GRIT - Geometry-Aware PEFT with K-FAC Preconditioning, Fisher-Guided Reprojection, and Dynamic Rank Adaptation

Anonymous ACL submission

Abstract

Parameter-efficient fine-tuning is now the default route to adapt LLMs to specific domains, but the standard choices, LoRA and QLoRA, are largely geometry-agnostic: they optimize inside a fixed, randomly oriented low-rank subspace with first-order descent and ignore the curvature of the loss. This wastes the limited update budget on weakly informative directions and amplifies drift along sharp ones. We introduce GRIT, a curvature-aware variant of LoRA that keeps the low-rank parameterization but does three things differently. It preconditions adapter gradients in rank space using K-FAC as a natural-gradient proxy. It periodically reprojects the low-rank factors onto the dominant Fisher eigendirections, so the adapter basis tracks where curvature is actually carrying signal. And it adapts the effective rank by reading the Fisher spectrum, concentrating capacity where it is justified. Across instruction-following, comprehension, and reasoning benchmarks on LLaMA backbones, GRIT matches or surpasses LoRA and QLoRA while reducing trainable parameters by roughly 46% on average (25 to 80% across tasks), and stays on or ahead of stronger PEFT and optimizer baselines like Orthogonal-LoRA, IA3, DoRA, and Shampoo. On a curvature-modulated forgetting law derived for GRIT, the same runs show consistently lower pretraining drift than LoRA at matched data and model size.

1 PEFT's Blind Spot: Learning Inertia & Catastrophic Forgetting

Adapting billion parameter LLMs stresses memory and bandwidth; PEFT addresses this by freezing most weights and training only a ***small parameter subset*** (e.g., low–rank updates). Among PEFT variants, **LoRA** [Hu et al., 2021] and **QLoRA** [Dettmers et al., 2023] have become de facto standards. **The emerging trade-off**: Recent evidence that ***"LoRA learns less and forgets less"*** [Biderman et al., 2024] indicates that, relative to full fine–tuning, LoRA's low–rank update budget often yields ***smaller gains on hard targets*** (e.g., code/math) while ***preserving*** more of the base model's broad abilities (e.g., commonsense and general language competence). In short, PEFT often trades ***peak task improvement*** for ***retention***: constrained adapters may underfit challenging distributions yet induce fewer high impact shifts that erase pretraining knowledge.

1.1 Diagnosis: Learning Inertia vs. High-Impact Updates

Forgetting is not just *how far* you move, but *where* you move. PEFT creates ***learning inertia*** by freezing most weights and restricting adaptation to a low-rank subspace. Yet forgetting still occurs when the induced update overlaps ***sharp curvature modes*** of the pretraining objective. Let $L_{\mathrm{pt}}(w)$ be the pretraining loss and Δw the PEFT update. Near a well-fit solution, first-order terms are small, so the ***quadratic curvature term*** dominates:

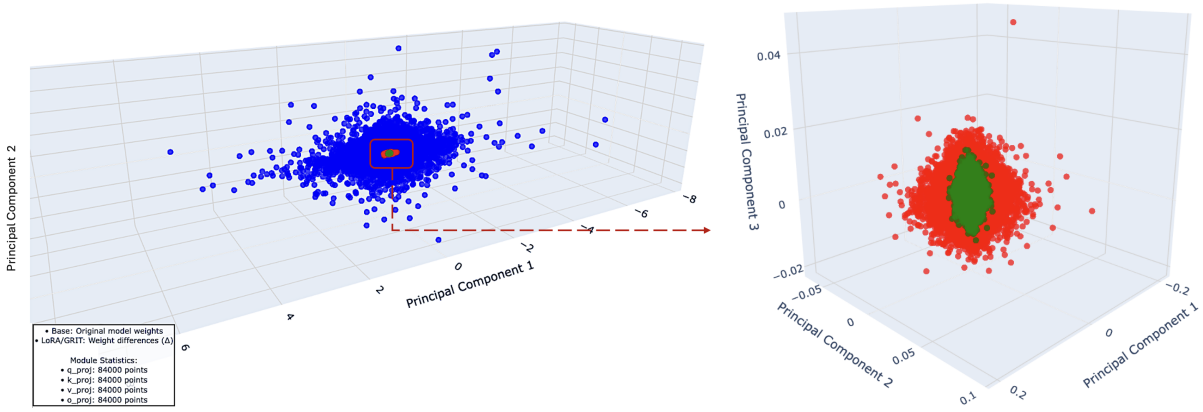
$$\Delta L_{\mathrm{pt}} \approx \frac{1}{2}\Delta w^{\top} H_{\mathrm{pt}} \Delta w = \frac{1}{2}\sum_{j} \lambda_j (u_j^{\top}\Delta w)^2,$$

where $H_{\mathrm{pt}} = \sum_j \lambda_j u_j u_j^{\top}$ is the Hessian eigendecomposition. **Intuition.** Forgetting is large when updates land on sharp directions: high λ_j amplifies large projections $|u_j^{\top}\Delta w|$ [Ghorbani et al., 2019; Keskar et al., 2017; Dinh et al., 2017].

From principle to practice. The quadratic form explains *why* forgetting increases but not *what* to monitor during training. We therefore introduce two *operational* geometry summaries that map directly onto the quadratic term and are simple to track online. **Two geometric amplifiers**:
(i) Tail mass of updates.

$$U_{\mathrm{hi}}(\tau) \;=\; \sum_{i} \mathbf{1}(|\Delta w_i| > \tau),$$

the count of coordinates exceeding a magnitude



(a) Global view: original model weights vs. LoRA/GRIT Δ-weights in PCA space.

(b) Zoom near the origin comparing LoRA vs. GRIT Δ-weights only.

Figure 1: **Geometry of parameter updates: GRIT concentrates Δ-weights into curvature-aligned subspaces.** *Setup.* PCA on parameter vectors from attention projections ($q_{\text{proj}}, k_{\text{proj}}, v_{\text{proj}}, o_{\text{proj}}$) across layers; points are mean-centered and embedded into the leading PCs (no extra scaling). **(a) Overall (left).** *Blue* points depict the **entire model's parameter space** as realized by the *original (base) weights*. Superimposed at the center, *red* points are **LoRA** *update deltas* (ΔW), and *green* points are **GRIT** *update deltas*. The visual shows that the full base space is broad and anisotropic, while both LoRA and GRIT operate in a much *smaller central region*, the effective fine-tuning manifold. **(b) Zoom on Δ (right).** The base cloud is omitted to compare updates directly: *red* = LoRA ΔW, *green* = GRIT ΔW. GRIT forms a *tighter, ellipsoidal core* with reduced radial spread versus LoRA, consistent with *rank space natural gradient* preconditioning and *Fisher aligned reprojection* that bias updates toward high curvature eigendirections while suppressing diffuse, low signal axes. **Interpretation.** GRIT densifies signal in a low-rank, curvature-aware subspace (smaller support, tighter covariance) without enlarging the update footprint steering limited parameters toward directions that matter for stability and generalization.

threshold τ.

Interpretation: heavier tails imply more frequent large coordinates, increasing the chance that $|u_j^\top \Delta w|$ becomes large and amplifies quadratic loss rise; this aligns with continual learning evidence that large, concentrated steps drive interference [Kirkpatrick et al., 2017; Zenke et al., 2017; Aljundi et al., 2018; Chaudhry et al., 2018].

(ii) Effective rank of the update covariance. Let $\{\lambda_j^{(\Delta)}\}$ be the eigenvalues (descending) of $\mathbb{E}[\Delta w\, \Delta w^\top]$. Let's define:

$$r_{\text{eff}} = \min\left\{k : \frac{\sum_{j=1}^{k} \lambda_j^{(\Delta)}}{\sum_j \lambda_j^{(\Delta)}} \geq \eta\right\} \quad (\eta \in (0,1)).$$

Interpretation: larger r_{eff} means update energy is spread across more directions, increasing overlap with sharp Hessian modes and thus $\sum_j \lambda_j (u_j^\top \Delta w)^2$ [Roy and Vetterli, 2007; Gavish and Donoho, 2014; Aghajanyan et al., 2021].

Takeaway. ***Tail mass*** controls *how big* the projections can be; ***effective rank*** controls *how many* directions those projections can land on. Either rising makes curvature overlap and hence ΔL_{pt} more likely. This diagnosis motivates *geometry-aware* PEFT procedures that *shrink tails* and *concentrate rank* away from sharp modes.

Evidence for the blind spot. Large-scale evidence shows a ***stable Pareto***: LoRA *learns less* than full fine-tuning on hard domains such as code and math, but *forgets less* of base abilities such as HellaSwag, ARC-C, and WinoGrande; full FT, in contrast, induces ***higher-rank*** perturbations, often 10–100$\times$ typical LoRA ranks [Biderman et al., 2024]. This matches ***catastrophic interference*** theory [McCloskey and Cohen, 1989; French, 1999] and recent ***continual-learning*** views of LLM adaptation [Wu et al., 2024]. Through the scaling-law lens of Sec. 4, (D_{ft}, N) sets ***how much*** forgetting to expect; the quadratic view explains ***why*** methods at the same budget diverge. Update ***geometry*** multiplies the baseline through adapter-restricted curvature exposure $\overline{\kappa} = \text{tr}(PH_{\text{pt}}P)$, effective-rank spread $\Phi(r_{\text{eff}})$, and tail mass $\Psi(U_{\text{hi}}(\tau))$. Standard, ***geometry-agnostic*** LoRA optimizes in a fixed first-order basis, tending to inflate $\overline{\kappa}$ and $U_{\text{hi}}(\tau)$ at fixed (D_{ft}, N) [Hu et al., 2021; Biderman et al., 2024]. This motivates ***curvature-aware PEFT***.

