x})$ with high probability, so

$$\Delta \;=\; \Omega\big(\sqrt{d_x/s}\big) \;\xrightarrow[d_x\to\infty]{}\; \infty, \tag{14}$$

even though the base Jacobian is exactly constant. A matching upper bound of the same order holds, giving $\Delta = \Theta(\sqrt{d_x/s})$.

Proof. The proof uses three facts. (i) At initialization the Jacobian is small: $b_0 = 0$ leaves a single nonzero entry, so $\|J_{\phi,0}\| = s|p_0|$. (ii) To reach zero loss, b must converge to an order-one value $b_\infty \ge 1/\sqrt{2s}$, regardless of how small p_0 is. (iii) The a-block of the Jacobian in (11) is $s\,b\,x^\top$, so this block alone has limiting norm $s\,b_\infty$, which exceeds the norm of the entire initial Jacobian. Dividing (iii) by (i) gives (13). It remains to establish (ii) by solving the flow exactly.

Step 1 (reduction to two scalar variables). Let $e := f - y = sbp - 1$ be the residual. Gradient flow gives $\dot a = -e\,\partial_a f = -e\,s\,b\,x$: the scalar $-esb$ multiplies the fixed vector x, so a moves only along x. Decompose $a = p\,x + a^\perp$ with $a^\perp \perp x$. Then $a^\perp$ remains constant and does not affect the output, since f depends on a only through $p = a^\top x$. The dynamics reduce to the pair (b, p), with $\dot b = -e\,s\,p$ and $\dot p = \dot a^\top x = -e\,s\,b$:

$$\dot b = -e\,s\,p, \qquad \dot p = -e\,s\,b. \tag{15}$$

Step 2 (a conserved quantity). The system (15) is symmetric, so $b\dot b - p\dot p = 0$, i.e. $\frac{d}{dt}(b^2 - p^2) = 0$, and $b^2 - p^2$ retains its initial value $-p_0^2$:

$$p^2 = b^2 + p_0^2 \qquad \text{for all } t. \tag{16}$$

Step 3 (how far b must travel). By (15), both velocities vanish only if $e = 0$ or $b = p = 0$; the latter is impossible because (16) implies $p^2 \ge p_0^2 > 0$, so $e = 0$ is the only stationary point on the trajectory's curve. The trajectory remains bounded: the gradient flow does not increase $\mathcal{L} = \frac{1}{2}e^2$, while along (16) $|e| \to \infty$ as $b \to \pm\infty$, so $b(t)$ remains bounded. Finally, $\dot{\mathcal{L}} = -\|\nabla\mathcal{L}\|^2$ and $\mathcal{L} \ge 0$ imply $\int_0^\infty \|\nabla\mathcal{L}\|^2 dt < \infty$, hence $\|\nabla\mathcal{L}\| \to 0$. A bounded trajectory with velocity tending to zero and only one possible stationary limit point converges to that point. Therefore $e \to 0$, i.e. $s\,b_\infty p_\infty = 1$. Substituting $p_\infty = 1/(sb_\infty)$ into (16) and multiplying by b_∞^2 gives the quadratic $b_\infty^4 + p_0^2 b_\infty^2 - 1/s^2 = 0$, whose positive root is given by (12). The lower bound follows from $\sqrt{p_0^4 + 4/s^2} \ge 2/s$, which gives $b_\infty^2 \ge 1/s - p_0^2/2 \ge 1/(2s)$ using $p_0^2 \le 1/s$.