Implication. If forgetting equals *(data/model)* times *(geometry/update)*, improving retention at fixed (D_{ft}, N) requires ***shrinking the geometry factor***. This motivates ***geometry-aware PEFT***:

estimate curvature in the adapter rank space and combine ***natural-gradient/K-FAC*** preconditioning with periodic ***reprojection*** toward high signal, low interference eigendirections [Amari, 1998; Martens and Grosse, 2015; Ollivier, 2015]. In short, **geometry-aware PEFT is needed**, and we propose **GRIT**[1]: ***Geometry-Aware PEFT with K-FAC Preconditioning, Fisher-Guided Reprojection, and Dynamic Rank Adaptation***.

2 GRIT: Geometry-Aware PEFT with K-FAC Preconditioning, Fisher-Guided Reprojection, and Dynamic Rank Adaptation

GRIT is a ***geometry-aware*** PEFT framework that turns LoRA updates ($\Delta W = BA$) into a ***dynamic, curvature-aligned*** procedure with three coupled controls: **(i) curvature-aware preconditioning** applies rank-space K-FAC [Martens and Grosse, 2015; Grosse and Martens, 2016; Amari, 1998] to the Fisher F, tempering steps along sharp directions; **(ii) spectrum-driven rank scheduling** reads the per-layer Fisher spectrum and allocates rank where curvature energy concentrates; **(iii) neural reprojection** periodically gates and aligns the low-rank factors with the top-k Fisher eigenspace ($U_k U_k^\top$), preserving task signal while discarding drift. Repeating this loop lets the adapter basis track ***high-signal, low-interference*** directions. Fig. 2 gives the objective, Fig. 5 the training flow, and Fig. 1 the resulting ***sparser, geometry-aligned, targeted*** updates.

2.1 Low-Rank Adaptation Setup

Consider a transformer module with a linear projection parameterized by a weight matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. *Full fine-tuning* updates all entries of W, which is prohibitive at scale. *Low-rank adaptation* (LoRA/QLoRA) instead introduces a parameter-efficient update

$$\Delta W = BA$$

where $B \in \mathbb{R}^{d_{\text{out}} \times r}$, $A \in \mathbb{R}^{r \times d_{\text{in}}}$, and $r \ll \min(d_{\text{in}}, d_{\text{out}})$. Thus the effective weight is $W' = W + \alpha\, \Delta W$ (with scaling α). This reduces trainable parameters from $O(d_{\text{out}} d_{\text{in}})$ to $O(r(d_{\text{in}} + d_{\text{out}}))$, preserving *expressivity per parameter* while keeping memory/compute feasible.

[1]Code is available at https://anonymous.4open.science/r/GRIT-26/

Caveat-geometry agnosticism. Standard LoRA learns A, B in a **fixed** low-rank subspace using first-order updates, *ignoring* curvature. As a result, steps can over-expose sharp pretraining directions, amplifying interference. **GRIT** removes this blind spot by making the low-rank subspace *geometry-aware* through three components: **curvature aware preconditioning**, **Fisher-guided reprojection**, and **dynamic rank adaptation**.

2.2 K-FAC Based Preconditioning

From gradients to *natural* gradients. Raw stochastic gradients need not align with loss geometry. The *natural gradient* rescales $\nabla_\theta \mathcal{L}$ by the inverse Fisher matrix (FIM), yielding steepest descent under the KL metric [Amari, 1998]:

$$\theta_{t+1} = \theta_t - \eta\, F^{-1} \nabla_\theta \mathcal{L}(\theta_t),\ F = \mathbb{E}\Big[\nabla \log p(x;\theta)\, \nabla \log p(x;\theta)^\top\Big]$$

Directly forming/inverting F is infeasible for LLMs (quadratic storage, cubic inversion).

K-FAC for layers, *restricted* to rank space. K-FAC approximates the layerwise Fisher for $y = Wx$ by a Kronecker product of second moments of *input activations* and *output gradients* [Martens and Grosse, 2015; Grosse and Martens, 2016]:

$$F_{\text{layer}} \approx \Sigma_g \otimes \Sigma_a,\ \Sigma_a = \mathbb{E}\Big[xx^\top\Big],\ \Sigma_g = \mathbb{E}\Big[gg^\top\Big],\ g \equiv \frac{\partial \mathcal{L}}{\partial y}$$

Then the *natural-gradient* preconditioned update admits

$$\nabla W_{\text{nat}} \approx \Sigma_g^{-1}\, \nabla W\, \Sigma_a^{-1},$$

avoiding explicit inversion of F_{layer}.

Rank space K-FAC for LoRA. For $\Delta W = BA$, GRIT applies K-FAC *within* the rank-r subspace spanned by A, B. Let $x \in \mathbb{R}^{d_{\text{in}}}$ be activations and $g \in \mathbb{R}^{d_{\text{out}}}$ the backpropagated gradients. Define rank-projected statistics

$$a_r = Ax \in \mathbb{R}^r,\ g_r = B^\top g \in \mathbb{R}^r$$

and rank-space covariances

$$\Sigma_a^{(r)} = \mathbb{E}\Big[a_r a_r^\top\Big],\ \Sigma_g^{(r)} = \mathbb{E}\Big[g_r g_r^\top\Big] \in \mathbb{R}^{r \times r}$$

Under the K-FAC independence approximation, the Fisher restricted to the LoRA subspace factorizes as

$$F_{\text{rank}} \approx \Sigma_g^{(r)} \otimes \Sigma_a^{(r)}.$$

Consequently,

$$\nabla(\Delta W)_{\text{nat}} \approx \Sigma_g^{(r)\,-1}\, \nabla(\Delta W)\, \Sigma_a^{(r)\,-1},$$

which decouples into factor-wise updates (for $\Delta W{=}BA$):

$$\min_{A\in\mathbb{R}^{d\times r},\, B\in\mathbb{R}^{r\times d}} \underbrace{L_{\text{task}}(W_0+BA)}_{\text{(1) Task Loss}} + \lambda_{\text{K}} \underbrace{\|F^{1/2}(BA)\|_F^2}_{\text{(2) Curvature Reg.}} + \lambda_{\text{R}} \underbrace{\|BA - U_k U_k^\top BA\|_F^2}_{\text{(3) Reprojection Reg.}}$$

Figure 2: **GRIT Objective: Curvature-Aware, Projection-Constrained Fine-Tuning.** This loss balances task performance with geometric awareness and subspace filtering: **(1) Task loss** $L_{\text{task}}(W_0 + BA)$ optimizes the instruction-tuning objective using the low-rank update BA; **(2) Curvature regularization** $\|F^{1/2}BA\|_F^2$ penalizes updates in high-sensitivity regions defined by the Fisher matrix F, promoting safe adaptation; **(3) Reprojection regularization** $\|BA - U_k U_k^\top BA\|_F^2$ encourages the update to remain within the subspace spanned by the top-k eigenvectors of F, filtering noisy or low-impact directions. Hyperparameters λ_{K} and λ_{R} control the curvature and reprojection terms.

$$\nabla B \leftarrow \nabla B\, \Sigma_g^{(r)\,-1},\ \nabla A \leftarrow \Sigma_a^{(r)\,-1}\, \nabla A$$

Intuitively, $\Sigma_g^{(r)\,-1}$ suppresses steps along *high-curvature output directions*, while $\Sigma_a^{(r)\,-1}$ removes *input scale anisotropy* in the adapter subspace, yielding **curvature-aligned**, *scale invariant* updates.

Practicalities. For stability, GRIT uses (i) *damping*: $\tilde{\Sigma}_a^{(r)} = \Sigma_a^{(r)} + \lambda_a I$, $\tilde{\Sigma}_g^{(r)} = \Sigma_g^{(r)} + \lambda_g I$; (ii) *streaming* (EMA) estimates with burn-in; and (iii) Cholesky solves on $r \times r$ matrices ($r \ll d_{\text{in}}, d_{\text{out}}$). Statistics are per layer and can be cached/offloaded. In GRIT, K-FAC is the **first** step; *Fisher-guided reprojection* and *dynamic rank scheduling* follow (see **Fig. 2** and **Fig. 5**).

Takeaway. Rank-space K-FAC gives a *lightweight* natural-gradient proxy, retaining key **second order** benefits while scaling to billion parameter LLMs [Amari, 1998; Martens and Grosse, 2015; Grosse and Martens, 2016].

2.3 Neural Reprojection

Motivation. Preconditioning corrects *step directions* but leaves the *update subspace* fixed. Over training, the LoRA subspace spanned by A and B (rank r) can drift, accumulate redundancy, or misalign with **high-curvature** directions, wasting gradient signal and inflating interference. *Neural reprojection* remedies this by **reshaping** the subspace itself to track informative curvature.