Step 4 (small baseline, large drift). At initialization, (11) with $b_0 = 0$ gives $J_{\phi,0} = [\,0,\; sp_0\,]$, so $\|J_{\phi,0}\| = s|p_0|$. At convergence, the a-block of $J_{\phi,\infty}$ is $s\,b_\infty x^\top$, and the norm of a matrix is at least the norm of any of its blocks, so $\|J_{\phi,\infty} - J_{\phi,0}\| \ge s\,b_\infty$. Dividing by the initial Jacobian norm gives $\Delta \ge b_\infty/|p_0|$, and (12) yields (13). For the matching upper bound, (16) gives $|p_\infty| \le b_\infty + |p_0|$, so $\|J_{\phi,\infty} - J_{\phi,0}\| \le s\,b_\infty + s|p_\infty - p_0| \le 2s(b_\infty + |p_0|)$ and hence $\Delta \le 2b_\infty/|p_0| + 2$, which is of the same order as the lower bound.

Step 5 (dimension). For $a_0 \sim \mathrm{Unif}(\mathbb{S}^{d_x-1})$, the projection $a_0^\top x$ has mean 0 and variance $1/d_x$, so $|p_0| = \Theta(1/\sqrt{d_x})$ with high probability (and $p_0^2 \le 1/s$ holds for large d_x); (13) yields (14). Finally, the base Jacobian $\partial_W f = x^\top$ does not depend on W, so its drift is identically zero. □

Remark G.3 (What goes wrong). This effect is driven by the alignment $\gamma = |a_0^\top x|$ of (7). In high dimension, a random direction is almost orthogonal to the fixed task direction x, so $\gamma = \Theta(1/\sqrt{d_x}) \to 0$, while the factor movement required to fit the target remains of order one ($b_\infty =$

$\Theta(1/\sqrt{s})$, independent of d_x). The left side of (10) is of order one and the right side shrinks, so the condition fails. Since (10) is only sufficient, this alone does not establish a loss of laziness, but Proposition G.2 computes Δ directly and shows that it diverges. This is not a failure of base laziness, since the base model is linear; it results from parameterizing the update as a product.

The fixed-A variant. The difficulty is Term (B), which is present only because both factors move. Fix $A = A_0$ and train only B, as in LoRA-FA (Zhang et al., 2023). Then $W = W_0 + s\,BA_0$ is *linear* in B, so the factor map $\Pi = \partial W/\partial B = s(\cdot)A_0$ is constant, $\Pi_t = \Pi_0$, and Term (B) is exactly zero. The LoRA Jacobian can therefore drift only through changes in the base Jacobian.

Theorem G.4 (Fixed-A LoRA stays as lazy as the base)**.** *Suppose the base model is lazy, with relative Jacobian drift Δ_W on the adapter subspace, and let $\kappa_A = \sigma_{\max}(A_0)/\sigma_{\min}(A_0)$. For the fixed-$A$ adapter $W = W_0 + s\,BA_0$, $\Delta \le \kappa_A \Delta_W$. In particular, if the rows of A_0 are orthonormal ($\kappa_A = 1$), then $\Delta \le \Delta_W$.*

Proof. Since Π is constant, (5) gives $J_\phi = J_W \Pi$ at every step, so (6) reduces to its first term, $J_{\phi,t} - J_{\phi,0} = (J_{W,t} - J_{W,0})\,\Pi$. Taking norms gives, $\|J_{\phi,t} - J_{\phi,0}\| \le \|J_{W,t} - J_{W,0}\|\,\|\Pi\|$, while $\|J_{\phi,0}\| = \|J_{W,0}\Pi\| \ge \|J_{W,0}\|\,\sigma_{\min}(\Pi)$ on the adapter subspace. Dividing these bounds and using $\|\Pi\|/\sigma_{\min}(\Pi) = \sigma_{\max}(A_0)/\sigma_{\min}(A_0) = \kappa_A$, gives,

$$\Delta = \sup_t \frac{\|J_{\phi,t} - J_{\phi,0}\|}{\|J_{\phi,0}\|} \;\le\; \kappa_A \sup_t \frac{\|J_{W,t} - J_{W,0}\|}{\|J_{W,0}\|} \;=\; \kappa_A \Delta_W.$$

With orthonormal rows, $\kappa_A = 1$. □

When the readout is trained jointly, as in our LoRA experiments, its movement enters through $J_W = W_{\text{head}}\,\partial\phi/\partial W$, so it is part of Δ_W and covered by the bound; the drift measured in Figure 2(c) includes it as well.

Revisiting the counterexample. In the example of Proposition G.2, freezing $a = a_0$ makes $f = s\,b\,p_0$ linear in b, so $J_\phi = sp_0$ is exactly constant and $\Delta = 0$ for every d_x. The same task that caused the two-factor drift to diverge is fit with a constant Jacobian when one factor is fixed. The remaining cost is conditioning rather than drift: poor alignment requires b to become large to fit the target. Orthonormalizing A_0 or choosing it to align with the task improves conditioning without reintroducing Term (B).

Remark G.5 (A tempting but incorrect argument). One sometimes argues that LoRA is always lazy in wide networks: "both factors move by $\mathcal{O}(1/\sqrt{m})$ as the width $m \to \infty$, so $\Pi_t - \Pi_0 \to 0$ and Term (B) vanishes." This argument relies on an incorrect premise. Although the base network's curvature shrinks with width, causing Term (A) to vanish, the product's curvature remains fixed at s, independent of m. The factor movement D is determined by the task and the alignment, not by the width. In Proposition G.2, the factors move by $\Theta(1/\sqrt{s})$, independent of the degree of over-parameterization. The *relative* drift diverges, while the absolute drift bounded by the argument does not. Kernel-regime descriptions of fine-tuning (Malladi et al., 2023b; Jang et al., 2024) are therefore conditional rather than universal; (10) provides one explicit form of the missing condition.

G.2 Bias-only fine-tuning (BitFit)

BitFit (Zaken et al., 2022) freezes every weight matrix and trains only the bias vectors. Writing $\theta = (w, b)$ with weights $w \in \mathbb{R}^{m_w}$ and biases $b \in \mathbb{R}^{m_b}$, split the full Jacobian into its weight and bias column blocks, $J(\theta) = [\,J_w(\theta)\;\; J_b(\theta)\,]$. BitFit updates θ only in the bias coordinates, so its parameter map is affine,

$$g(b) = (w_0, b) = \bar{\theta}_0 + E_b\,b, \qquad \bar{\theta}_0 := (w_0, 0), \qquad E_b = \begin{bmatrix} 0 \\ I_{m_b} \end{bmatrix}, \tag{17}$$

where E_b is the fixed selection matrix for the bias coordinates. Its columns are distinct standard basis vectors, so $E_b^\top E_b = I$. It is therefore an isometry, with $\|E_b\| = 1$ and $\sigma_{\min}(E_b) = 1$. Since g is affine, $\nabla g = E_b$ is constant and $\nabla^2 g = 0$. For the BitFit model $h := f \circ g$, the chain rule gives

$$J_h(b) = J(g(b))\,E_b = J_b(w_0, b), \tag{18}$$

so the BitFit Jacobian is simply the bias block of the base Jacobian, evaluated at frozen weights.

Lemma G.6 (Coordinate restriction adds no curvature)**.** *For $h = f \circ g$ with g affine as in (17),*

$$\nabla^2 h = E_b^\top (\nabla^2 f)\, E_b, \qquad \textit{hence} \qquad \beta_h \le \|E_b\|^2 \beta_f = \beta_f, \tag{19}$$

where $\beta_f = \sup \|\nabla^2 f\|$ and $\beta_h = \sup \|\nabla^2 h\|$.