Curvature-aligned subspace. Let the rank-space covariances (Sec. K-FAC) be

$$\Sigma_a^{(r)} = \mathbb{E}\left[a_r a_r^\top\right],\ \Sigma_g^{(r)} = \mathbb{E}\left[g_r g_r^\top\right] \in \mathbb{R}^{r\times r},$$

with eigendecompositions

$$\Sigma_a^{(r)} = U_A \Lambda_A U_A^\top, \qquad \Sigma_g^{(r)} = U_G \Lambda_G U_G^\top,$$

where $U_A, U_G \in \mathbb{R}^{r\times r}$ are orthogonal and $\Lambda_A, \Lambda_G \succeq 0$ carry *curvature energy* per direction.

Algorithm 1 GRIT training loop

Require: Pretrained weights W_0; LoRA rank r; LoRA scaling α; data stream $\mathcal{D}$; learning rate η; damping λ; frequencies (`kfac_upd`, `reproj_freq`); thresholds (`kfac_min`, τ, `min_rank`); flags (`ng_warmup`, `reproj_warmup`, `use_two_sided`, `rank_adapt`)

Initialize LoRA factors $\{A_\ell, B_\ell\}$; set $A_{\text{cov}} \leftarrow 0$, $G_{\text{cov}} \leftarrow 0$; *inv_ready* $\leftarrow$ **False**; $n_{\text{cov}} \leftarrow 0$; *step* $\leftarrow 0$
for each minibatch $(x, y) \in \mathcal{D}$ **do**
  Forward with $W' = W_0 + \alpha \sum_\ell B_\ell A_\ell$; compute loss $\mathcal{L}$
  Backprop to obtain raw gradients ∇A_ℓ, ∇B_ℓ and per-layer tensors X, δY
  if *step* mod `kfac_upd` $= 0$ **then**
    $a_r \leftarrow X A_\ell^\top$; $g_r \leftarrow \delta Y B_\ell$ ▷ rank-space stats (step 0+)
    Accumulate: $A_{\text{cov}} \leftarrow A_{\text{cov}} + a_r^\top a_r$; $G_{\text{cov}} \leftarrow G_{\text{cov}} + g_r^\top g_r$; $n_{\text{cov}} \leftarrow n_{\text{cov}}+1$
    if $n_{\text{cov}} \geq$ `kfac_min` **then**
      Compute $(A_{\text{cov}}+\lambda I)^{-1}$ and $(G_{\text{cov}}+\lambda I)^{-1}$; *inv_ready* $\leftarrow$ **True**
    end if
  end if
  if *step* $\geq$ `ng_warmup` **and** *inv_ready* **then**
    Natural gradient: $\nabla B_\ell \leftarrow \nabla B_\ell G_{\text{cov}}^{-1}$; $\nabla A_\ell \leftarrow A_{\text{cov}}^{-1} \nabla A_\ell$
  end if
  Optimizer step on $\{A_\ell, B_\ell\}$; freeze W_0
  if *step* $\geq$ `reproj_warmup` **and** *step* mod `reproj_freq` $= 0$ **then**
    Eigendecompose $A_{\text{cov}} = U_A \Lambda_A U_A^\top$
    if `use_two_sided` **and** *inv_ready* **then**
      Eigendecompose $G_{\text{cov}} = U_G \Lambda_G U_G^\top$
    end if
    if `rank_adapt` **then**
      $k \leftarrow \min\{j \mid \sum_{i=1}^{j} \lambda_i / \sum_{i=1}^{r} \lambda_i \geq \tau\}$; $k \leftarrow \max(k, \texttt{min_rank})$
    else
      $k \leftarrow$ `reproj_k`
    end if
    $A_\ell \leftarrow U_A^{(k)} U_A^{(k)\top} A_\ell$
    $B_\ell \leftarrow \begin{cases} B_\ell U_G^{(k)} U_G^{(k)\top}, & \text{if two-sided} \\ B_\ell U_A^{(k)} U_A^{(k)\top}, & \text{otherwise} \end{cases}$
  end if
  step $\leftarrow$ *step*$+1$
end for
return $W' = W_0 + \alpha \sum_\ell B_\ell A_\ell$

Let $U_A^{(k)}$ and $U_G^{(k)}$ collect the top-k eigenvectors (by eigenvalue), and define projectors

$$P_A = U_A^{(k)} (U_A^{(k)})^\top,\ P_G = U_G^{(k)} (U_G^{(k)})^\top \in \mathbb{R}^{r\times r}$$

We **reproject** the LoRA factors onto these domi-

nant eigenspaces:

$$\boxed{A \leftarrow P_A A,\ B \leftarrow B P_G}$$

so the low-rank update $\Delta W = BA$ remains in a *rank-r* space with a **curvature-aligned** basis.

Gating, scheduling, and stability. To avoid premature rotations, we enable the G-side projection only after $\Sigma_g^{(r)}$ has accumulated at least $N_{\min}$ effective samples; otherwise, we *fallback* to $U_A^{(k)}$ for both sides in the first epochs. Reprojection runs at a fixed frequency (every T steps) or adaptively when the spectrum mass ratio $\sum_{i=1}^{k} \Lambda_{(\cdot),i} / \sum_{i=1}^{r} \Lambda_{(\cdot),i}$ crosses a threshold. Since $\Sigma_a^{(r)}$ and $\Sigma_g^{(r)}$ are $r \times r$, eigendecompositions are *cheap*, and $U_A^{(k)}, U_G^{(k)}$ are cached between steps. For numerical robustness, we use $\tilde{\Sigma}_{(\cdot)}^{(r)} = \Sigma_{(\cdot)}^{(r)} + \lambda I$ (damping), warm up statistics before projection, and interpolate updates if needed:

$$\begin{aligned} A &\leftarrow (1-\gamma)A + \gamma\, P_A A, \\ B &\leftarrow (1-\gamma)B + \gamma\, B P_G, \qquad \gamma \in [0,1]. \end{aligned}$$

Effect. Neural reprojection *removes low-energy directions*, suppresses noise, and **rotates** the LoRA subspace toward *high-signal, low-interference* eigendirections. Unlike pure preconditioning (which only rescales steps), reprojection **evolves the basis** so that adaptation occurs where curvature indicates capacity is most valuable, improving **stability** and **parameter efficiency** at fixed rank.

2.4 Dynamic Rank Adaptation

Why adapt rank? Reprojection aligns the subspace, but a *fixed* rank r can still **underfit** (too few directions) or **overfit** (retain redundant, low-energy directions). GRIT therefore lets the *effective* rank track the spectrum.

Energy-based rule. Let $\lambda_1 \geq \cdots \geq \lambda_r \geq 0$ be eigenvalues of a rank-space covariance (activation- or Fisher-side; cf. Secs. 1.1, K-FAC). Define the smallest k capturing energy fraction $\tau \in (0,1]$:

$$k = \min\left\{ j \;\middle|\; \frac{\sum_{i=1}^{j} \lambda_i}{\sum_{i=1}^{r} \lambda_i} \geq \tau \right\}.$$

Here "energy" is cumulative variance/mass in leading eigendirections.

Constraints and gating. We bound

$$k \in [\,\texttt{min_rank},\, r\,],$$

where `min_rank` prevents collapse and r is the max LoRA rank at initialization. To avoid early misestimation, we use a warmup gate: updates to k start only after sufficient samples for stable spectra (e.g., EMA/Fisher burn in).

How adaptation is realized. Rank adaptation is implemented through the reprojection operators. With $U^{(k)}$ the top-k eigenvectors, projectors $P_A = U_A^{(k)}(U_A^{(k)})^\top,\ P_G = U_G^{(k)}(U_G^{(k)})^\top$ suppress low energy directions during

$$A \leftarrow P_A A,\ B \leftarrow B P_G$$

No hard masking or tensor resizing is required since the directions remain stored and can re-enter if their eigenvalues grow.

Update incorporation. After reprojection and rank selection,

$$\Delta W_{\text{new}} = B_{\text{new}} A_{\text{new}},\ W' = W + \Delta W_{\text{new}}$$

yielding curvature-aware, rank-adaptive updates under a stable adapter parameterization.

Implementation summary. Algorithm 1 maps directly to our system: rank space covariances via autograd hooks; sample gated, damped K-FAC inverses; natural-gradient preconditioning before the optimizer step; and periodic Fisher-guided reprojection with dynamic rank. cf. Apndx. B and Apndx. C.

3 Experiments and Results

Setup & Datasets. Unless stated, we fine-tune **Llama 3.2–3B** and **Llama 3.1–8B** with *4-bit NF4* quantization and `bf16` compute. **GRIT** reuses the **QLoRA** data pipeline for an apples-to-apples comparison; *reprojection* is gated by curvature-sample thresholds, and *dynamic rank* follows a cumulative-energy rule. Seeds, optimizers, and schedules appear in Appx. A.7. Evaluation spans five benchmarks: *Alpaca* [Wang et al., 2023] (52k instruction–response pairs), *Dolly 15k* [Databricks, 2023] (15k human-written prompts), *BoolQ* [Clark et al., 2019] (yes/no over Wikipedia), *QNLI* from GLUE [Wang et al., 2019] (sentence-pair entailment), and *GSM8K* [Cobbe et al., 2021] (grade-school math reasoning).