Proof. Differentiating $\nabla h(b) = \nabla f(g(b))\, E_b$ once more in a direction u gives, by the product rule, $\nabla^2 h(b)[u, \cdot] = \nabla^2 f(g(b))[E_b u, E_b \cdot] + \nabla f(g(b))\, \nabla^2 g(b)[u, \cdot]$. The second term vanishes because $\nabla^2 g = 0$; taking operator norms and using $\|E_b\| = 1$ gives $\beta_h \le \beta_f$. □

Comparison with LoRA. In the LoRA decomposition $\nabla^2 h = (\nabla g)^\top (\nabla^2 f)(\nabla g) + (\nabla f)(\nabla^2 g)$, the second summand is the reparametrization-curvature term that is nonzero for the bilinear product BA and drives LoRA's Jacobian drift. This term is identically zero for BitFit, which inherits only the base curvature, compressed by the isometry E_b.

Theorem G.7 (BitFit adds no reparametrization-induced drift)**.** *Suppose the base model is β_f-smooth on the region visited during training (its Jacobian J is β_f-Lipschitz), and the bias-block Jacobian at initialization is nondegenerate, $\sigma^+_{\min}(J_{b,0}) > 0$. Let $D_b = \sup_t \|b_t - b_0\|$ be the bias movement. Then, measuring drift relative to $\sigma^+_{\min}(J_{b,0})$,*

$$\Delta_{\textit{bit}} \;\le\; \frac{\beta_f\, D_b}{\sigma^+_{\min}(J_{b,0})} \;=:\; \delta_{\textit{bit}}, \tag{20}$$

which is the base model's laziness coefficient measured on the bias subspace, with the base curvature in the numerator and the base Jacobian's bias block in the denominator. The restriction map E_b is a fixed isometry, so there is no reparametrization term and no conditioning inflation: the analogue of LoRA's factor κ_A is 1.

Proof. By (18), the BitFit Jacobian at any time is $J_h(b_t) = J(\theta_t)E_b$ with $\theta_t = (w_0, b_t)$, so its displacement from initialization is $J_h(b_t) - J_h(b_0) = (J(\theta_t) - J(\theta_0))E_b$. Taking spectral norms and using $\|E_b\| = 1$, the Lipschitz bound on J, and the fact that only the biases move ($\|\theta_t - \theta_0\| = \|b_t - b_0\|$), we obtain

$$\|J_h(b_t) - J_h(b_0)\| \;\le\; \|J(\theta_t) - J(\theta_0)\| \;\le\; \beta_f \|\theta_t - \theta_0\| \;=\; \beta_f \|b_t - b_0\| \;\le\; \beta_f D_b.$$

Finally, $J_h(b_0) = J(\theta_0)E_b = J_{b,0}$, so $\sigma^+_{\min}(J_h(b_0)) = \sigma^+_{\min}(J_{b,0})$. Dividing the drift by this baseline gives (20). □

Remark G.8 (What Theorem G.7 does not say). It is tempting to interpret (20) as $\Delta_{\text{bit}} \le \Delta$, meaning that BitFit is lazier than the base model. It does not follow, and the comparison can fail on both sides of the ratio. *Denominators:* the isometry gives $\|J_b\| \le \|J\|$, but the denominator requires the reverse inequality, and $\sigma^+_{\min}$ of a column sub-block bears no monotone relation to that of the full matrix (with one weight and one bias and $J = [1 \;\; \epsilon]$, $\sigma^+_{\min}(J) \approx 1$ while $\sigma^+_{\min}(J_b) = \epsilon$). *Numerators:* D_b is the bias movement along BitFit's own trajectory; if the bias-block kernel is poorly conditioned the biases may have to travel farther than under full training to fit the same residual. Thus BitFit can be less lazy than the base model. What (20) establishes unconditionally and is the relevant claim for a replay scheme: the restriction contributes nothing of its own. Any excess of δ_{bit} is ordinary ill-conditioning of the bias-block kernel, a property of f's own adjoints, already present at $t = 0$, rather than the reparametrization-induced blow-up of bilinear LoRA. BitFit can fail to be lazy; it cannot be the cause.

Remark G.9 (The lazy-regime rate). Theorem G.7 bounds the drift in terms of the bias movement D_b, which is not known in advance. Write $e_0 := f(\theta_0) - y$ for the initial residual on the fine-tuning set; fine-tuning aims to eliminate this residual, and $\|e_0\|$ is the only quantity through which the task enters. In the lazy regime the BitFit model is well approximated by its linearization at b_0, $h(b_0+\delta) \approx f(\theta_0) + J_{b,0}\,\delta$, so fitting the task means $J_{b,0}\,\delta \approx -e_0$. Suppose e_0 is reachable (it lies in, or near, $\mathrm{range}(J_{b,0})$). The minimum-norm solution is $\delta = -J^+_{b,0} e_0$ where $J^+_{b,0}$ the Moore–Penrose pseudoinverse, and since $\|J^+_{b,0}\| = 1/\sigma^+_{\min}(J_{b,0})$,

$$D_b \;\lesssim\; \frac{\|e_0\|}{\sigma^+_{\min}(J_{b,0})}.$$

This solution is selected by gradient descent from b_0: every increment lies in $\mathrm{range}(J_{b,0}^\top)$, so the displacement remains in the row space and converges to precisely this δ. Affineness justifies the estimate: by Lemma G.6, the linearization error arises only from the base curvature β_f, without the self-amplification that makes the same estimate circular for bilinear LoRA, where moving B changes the A-block Jacobian. Substituting into (20),

$$\delta_{\text{bit}} \lesssim \frac{\beta_f \|e_0\|}{\sigma^+_{\min}(J_{b,0})^2}, \tag{21}$$

the Chizat-form laziness coefficient (Chizat et al., 2019) for the bias block. The denominator is now squared: ill-conditioning enters twice, once through the baseline and once through the additional movement it requires. In the over-parameterized regime, $\beta_f \to 0$, so $\delta_{\text{bit}} \to 0$ and laziness is recovered whenever the bias block is nondegenerate and e_0 is reachable. Reachability is a separate restriction not implied by $\sigma^+\min(Jb, 0) > 0$: the bias tangent space has dimension m_b, far below the full parameter count, so a task whose residual has a large component orthogonal to $\mathrm{range}(J_{b,0})$ admits no bias-only correction at any D_b. In this case, the estimate does not apply, and BitFit underfits: the drift remains small because the biases cannot move in a direction that fits the residual.

The bias subspace has no alignment bottleneck. Theorem G.7 requires $\sigma^+_{\min}(J_{b,0}) > 0$. For the bound to be useful, this quantity must not collapse as a random low-rank adapter's alignment does (Proposition G.2). This collapse has no analogue in the bias subspace for structural reasons. At layer ℓ, the pre-activation is $z^{(\ell)} = W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}$, so

$$\frac{\partial z^{(\ell)}}{\partial b^{(\ell)}} = I \qquad \Longrightarrow \qquad J_b^{(\ell)} = \frac{\partial f}{\partial b^{(\ell)}} = \frac{\partial f}{\partial z^{(\ell)}} \frac{\partial z^{(\ell)}}{\partial b^{(\ell)}} = \frac{\partial f}{\partial z^{(\ell)}}. \tag{22}$$