Baselines. We compare GRIT to strong PEFT baselines: **LoRA** [Hu et al., 2021], **QLoRA** [Dettmers et al., 2023], **IA^3** [Liu et al., 2022] (per-module gating in lieu of rank updates), **DoRA** [Liu et al., 2024] (direction magnitude decomposition for stabler adapters), and **orthogonal-LoRA** (control basis orthogonality).

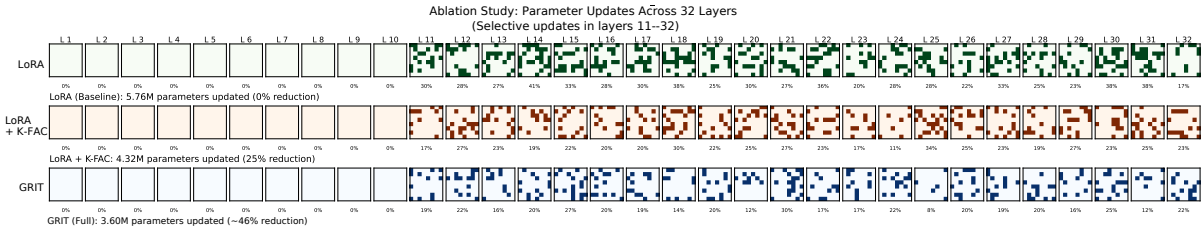


Figure 3: **Ablation - *parameter update patterns* across LLaMA (32 layers).** Each **8×8** mini-grid depicts one layer; *colored cells* mark updated parameters. We freeze **layers 1–10** because these early layers encode *task-agnostic representations*, while **layers 10–15** serve as *transition* to task-specific features and **layers 16–32** provide *refinement* [Zhao et al., 2024]; this follows the PEFT practice of adapting only the **later layers** (**11–32**). Within adapted layers, the mean update density is **LoRA** *30%* (green), **LoRA+K-FAC** *22.5%* (orange), and **GRIT** *18.75%* (blue). *Note:* these are **per-layer** densities; the **overall** update fraction is smaller because early layers are *not* adapted. Totals (annotated under each row): **LoRA** *5.76M* params (0% reduction), **LoRA+K-FAC** *4.32M* (25% reduction), **GRIT** *3.60M* (37.5% reduction). Counts assume each adapted layer updates **4** weight matrices of **65,536** parameters. *Takeaway:* **curvature-aware** preconditioning and **reprojection** *reduce update density* vs. standard LoRA while retaining capacity.

To isolate the role of curvature modeling apart from subspace alignment, we include factored second-order **Shampoo** [Anil et al., 2021] *without* Fisher-guided reprojection as an optimizer control. This set spans where capacity is injected (ranks vs. gates), how it is constrained (quantization, decomposition, orthogonality), and how updates are preconditioned (first vs. second order), providing a backdrop for GRIT's geometry-aware contributions.

Performance. GRIT matches or exceeds quality while sharply cutting trainable parameters. Across five benchmarks and two model sizes, GRIT attains parity or small gains over strong PEFT baselines while *substantially* reducing update footprint (**Table 15**). On **Alpaca**, GRIT is within $<1\%$ of the best ROUGE-1/2/L and BERTScore at both scales, yet trains only **8.45M** params on 3B ($\mathbf{65.3\%}\downarrow$ vs. LoRA/QLoRA) and **19.27M** on 8B ($\mathbf{77.0\%}\downarrow$). On **Dolly-15k**, GRIT attains the top ROUGE-1/2/L and BLEU for 3B, with a **30.0%** reduction in trained params (17.01M vs. 24.31M), and remains competitive on 8B while trimming **35.2%–54.5%**. On **GSM8K** (reasoning), GRIT ties or trails by $< 1.5\%$ on sequence metrics for 3B, and *wins accuracy* on 8B (**0.6619**, best-in-block) with a **27.8%** reduction (60.5M). On **QNLI** (NLI), GRIT is best or within $< 1.0\%$ across Accuracy/Precision/Recall/F1 at 3B while cutting params by **68.1%** (7.75M), and remains near the block leader on 8B with **64.9%–80.0%** savings. On **BoolQ**, GRIT is at or near the top across metrics with **38.2%** (3B) and **26.6%** (8B) fewer trained parameters. An extended ablation on Llama-3.2 3B (**Table 1**) confirms the trend against stronger PEFT/optimizer controls: orthogonal-LoRA, IA^3, DoRA/Eff-FT, and Shampoo show modest metric fluctuations around GRIT, but none close the parameter-efficiency gap.

Where the savings come from. GRIT's dynamic, geometry-aware allocation adapts the *effective* rank and concentrates updates in informative layers, yielding task-dependent compression rather than a fixed adapter budget. This appears as consistent per-task reductions in the "# Params Trained" column of **Table 15** (e.g., Alpaca: $\mathbf{65.3\%}$–$\mathbf{77.0\%}\downarrow$; Dolly-15k: $\mathbf{30.0\%}$–$\mathbf{54.5\%}\downarrow$; BoolQ/QNLI: $\mathbf{26.6\%}$–$\mathbf{80.0\%}\downarrow$) while maintaining block-best or near-best quality. The update pattern visualization in **Fig. 3** shows the same story at the layer level: GRIT suppresses early layers and densifies updates selectively in middle to late blocks, delivering *sparser, better aimed* updates at the same memory budget. Overall, **GRIT offers a favorable quality–efficiency trade-off** relative to LoRA/QLoRA and stronger PEFT baselines, with *consistent parameter savings and competitive accuracy* across instruction following, classification, and math reasoning.

4 Scaling Laws for Forgetting: LoRA vs. GRIT

Fine-tuning LLMs typically induces ***catastrophic forgetting***, measured as the increase in pretraining loss L_{pt} after adaptation. Following Bethune et al. [2025], forgetting follows a ***power law*** in fine-tuning data volume D_{ft} and model size N:

$$\boxed{L_{pt} = L_{pt}^0 + A\frac{D_{ft}^{\beta}}{N^{\alpha}} + E}$$

Model: Llama-3.2 (3B)	LoRA	QLoRA	GRIT (vs. LoRA)	Orthogonal-LoRA	IA^3	DoRA/Eff-FT	Shampoo
ALPACA							
ROUGE-1	0.1852	0.1292	**0.1844** (↓ 0.8%)	0.1870 (↑ 1.4%)	0.1680 (↓ 8.9%)	**0.1915** (↑ 3.9%)	0.1885 (↑ 2.2%)
ROUGE-2	0.0825	0.0562	**0.0818** (↓ 0.8%)	0.0836 (↑ 2.2%)	0.0710 (↓ 13.2%)	**0.0868** (↑ 6.1%)	0.0850 (↑ 3.9%)
ROUGE-L	0.1426	0.0983	**0.1425** (↓ 0.1%)	0.1440 (↑ 1.1%)	0.1310 (↓ 8.1%)	**0.1478** (↑ 3.7%)	0.1456 (↑ 2.2%)
BLEU	0.0443	0.0235	**0.0430** (↓ 2.9%)	0.0451 (↑ 4.9%)	0.0380 (↓ 11.6%)	**0.0472** (↑ 9.8%)	0.0461 (↑ 7.2%)
BERTScore	0.8343	0.7948	**0.8354** (↑ 0.1%)	0.8350 (↓ 0.0%)	0.8230 (↓ 1.5%)	0.8349 (↓ 0.1%)	0.8351 (↓ 0.0%)
# Params Trained	24.31M	24.31M	**8.45M** (65.3% ↓)	24.31M (0.0%)	**2.10M** (91.4% ↓)	24.31M (0.0%)	24.31M (0.0%)
Dolly-15k							
ROUGE-1	0.1733	0.1108	**0.1976** (↑ 14.0%)	0.1795 (↓ 9.2%)	0.1602 (↓ 18.9%)	0.1931 (↓ 2.3%)	0.1864 (↓ 5.7%)
ROUGE-2	0.0824	0.0519	**0.0994** (↑ 20.7%)	0.0856 (↓ 13.9%)	0.0718 (↓ 27.8%)	**0.1006** (↑ 1.2%)	0.0941 (↓ 5.3%)
ROUGE-L	0.1368	0.0884	**0.1568** (↑ 14.6%)	0.1402 (↓ 10.6%)	0.1243 (↓ 20.7%)	0.1541 (↓ 1.7%)	0.1486 (↓ 5.2%)
BLEU	0.0533	0.0297	**0.0560** (↑ 5.1%)	0.0542 (↓ 3.2%)	0.0451 (↓ 19.5%)	**0.0574** (↑ 2.5%)	0.0566 (↑ 1.1%)
BERTScore	0.8295	0.8005	**0.8344** (↑ 0.6%)	0.8311 (↓ 0.4%)	0.8192 (↓ 1.8%)	0.8335 (↓ 0.1%)	0.8322 (↓ 0.3%)
# Params Trained	24.31M	24.31M	**17.01M** (30.0% ↓)	24.31M (0.0%)	**2.10M** (91.4% ↓)	24.31M (0.0%)	24.31M (0.0%)
QNLI							
Accuracy	0.8938	0.8885	**0.9053** (↑ 1.3%)	0.8984 (↓ 0.8%)	0.8820 (↓ 2.6%)	0.9026 (↓ 0.3%)	0.9001 (↓ 0.6%)
Precision	0.8939	0.8971	**0.9059** (↑ 1.3%)	0.8991 (↓ 0.8%)	0.8842 (↓ 2.4%)	0.9037 (↓ 0.2%)	0.9010 (↓ 0.5%)
Recall	0.8939	0.8893	**0.9055** (↑ 1.3%)	0.8977 (↓ 0.9%)	0.8805 (↓ 2.8%)	0.9021 (↓ 0.4%)	0.8990 (↓ 0.7%)
F1	0.8938	0.8880	**0.9052** (↑ 1.3%)	0.8981 (↓ 0.8%)	0.8810 (↓ 2.7%)	0.9029 (↓ 0.3%)	0.8996 (↓ 0.6%)
# Params Trained	24.31M	24.31M	**7.75M** (68.1% ↓)	24.31M (0.0%)	**1.20M** (95.1% ↓)	24.31M (0.0%)	24.31M (0.0%)
BoolQ							
Accuracy	0.7834	0.7525	**0.7749** (↓ 1.1%)	0.7860 (↑ 1.4%)	0.7402 (↓ 4.5%)	0.7815 (↑ 0.9%)	0.7851 (↑ 1.3%)
Precision	0.7982	0.7491	**0.7908** (↓ 0.9%)	0.8010 (↑ 1.3%)	0.7385 (↓ 6.6%)	0.7972 (↑ 0.8%)	0.7994 (↑ 1.1%)
Recall	0.8720	0.9050	**0.8671** (↓ 0.6%)	0.8692 (↑ 0.2%)	0.8204 (↓ 5.4%)	0.8701 (↑ 0.3%)	0.8710 (↑ 0.4%)
F1	0.8335	0.8197	**0.8272** (↓ 0.8%)	0.8357 (↑ 1.0%)	0.7796 (↓ 5.8%)	0.8324 (↑ 0.6%)	0.8348 (↑ 0.9%)
# Params Trained	24.31M	24.31M	**15.03M** (38.2% ↓)	24.31M (0.0%)	**1.60M** (93.4% ↓)	24.31M (0.0%)	24.31M (0.0%)
GSM8K							
Accuracy	0.3935	0.3836	0.3867 (↓ 1.7%)	0.3962 (↑ 2.5%)	0.3564 (↓ 7.8%)	**0.3991** (↑ 3.2%)	0.3978 (↑ 2.9%)
# Params Trained	24.31M	24.31M	**15.30M** (37.1% ↓)	24.31M (0.0%)	**2.10M** (91.4% ↓)	24.31M (0.0%)	24.31M (0.0%)