The BitFit Jacobian is exactly the network's per-layer adjoint, or backpropagated error signal, stacked over layers. Three comparative consequences follow, none of which provides a lower bound on $\sigma^+\min(Jb, 0)$ itself. *No initialization degeneracy:* standard LoRA sets $B_0 = 0$, which makes the initial adapter Jacobian rank-deficient in the A-block, so a factorization built at initialization misses those directions. By (22), the initial bias Jacobian is the full adjoint, nonzero on every active unit, and degenerate exactly where the base network's own backward signal is (dead or saturated units). This degeneracy is inherited from the base network. *Unit gain, hence natural alignment:* each bias drives its own pre-activation with gain exactly 1, so the bias subspace is the set of pre-activation coordinate directions the network's nonlinearities act on, whereas a random rank-ρ adapter's columns are, in high dimension, nearly orthogonal to the task directions. *No rank bottleneck:* BitFit trains one parameter per neuron, $m_b = \Theta(\text{depth} \times \text{width})$, a budget that grows with the model whereas a rank-ρ adapter's is fixed. The fit is spread across many parameters that each move little, as in full-model laziness, rather than forced through a rank-ρ bottleneck. This keeps D_b controlled in Remark G.9. None of these three properties provides a lower bound on $\sigma^+\min(Jb, 0)$: since $\sigma^+\min(Jb, 0)^2 = \lambda^+\min(Jb, 0 J_{b,0}^\top)$ and by (22) $J_{b,0} J_{b,0}^\top = \sum_\ell (\partial f/\partial z^{(\ell)})(\partial f/\partial z^{(\ell)})^\top$, the quantity of interest is the least nonzero eigenvalue of the tangent kernel restricted to the bias coordinates. For the unrestricted tangent kernel, the NTK literature provides lower bounds of this kind (Jacot et al., 2018); whether they hold after restriction remains open.

Remark G.10 (Relation to LoRA). Two-factor LoRA can lose laziness through two mechanisms: the bilinear reparametrization-curvature term and the alignment collapse of a random low-rank subspace. Freezing one LoRA factor eliminates the first mechanism (Theorem G.4). BitFit eliminates both: as a coordinate restriction it is affine (no curvature term, Lemma G.6), and as the unit-gain adjoint subspace it has no alignment bottleneck. In the common framework, it corresponds to the fixed-A result with the adapter map replaced by an isometry. Ill-conditioning of the trainable subspace itself remains possible for every method, including BitFit, for which it is the only remaining mechanism. Both analyses are stated for gradient flow; the discrete-time corrections are standard.

H Low-Rank Jacobian: Empirical Estimation and Theoretical Motivation

H.1 Why a Reference Set Preserves the Dominant Eigenspace

We formalize the motivation for approximating the full Jacobian covariance using a small set of parameter columns. Fix an output unit j and write

$$\widetilde{J}_{:,j,:} = [z_1 \quad \cdots \quad z_m], \qquad z_k \in \mathbb{R}^n.$$

Define the normalized full covariance by

$$C := \frac{1}{m}\widetilde{J}_{:,j,:}\widetilde{J}_{:,j,:}^\top = \frac{1}{m}\sum_{k=1}^{m} z_k z_k^\top.$$

Let $\mathcal{R} \subseteq \{1, \dots, m\}$ be a uniformly sampled subset of R column indices, and define

$$\widehat{C}_{\mathcal{R}} := \frac{1}{R}\widetilde{J}_{:,j,\mathcal{R}}\widetilde{J}_{:,j,\mathcal{R}}^\top = \frac{1}{R}\sum_{k\in\mathcal{R}} z_k z_k^\top.$$

Proposition H.1 (Reference-set covariance approximation)**.** *Assume $\|z_k\|_2^2 \le L$ for all k. Then*

$$\mathbb{E}_{\mathcal{R}}\left[\widehat{C}_{\mathcal{R}}\right] = C.$$

Moreover, for any $\delta \in (0,1)$, matrix concentration for sampling without replacement gives, with probability at least $1-\delta$,

$$\left\|\widehat{C}_{\mathcal{R}} - C\right\|_2 \le cL\left(\sqrt{\frac{\log(2n/\delta)}{R}} + \frac{\log(2n/\delta)}{R}\right),$$

for a universal constant c. Consequently,

$$\widetilde{J}_{:,j,\mathcal{R}}\widetilde{J}_{:,j,\mathcal{R}}^\top \approx \frac{R}{m}\,\widetilde{J}_{:,j,:}\widetilde{J}_{:,j,:}^\top$$

as R increases.