Table 1: **More Baselines on *Llama-3.2 3B***. Rows are grouped by dataset (blue/gray bands). Each metric cell reports the *absolute* score followed by a *relative delta*: for all columns except **GRIT**, the delta is computed *vs. GRIT*; in the **GRIT** column, the delta is computed *vs. LoRA* to show GRIT's improvement or drop. For ROUGE/BLEU/BERTScore/Accuracy/Precision/Recall/F1, larger is better; arrows indicate better/worse. **Bold** marks the best value within the row. The "# Params Trained" rows report the *absolute* number of trainable parameters for each method and, in parentheses, the % change *vs. LoRA* (lower is better). This layout makes quality deltas relative to GRIT explicit while exposing GRIT's parameter savings over LoRA.

where L_{pt}^0 is the original pretraining loss and A, α, β, E are fitted constants. The law captures the empirical trade-off: more fine-tuning data increases drift, while larger models dilute per-update distortion.

GRIT as an effective-capacity correction. The baseline power law explains how forgetting scales with D_{ft} and N, but it does not specify *where* the fine-tuning update moves in parameter space. This missing geometry matters because sharp directions amplify even small update projections, while flatter directions absorb adaptation with less pretraining drift. Near a pretrained solution, a second-order expansion gives

$$\Delta L_{pt} \approx \frac{1}{2}\Delta w^\top H_{pt} \Delta w = \frac{1}{2}\sum_j \lambda_j (u_j^\top \Delta w)^2,$$

where λ_j and u_j are Hessian/Fisher eigenvalues and eigendirections of the pretrained loss. Thus forgetting is controlled not only by update size, but by the ***curvature-weighted interference mass***

$$\mathcal{I}(\Delta w) = \sum_j \lambda_j (u_j^\top \Delta w)^2.$$

Fig. 4 makes the mechanism explicit. **Panel A** shows a ***sharp-mode head*** in the pretrained Hessian/Fisher spectrum: a few directions dominate curvature, while the tail is comparatively flat. **Panel B** shows the failure mode: ***LoRA leaks update energy into these sharp modes***, increasing the interference mass $\mathcal{I}$. ***GRIT instead suppresses sharp-mode overlap*** and concentrates energy in stable, high-signal subspaces. **Panel C** shows the causal controls: ***rank-space K-FAC*** rescales sharp directions, ***Fisher-guided reprojection*** rotates the adapter basis, and ***dynamic rank adaptation*** removes diffuse low-signal support.

This gives a direct algebraic interpretation of the geometry multiplier. If GRIT reduces curvature-weighted interference relative to LoRA by

$$c_{\text{geom}} = \frac{\mathcal{I}_{\text{GRIT}}}{\mathcal{I}_{\text{LoRA}}},$$

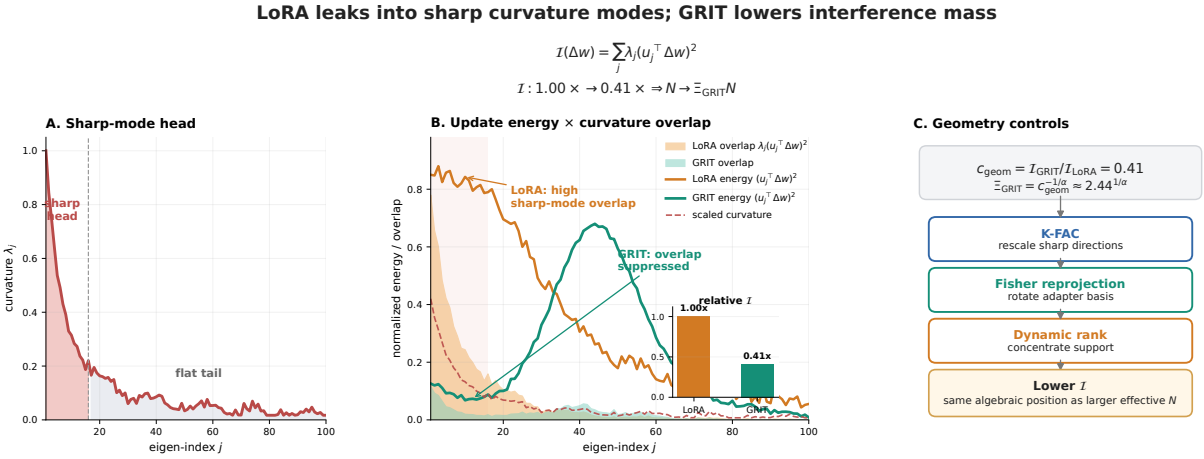


Figure 4: ***LoRA forgets by overlapping sharp curvature modes; GRIT suppresses this overlap.*** For an update Δw, forgetting scales with the curvature-weighted interference mass $\mathcal{I}(\Delta w) = \sum_j \lambda_j (u_j^\top \Delta w)^2$. **(A)** The pretrained model exhibits a highly anisotropic Hessian/Fisher spectrum with a small sharp-mode head. **(B)** LoRA distributes update energy broadly, leaking into sharp modes and producing large interference mass, whereas GRIT redirects energy toward flatter, stable eigenspaces and reduces the overlap term. **(C)** GRIT achieves this through ***rank-space K-FAC*** (step rescaling), ***Fisher-guided reprojection*** (basis rotation), and ***dynamic rank adaptation*** (support concentration), lowering the normalized interference estimate from $1.00\times$ to $0.41\times$.

then the baseline forgetting term is rescaled as

$$A\frac{D_{ft}^{\beta}}{N^{\alpha}} \cdot c_{\mathrm{geom}}.$$

Here, $c_{\mathrm{geom}} < 1$ means that GRIT induces less pretraining-loss drift for the same D_{ft} and N. Equivalently, we choose Ξ_{GRIT} so that

$$(\Xi_{\mathrm{GRIT}} N)^{-\alpha} = c_{\mathrm{geom}} N^{-\alpha},$$

which yields

$$A\frac{D_{ft}^{\beta}}{N^{\alpha}} c_{\mathrm{geom}} = A\frac{D_{ft}^{\beta}}{(\Xi_{\mathrm{GRIT}} N)^{\alpha}}, \qquad \Xi_{\mathrm{GRIT}} = c_{\mathrm{geom}}^{-1/\alpha} = \left(\frac{\mathcal{I}_{\mathrm{LoRA}}}{\mathcal{I}_{\mathrm{GRIT}}}\right)^{1/\alpha}.$$

Thus $N \to \Xi_{\mathrm{GRIT}} N$ is not a claim that GRIT increases the literal parameter count. It is an equivalent way to express that GRIT reduces the geometry-dependent part of forgetting in the same algebraic position as larger effective model capacity. In the normalized estimate shown in Fig. 4, $\mathcal{I}$ drops from 1.00 to 0.41, giving $c_{\mathrm{geom}} \approx 0.41$ and therefore $\Xi_{\mathrm{GRIT}} \approx 2.44^{1/\alpha}$.

Finally, we parameterize this multiplier using auditable geometry statistics:

$$\Xi_{\mathrm{GRIT}} = (1 + \gamma_r r_{\mathrm{eff}})(1 + \gamma_a \rho_{\mathrm{align}})(1 + \gamma_p \pi_{\mathrm{proj}}),$$

where r_{eff} is effective rank, ρ_{align} measures alignment with the Fisher top-k subspace, and π_{proj} is the spectral mass retained after reprojection. These are observable during training, so the multiplier can be logged rather than treated as a black-box fit. At fixed (D_{ft}, N), increasing these terms raises Ξ_{GRIT}, lowers curvature-weighted interference, and predicts reduced forgetting.

5 Conclusion

What we did. We introduced **GRIT**, a *geometry-aware* PEFT recipe that augments LoRA with three synergistic components: *rank-space natural gradients* via K-FAC, *Fisher-guided reprojection* to align updates with dominant curvature, and *dynamic rank adaptation* to allocate capacity where signal concentrates. Together, these mechanisms steer low-rank updates into *high signal, low interference* directions.

What we achieved. Across instruction-following, classification, and reasoning tasks on LLaMA backbones, GRIT *matches or exceeds* strong baselines while *substantially reducing* trainable parameters (typ. ∼46% average, 25–80% ∼tasks), yielding a robust efficiency–quality trade-off. Empirically, GRIT's curvature modulated forgetting obeys a power law with a larger effective capacity factor, consistently lower drift at fixed data and model size.

Future work includes **stronger curvature estimators** beyond rank-space K-FAC, **principled schedules** for reprojection frequency and rank budgets, and **broader evaluations** on *multimodality*.

Limitations

GRIT makes significant contributions towards increased stability and forgetting behavior during low-rank adaptation. However, several limitations also exist. These limitations are mostly related to the estimation of curvature, scheduling of sub-

spaces, and how the advantages of geometric control carry over to other scenarios outside of the considered ones. The estimation of Fisher/K-FAC statistics is done using mini-batches and can be very noisy during the initial stage of training, possibly causing instability in the preconditioning process or inducing premature reprojection. Subspace scheduling may require different schedules depending on the task, type of model, and data distribution, and determining an optimal one would require future research. In case of spurious correlations between different tasks within the data domain, the reprojection will preserve progress but deteriorate generalization capabilities beyond this domain. Flat spectra or small eigenvalue gaps can also cause effective rank to fluctuate, leading to uneven capacity allocation across layers.

GRIT also maintains covariance statistics and periodically performs eigendecompositions and reprojections, incurring overhead beyond purely first-order PEFT methods. Very large models or aggressive projection schedules may require additional system optimization. With quantized backbones, numerical noise can further bias covariance estimates and low-rank inverses. The damping factors, frequencies of reprojection, ranks of selection, and warm-up periods are model and task specific and may not be portable as is to other models and objectives, making their application less portable. Lastly, our experiments focus on English, short-context benchmarks on two LLaMA models. Long-context adaptation, multilingual transfer, continual learning, retrieval-augmented settings, and multimodal backbones remain untested.

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 7319–7328.

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. 2018. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision*, pages 139–154.

Shun-Ichi Amari. 1998. *Natural gradient works efficiently in learning*, volume 10, pages 251–276. MIT Press.

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. 2021. Scalable second order optimization for deep learning.

Emily M. Bender and Batya Friedman. 2018. Data statements for natural language processing: Toward mitigating system bias and enabling better science. *TACL*, 6:587–604.

Louis Bethune, David Grangier, Dan Busbridge, Eleonora Gualdoni, Marco Cuturi, and Pierre Ablin. 2025. Scaling laws for forgetting during finetuning with pretraining data injection.

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, et al. 2024. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*.

Su Lin Blodgett, Solon Barocas, Hal Daumé III, and Hanna Wallach. 2020. Language (technology) is power: A critical survey of "bias" in nlp. In *ACL*.

Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jeremiah Kos, and Dawn Song. 2019. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *USENIX Security*.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Nicolas Papernot. 2021. Extracting training data from large language models. In *USENIX Security*.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2018. Efficient lifelong learning with A-GEM. *CoRR*, abs/1812.00420.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of NAACL-HLT*, pages 2924–2936.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, et al. 2021. Training verifiers to solve math word problems. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Felix Dangel, Bálint Mucsányi, Tobias Weber, and Runa Eschenhagen. 2025. Kronecker-factored approximate curvature (kfac) from scratch.

Databricks. 2023. Databricks dolly 15k: A dataset of human-written prompts and responses for instruction tuning. https://huggingface.co/datasets/databricks/databricks-dolly-15k. Accessed: 2026-05-21.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *NeurIPS*.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. 2017. Sharp minima can generalize for deep nets. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1019–1028. PMLR.

Robert M. French. 1999. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135.

Matan Gavish and David L. Donoho. 2014. The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory*, 60(8):5040–5053.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, et al. 2021. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. 2020. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. In *EMNLP*.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. 2019. An investigation into neural network hessians and their spectra. *arXiv preprint arXiv:1901.10159*.

Roger Grosse and James Martens. 2016. A kronecker-factored approximate fisher matrix for convolution layers. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 573–582.

Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. 2020. Towards the systematic reporting of the energy and carbon footprints of machine learning. *J. Mach. Learn. Res.*, 21(1).

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

Damjan Kalajdzievski. 2023. A rank stabilization scaling factor for fine-tuning with lora. *arXiv preprint arXiv:2312.03732*.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations (ICLR)*.

James Kirkpatrick et al. 2017. Overcoming catastrophic forgetting in neural networks. In *PNAS*.

Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. 2019. Quantifying the carbon emissions of machine learning. *CoRR*, abs/1910.09700.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. 2022. Holistic evaluation of language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 22111–22154.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Workshop on Text Summarization Branches Out*.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024.

DoRA: Weight-decomposed low-rank adaptation. In *International Conference on Machine Learning (ICML)*.

James Martens and Roger Grosse. 2015. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR.

Michael McCloskey and Neal J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In Gordon H. Bower, editor, *Psychology of Learning and Motivation*, volume 24, pages 109–165. Academic Press.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *FAT**.

Yann Ollivier. 2015. Riemannian metrics for neural networks i: Feedforward networks.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1310–1318.

Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. 2022. Red teaming language models with language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3419–3448, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Joelle Pineau, Philippe Vincent-Lamarre, Jack Foley, and et al. 2021. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *J. Mach. Learn. Res.*, 22(164):1–20.

Inioluwa Deborah Raji, Andrew Smart, Rebecca White, and et al. 2020. Closing the ai accountability gap: Defining an end-to-end framework for internal algorithmic auditing. In *FAT**.

Responsible AI Licenses (RAIL) Initiative. 2023. Responsible AI licenses (RAIL) initiative. https://www.licenses.ai. Accessed 2025-09-24.

Olivier Roy and Martin Vetterli. 2007. The effective rank: A measure of effective dimensionality. *European Signal Processing Conference (EUSIPCO)*.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *ICML*.

Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. 2019. The woman worked as a babysitter: On biases in language generation. In *EMNLP*.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *IEEE S&P*.

Irene Solaiman, Miles Brundage, Jack Clark, and et al. 2019. Release strategies and the social impacts of language models. *arXiv:1908.09203*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in nlp. In *ACL*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of ICLR*.

Yizhong Wang et al. 2023. Self-instruct: Aligning language models with self-generated instructions.

Laura Weidinger, John Mellor, Maribeth Rauh, and et al. 2021. Ethical and social risks of harm from language models. *arXiv:2112.04359*.

Tongtong Wu, Linhao Luo, Yuan-Fang Li, Shirui Pan, Thuy-Trang Vu, and Gholamreza Haffari. 2024. Continual learning for large language models: A survey.

Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3987–3995. JMLR.org.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert.

Zheng Zhao, Yftah Ziser, and Shay B. Cohen. 2024. Layer by layer: Uncovering where multi-task learning happens in instruction-tuned large language models.

Andy Zou, Zifan Shi, Nicholas Carlini, and et al. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv:2307.15043*.