The factor R/m affects only the eigenvalues and therefore leaves the eigenvectors unchanged. More formally, let U_r and $\widehat{U}_r$ denote the leading r-dimensional eigenspaces of C and $\widehat{C}_{\mathcal{R}}$, respectively, and suppose

$$\gamma_r := \lambda_r(C) - \lambda_{r+1}(C) > 0.$$

The Davis–Kahan perturbation theorem then gives

$$\left\|\sin\Theta(\widehat{U}_r, U_r)\right\|_2 \le \frac{2\|\widehat{C}_R - C\|_2}{\gamma_r}.$$

Thus, whenever the covariance is well approximated and its dominant spectrum is separated by a non-negligible eigengap, the reference set recovers approximately the same leading eigenspace as the full Jacobian.

The result above formally concerns uniform sampling. Our implementation instead selects columns using gradient magnitude, concentrating the reference budget on parameters that contribute strongly to the update. We use the uniform-sampling result as the statistical justification that a small subset can preserve the covariance structure. Gradient-informed selection is an algorithmic choice intended to improve the approximation for a fixed R; its effectiveness is evaluated empirically in Appendix B.

H.2 Error of the Low-Rank Gradient Approximation

We now quantify the approximation used in Algorithm 1. For clarity, fix one output unit j and write $J_j := J_{:,j,:} \in \mathbb{R}^{n\times m}$ and $a := a_{t,:,j} \in \mathbb{R}^n$. The exact contribution of this output unit to the parameter gradient is

$$g = J_j^\top a.$$

Let

$$J_j = U\Sigma W^\top$$

be the singular value decomposition of J_j, with singular values $\sigma_1 \geq \sigma_2 \geq \cdots$, and let U_r, Σ_r, and W_r denote the corresponding leading r components. By the Eckart–Young–Mirsky theorem,

$$J_{j,r} := U_r \Sigma_r W_r^\top$$

is the best rank-r approximation of J_j, and its discarded Frobenius energy is

$$\|J_j - J_{j,r}\|_F^2 = \sum_{i>r} \sigma_i^2(J_j).$$

Hence the energy rank r_α used in Section 3.2 is precisely the smallest rank whose truncated SVD retains an α fraction of the squared Frobenius energy.

Replaying the gradient through this rank-r approximation gives

$$g_r = J_{j,r}^\top a = J_j^\top U_r U_r^\top a.$$

The discarded component is therefore

$$g - g_r = J_j^\top (I - U_r U_r^\top) a.$$

Since U_r contains the leading left singular vectors of J_j,

$$\left\| J_j^\top (I - U_r U_r^\top) \right\|_2 = \sigma_{r+1}(J_j),$$

and hence

$$\boxed{\|g - g_r\|_2 \leq \sigma_{r+1}(J_j)\, \|a\|_2.}$$

Thus, a rapidly decaying Jacobian spectrum simultaneously implies that the slice is well approximated by a low-rank matrix and that replaying gradients through this approximation incurs small error.

Our algorithm does not have access to the exact basis U_r. Instead, it recovers a basis V_j from the reference-set Jacobian as described in Appendix H.1. The update actually used is

$$\widehat{g} = J_j^\top V_j V_j^\top a = P_j V_j^\top a, \qquad P_j := J_j^\top V_j.$$

Adding and subtracting the ideal rank-r projection gives

$$\|g - \widehat{g}\|_2 \leq \sigma_{r+1}(J_j)\|a\|_2 + \|J_j\|_2 \left\| U_r U_r^\top - V_j V_j^\top \right\|_2 \|a\|_2.$$

The first term is the intrinsic rank-r truncation error. The second measures the additional error introduced by recovering the dominant subspace from the reference set.