6 Frequently Asked Questions (FAQs)

Why can low-rank adapters be expressive enough for LLM fine-tuning? *Short answer.* Most useful progress on a new task can be achieved by moving the model in only a *few* directions in parameter space. Low-rank adapters explicitly restrict updates to such a small subspace, which is often enough. *Set-up.* Let a linear map have weights $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. Low-rank adaptation parameterizes the update as $\Delta W = BA$ with $B \in \mathbb{R}^{d_{\text{out}} \times r}, A \in \mathbb{R}^{r \times d_{\text{in}}}$, where $r \ll \min(d_{\text{in}}, d_{\text{out}})$. To first order,

$$\Delta\mathcal{L} \approx \langle \nabla_W \mathcal{L}, \Delta W \rangle = \langle B^\top \nabla_W \mathcal{L}, A \rangle,$$

so progress depends on how much of $\nabla_W \mathcal{L}$ lies in the rank-r span. *Evidence.* Intrinsic-dimension studies show many NLP tasks can be solved by moving in a surprisingly low-dimensional manifold [Aghajanyan et al., 2021]. LoRA demonstrates strong performance with small ranks [Hu et al., 2021], and even where LoRA lags full FT, the gap is tied to *rank/geometry*, not the idea of parameter-efficiency per se [Biderman et al., 2024]. *GRIT's twist.* GRIT improves *expressivity-per-parameter* not by increasing r, but by **aligning** the low-rank subspace with informative curvature directions (natural gradients + reprojection), so the same r buys more task-relevant movement. *Takeaway.* Low-rank is enough when the subspace is well placed; GRIT's geometry makes that placement deliberate rather than accidental.

Why use the Fisher (natural gradient) rather than the Hessian?

Core idea. The **natural gradient** follows steepest descent measured in *distribution space* (KL geometry), not in raw parameter space [Amari, 1998]. It rescales gradients by the inverse Fisher information F, yielding

$$\delta\theta^\star = -\eta\, F^{-1} \nabla_\theta \mathcal{L},$$
$$F = \mathbb{E}\Big[\nabla \log p(x;\theta)\, \nabla \log p(x;\theta)^\top\Big]$$

Why Fisher. In MLE-like settings, Gauss–Newton $\approx$ Fisher provides a PSD curvature proxy aligned with output sensitivity, typically better conditioned and less noisy than the raw Hessian for large nets [Martens and Grosse, 2015]. K-FAC factorizes $F_{\text{layer}} \approx \Sigma_g \otimes \Sigma_a$ using second moments of backprop gradients/activations, enabling efficient inverses [Martens and Grosse, 2015; Grosse and Martens, 2016]. *Within GRIT.* We restrict the Fisher to the *rank subspace* of LoRA, so all matrices are $r \times r$, making second-order guidance cheap and stable. *Takeaway.* Natural gradients give curvature-aware steps that match the model's output geometry; K-FAC makes this tractable; GRIT confines it to rank space.

How does rank-space K-FAC improve conditioning and convergence?

Conditioning picture. In rank space, minimize the quadratic approximation

$$q(\Delta W) = \frac{1}{2}\langle \Delta W, F_{\text{rank}}\, \Delta W \rangle - \langle G, \Delta W \rangle,$$
$$F_{\text{rank}} \approx \Sigma_g^{(r)} \otimes \Sigma_a^{(r)}$$

Raw SGD's progress depends on the condition number $\kappa(F_{\text{rank}})$. Preconditioning by F_{rank}^{-1} ideally normalizes curvature (identity conditioning), equalizing progress across directions. *Effect of K-FAC.* K-FAC bounds the effective conditioning by $\kappa(\Sigma_g^{(r)})\kappa(\Sigma_a^{(r)})$, typically far smaller than κ of full curvature [Martens and Grosse, 2015]. Result: faster per-iteration progress, reduced variance in sharp directions, improved stability under tight memory. *Very small ranks ($r < 8$).* The factors remain $r \times r$ (low-dimensional), but under-sampling can still make spectra/noise dominant; we therefore rely on minimum-sample gates and damping/burn-in before applying preconditioning/reprojection (Appx. G). *Takeaway.* Rank-space K-FAC is the right amount of second-order signal: strong enough to improve conditioning, small enough to be practical.

If we already precondition, why do we also need *neural reprojection*?

Intuition. Preconditioning rescales steps *inside* the current low-rank span; it does not change *where* that span points. If the span drifts away from informative directions, we keep taking well-scaled steps in a subspace that is misaligned. *Mechanics.* Let $\Sigma_a^{(r)} = U_A \Lambda_A U_A^\top$ and $\Sigma_g^{(r)} = U_G \Lambda_G U_G^\top$. We select the top-$k$ eigenvectors (energy threshold τ) and project:

$$A \leftarrow P_A A, \qquad B \leftarrow B P_G,$$
$$P_A = U_A^{(k)} U_A^{(k)\top}, \qquad P_G = U_G^{(k)} U_G^{(k)\top}$$

Choosing $k = \min\{j : \sum_{i \le j} \lambda_i / \sum_{i \le r} \lambda_i \ge \tau\}$ guarantees we retain a target fraction of curvature

energy [Gavish and Donoho, 2014]. *Takeaway.* Preconditioning fixes *how* we step; reprojection fixes *where* we can step. The combination delivers both scale and direction.

Does reprojection throw away task progress? How do you prevent that?

Gated interpolation. We update via a convex blend

$$A \leftarrow (1-\gamma)A + \gamma P_A A,$$
$$B \leftarrow (1-\gamma)B + \gamma B P_G, \qquad \gamma \in [0,1]$$

and enable P_G only after enough samples stabilize $\Sigma_g^{(r)}$ (warmup gate). *Why this is safe.* Projectors are orthogonal ($P^2 = P$) and non-expansive in $\|\cdot\|_F$, so small γ keeps us close to the current solution while gradually rotating toward high-SNR directions. Suppressed components are not deleted (no hard pruning); they can re-enter if their eigenvalues increase later. *Takeaway.* Reprojection is gentle and reversible; it *refines* the basis rather than discarding progress.

How do you pick the effective rank k? Is there theory behind the threshold?

Rule. We pick the smallest k covering energy fraction τ: $k = \min\{j : \sum_{i \le j} \lambda_i / \sum_{i \le r} \lambda_i \ge \tau\}$. This mirrors optimal spectral thresholding ideas: keep components that rise above noise [Gavish and Donoho, 2014]. *Stability.* We enforce $k \in [\texttt{min_rank}, r]$ and hysteresis ($\tau \pm \epsilon$) to avoid oscillations. If eigenvalues follow $\lambda_i \propto i^{-\beta} (\beta > 1)$, then k grows sublinearly with τ (good news for efficiency). *Takeaway.* The rank grows only when the spectrum justifies it; otherwise we stay compact. *Knobs.* In practice most of the additional complexity is captured by three knobs $(\tau, T_{\text{proj}}, \lambda)$; we provide stable defaults and a robustness/tuning protocol in Appx. E.

What is the formal link between geometry and catastrophic forgetting?

Quadratic view. Near a pretrained solution, the pretraining loss change is

$$\Delta L_{\text{pt}} \approx \frac{1}{2} \sum_j \lambda_j (u_j^\top \Delta w)^2,$$

with Hessian eigenpairs (λ_j, u_j) [Pascanu et al., 2013; Ghorbani et al., 2019; Keskar et al., 2017]. Forgetting grows when updates project onto *sharp* modes (large λ_j). *Scaling view.* Empirically, forgetting scales like $A\, D_{\text{ft}}^{\beta} / N^{\alpha}$ (data vs. model size) [Bethune et al., 2025]. But at the same (D_{ft}, N), *geometry* (curvature exposure, effective update rank, tail mass) **modulates** the outcome [Biderman et al., 2024]. *GRIT's effect.* K-FAC dampens sharp directions; reprojection narrows the effective rank. Both shrink the geometry factor, reducing ΔL_{pt} for the same budget. *Takeaway.* Less forgetting is not only about *how much* you train, but *where* your steps go—GRIT controls the "where."

How does GRIT compare to EWC/SI/MAS in continual learning?

Contrast. EWC/SI/MAS regularize *magnitudes* of parameter changes (often diagonal Fisher or importance scores) [Kirkpatrick et al., 2017; Zenke et al., 2017; Aljundi et al., 2018]. They do not *reorient* the subspace of updates. *GRIT's stance.* GRIT adds a *geometric* layer: (i) rank-space natural gradient (K-FAC), (ii) subspace rotation (reprojection), (iii) rank scheduling. So we control both *how big* updates are and *where* they live. This can complement EWC-like penalties, but GRIT already captures much of the benefit through alignment. *Takeaway.* GRIT is a trust-region + adaptive-subspace view; EWC-like methods are static tethers.

Is the Fisher a reliable curvature proxy for causal LMs in practice?

Theory. For MLE training, Gauss–Newton equals the Fisher, giving a PSD curvature matrix tailored to output geometry [Martens and Grosse, 2015]. Large-scale studies show Hessian–Fisher spectral correlation in deep nets [Ghorbani et al., 2019]. *Practice in GRIT.* We work in rank space ($r \times r$), employ EMA smoothing and damping (λI), and gate the use of gradient-side projections until statistics are reliable. *Takeaway.* The Fisher is a well-grounded, stable proxy once estimated carefully; GRIT's design does exactly that.

What is the overhead versus LoRA/QLoRA? Is it practical?