Let C_j denote the normalized full Jacobian covariance and $\widehat{C}_j$ its reference-set approximation from Appendix H.1. If

$$\gamma_r = \lambda_r(C_j) - \lambda_{r+1}(C_j) > 0$$

is the eigengap separating the retained directions from the remaining spectrum, the Davis–Kahan perturbation theorem gives

$$\left\| U_r U_r^\top - V_j V_j^\top \right\|_2 \lesssim \frac{\|\widehat{C}_j - C_j\|_2}{\gamma_r}.$$

Consequently,

$$\boxed{\|g - \widehat{g}\|_2 \lesssim \left[\sigma_{r+1}(J_j) + \|J_j\|_2 \frac{\|\widehat{C}_j - C_j\|_2}{\gamma_r} \right] \|a\|_2.}$$

The approximation error therefore has two distinct sources: the spectral tail discarded beyond rank r, and the error incurred by recovering the dominant subspace from only R reference parameters. Increasing r reduces the first term. The upper bound on the second, tightens as the covariance error decreases or the eigengap increases. Under uniform sampling, the concentration bound in Appendix H.1 improves as R increases. The accuracy gains observed with our gradient-informed selection as r and R increase, are consistent with this analysis.

H.3 Empirical Rank Estimation

Motivation. Our method relies on the Jacobian admitting an effective low-dimensional representation. Directly computing the spectrum or rank of the full Jacobian is infeasible for modern networks, so we estimate its dominant spectral structure from the reference-set Jacobian introduced in Appendix H.1.

Empirical estimation. For a fixed output unit j, let

$$\widetilde{J}_{:,j,R} \in \mathbb{R}^{n' \times R}$$

denote the centered Jacobian restricted to the R selected parameter columns. We form the $R \times R$ Gram matrix

$$C_j = \widetilde{J}_{:,j,R}^{\top} \widetilde{J}_{:,j,R}.$$

By Proposition 3.1, C_j and $\widetilde{J}_{:,j,R} \widetilde{J}_{:,j,R}^{\top}$ have exactly the same nonzero eigenvalues. Thus, the spectral structure of the reference Jacobian can be analyzed through the much smaller matrix C_j.

We report two notions of effective rank. First, the numerical rank is

$$r_{\mathrm{num}}(C_j) := \#\left\{\lambda_i(C_j) > R\epsilon\, \lambda_1(C_j)\right\},$$

where ϵ is machine precision. Second, because numerical rank can be sensitive to very small eigenvalues, we report the energy rank

$$r_\alpha(C_j) := \min\left\{ r : \frac{\sum_{i=1}^{r} \lambda_i(C_j)}{\sum_i \lambda_i(C_j)} \geq \alpha \right\},$$

for $\alpha \in \{0.95, 0.99\}$.

Relation to the full Jacobian. Let the reference columns contain a fraction

$$f := \frac{\|\widetilde{J}_{:,j,R}\|_F^2}{\|\widetilde{J}_{:,j,:}\|_F^2}$$

of the full slice's squared Frobenius energy. If $f \geq \alpha$, retaining an α/f fraction of the reference energy is sufficient to retain an α fraction of the full energy. Hence,

$$r_\alpha\left(\widetilde{J}_{:,j,:}\right) \leq r_{\alpha/f}\left(\widetilde{J}_{:,j,R}\right).$$

Thus, when the reference columns capture nearly all of the full slice's energy, the reference spectrum provides a direct bound on the energy rank of the full slice. In our measurements the reference columns hold $f \geq 0.99$ of every slice's energy, so $\alpha/f \leq 0.96$ and the bound applies.

Interpretation. We examine how r_{num} and r_α behave as the number of samples n' increases. Effective low-rank structure is indicated when the number of dominant directions stabilizes well below the maximal possible rank $\min(n', R)$. Appendix H.1 further shows when the dominant eigenspace of the reference Jacobian approximates that of the full Jacobian.

Summary. The reference-set Gram matrix provides a scalable way to estimate the Jacobian's effective spectral dimension without constructing the full $n \times m$ matrix. The energy-capture bound above and the eigenspace analysis of Appendix H.1 justify using this reference spectrum to assess low-rank structure in the full Jacobian.