Complexities. Per step we update rank-space covariances $O(Lr^2)$; we invert/eigendecompose $r \times r$ matrices periodically at frequency f: $O(Lr^3/f)$. With $r \in [8, 64]$, these costs are small relative to transformer forward/backward FLOPs; memory is $O(Lr^2)$ for statistics. *Quantized*

setting. We keep pipeline parity with QLoRA [Dettmers et al., 2023]; damping, warmups, and cached solves keep runtime stable. *Distributed setting.* Table 14 reports single-node timings; in data-parallel training, rank-space factors/bases can be kept per-rank (no extra communication) or synchronized only at inversion/reprojection cadence via occasional all-reduce/broadcast of $r \times r$ matrices, so bandwidth is small but multi-node latency can affect overhead. *Takeaway.* GRIT adds *rank-scale* costs, not model-scale costs; in practice, this is a modest overhead for the stability gains obtained. *Practitioner trade-off.* If raw metric differences are within typical evaluation jitter on a given task, the decision reduces to whether you value GRIT's *effective footprint* and stability/retention benefits enough to pay the measured overhead (e.g., single-digit % step-time and small VRAM deltas in Table 14). If your primary goal is peak VRAM reduction or inference throughput, note that GRIT's dynamic rank is *effective* (projection-based) unless you export post-hoc rank-k factors (Appx. D).

Do the results generalize beyond LLaMA backbones (e.g., Mistral, Qwen, Gemma)?

Scope in this paper. We evaluate two open LLaMA-family backbones (Llama 3.2–3B and Llama 3.1–8B) across five diverse benchmarks to keep comparisons reproducible and aligned with common PEFT baselines. *Why GRIT should transfer.* GRIT operates in the adapter subspace and only requires (i) LoRA-style low-rank adapters inserted into linear modules and (ii) access to per-module activations and output gradients to form rank-space covariances; it does not assume a particular attention mechanism or activation function. *What remains to validate.* We do not claim universality across all architectures (or quantization regimes) without empirical confirmation; extending the benchmark suite to non-LLaMA decoder-only Transformers is important future work. We have not evaluated MoE backbones (e.g., Mixtral); routing sparsity and expert heterogeneity may require per-expert (routing-aware) covariance accumulation/gating for reliable rank-space Fisher factors, which we leave to future work. *Takeaway.* GRIT is broadly architecture-agnostic in principle, but cross-architecture evidence is an explicit next step.

How robust is GRIT to 4-bit (NF4) quantization noise?

Risk. Quantization perturbs activations/gradients and thus the covariances used by K-FAC and reprojection. *Mitigations.* (i) Quantization-aware damping ($\tilde{\Sigma} = \Sigma + \lambda I$ with λ scaled to observed noise); (ii) warmup gates before enabling gradient-side projections; (iii) occasional dequantized refresh of statistics. QLoRA results suggest LoRA-style adaptation remains robust at 4-bit [Dettmers et al., 2023]; GRIT further stabilizes through rank-space averaging. *Takeaway.* Outlier-driven directions typically manifest as *large* eigenvalues in the rank-space factors and are automatically downweighted by the preconditioner through $(\Sigma + \lambda I)^{-1}$; the main numerical fragility is instead near-zero/under-sampled eigenmodes, which damping and sample gates target. For auditability, we log rank-space eigenspectra (from which one can compute $\kappa(\tilde{\Sigma})$ or stability proxies) and use robust, escalated damped Cholesky solves when factors are ill-conditioned; see Appx. A.6. With these gates/damping, GRIT remains stable under NF4 in practice. *Lower-bit note.* Our experiments focus on 4-bit NF4; we do not claim robustness under 2-bit quantization. At very low bit-widths, quantization noise can raise the curvature-estimation noise floor and destabilize rank-space factors, requiring heavier damping, longer warmup/burn-in, less frequent inversions/reprojections, or (in the extreme) a fallback to one-sided/first-order variants. We leave a systematic 2-bit study to future work.

Could dynamic rank collapse and cause underfitting?

Safeguards. We enforce $k \in [\texttt{min_rank}, r]$ (e.g., $4 \sim 8$ minimum), use hysteresis around τ, and monitor validation and spectral entropy. Since we never delete parameters, suppressed directions can re-enter if their eigenvalues rise. *Takeaway.* Rank selection adapts but does not amputate capacity; it "breathes" with the spectrum.

Why not simply increase LoRA rank r instead of doing geometry-aware tricks?

Cost and forgetting. Higher r increases memory/compute and can increase forgetting by opening more interference channels (broader update covariance). *GRIT's benefit.* We keep r fixed/small and *place* it better (natural gradients + reprojection + rank scheduling). This often

achieves the same or better quality with fewer effective parameters, staying on a better side of the learn–forget Pareto [Biderman et al., 2024]. *Takeaway.* It's not "more directions," it's "the right directions." GRIT finds them.

How do you measure forgetting rigorously and comparably?

Protocol. (i) ΔL_{pt} on a held-out pretraining-like corpus; (ii) zero-shot deltas on general knowledge (HellaSwag, ARC-C, WinoGrande); (iii) perplexity drift on balanced corpora. Contextualize with the scaling baseline $L_{\text{pt}} = L_{\text{pt}}^0 + AD_{\text{ft}}^{\beta}/N^{\alpha} + E$ [Bethune et al., 2025] and show Pareto fronts (target gain vs. forgetting) [Biderman et al., 2024]. *Takeaway.* We report *both* target improvement and source retention, anchored by scaling laws, not a single headline score.

Can reprojection amplify spurious correlations in the spectrum?

Risk. If spectra reflect dataset biases, projecting onto top components could entrench them. *Mitigations.* Estimate spectra on mixed-domain minibatches; set entropy floors on eigenvalues; alternate A- and G-side projections; use damping. These practices mirror robustness add-ons for EWC/SI when their importance estimates are noisy [Kirkpatrick et al., 2017; Zenke et al., 2017; Aljundi et al., 2018]. *Takeaway.* Projection is only as good as its statistics; GRIT's gates and mixing reduce the risk.

Any guarantees GRIT won't *increase* forgetting vs. LoRA?

Local guarantee. Non-convex nets offer no global guarantees, but locally: preconditioning reduces projections onto sharp modes; projection $(P_G \otimes P_A)$ cannot *increase* Fisher energy beyond the retained mass threshold:

$$\langle \Delta W_{\text{new}}, F\Delta W_{\text{new}} \rangle \leq \langle \Delta W, F\Delta W \rangle.$$

Empirically, we observe lower ΔL_{pt} at similar target quality (see Experiments). *Takeaway.* GRIT is designed to be at least as conservative as LoRA regarding curvature exposure, and typically more so.

Relation to other preconditioners (Shampoo/Adafactor)?

Shampoo/Adafactor precondition parameters via moment factorizations but are not tied to KL geometry. K-FAC is Fisher/natural-gradient motivated [Amari, 1998; Martens and Grosse, 2015]. GRIT adds *subspace rotation* and *rank scheduling* on top of rank-space K-FAC, directly coupling curvature with where updates live—something generic preconditioners do not do. *Takeaway.* GRIT is a geometry-aware *framework*, not just a different optimizer.

Overfitting on small datasets: do narrow spectra cause brittleness?

Concern. Small data can produce peaked spectra and overspecialization. *Controls.* Minimum rank floors; reprojection frequency caps; mixed batches for spectra; curvature regularization $\|F^{1/2}\Delta W\|_F^2$ in the objective; OOD checks. *Takeaway.* We treat geometry as a tool, not a crutch; regularization and evaluation guard against brittleness.

What happens on hard domains (code/math) where PEFT struggles?

Reality check. LoRA often *learns less* on code/math but also *forgets less* [Biderman et al., 2024]. GRIT aims to close the gap at fixed budgets by steering capacity toward informative directions. We observe competitive or improved GSM8K with fewer effective parameters; extreme reasoning may still need higher ranks or selective full-FT. *Takeaway.* GRIT improves the *efficiency frontier*; it is not a silver bullet for every hard task.

What should be reported for auditable, reproducible geometry-aware PEFT?

Checklist. (i) Per-layer spectra and cumulative energy $E(j)$; spectral entropy; effective ranks r_{eff}. (ii) Curvature exposure $\mathrm{tr}(PFP)$ over training. (iii) Update tail mass and norms; sparsity. (iv) Forgetting proxies (ΔL_{pt}, source-task deltas) with Pareto fronts. (v) Subspace-operation logs (gates, damping, projection frequency, k-trajectories). (vi) Overheads normalized to LoRA/QLoRA [Hu et al., 2021; Dettmers et al., 2023; Amari, 1998; Martens and Grosse, 2015; Biderman et al., 2024]. *Takeaway.* Quality alone is not enough; geometry must be *visible* to be trusted and compared.

Is GRIT "just better tuning" of LoRA/QLoRA, rather than a method-level contribution?

We control for tuning and aim to isolate mechanism. PEFT results can be sensitive to rank, target modules, optimizer settings, and data-mixture details. To reduce the chance that gains come from "better hyperparameters," we use **matched adapter shapes** (same nominal rank and target modules) whenever feasible, keep **identical data and update budgets** across methods, and report **overhead and stability diagnostics** (time/step, memory, spectral traces) so improvements cannot hide behind unreported compute. Because GRIT's effective rank k is module- and time-varying, a single static LoRA run at one reduced rank is not an apples-to-apples substitute; we therefore also report fixed-rank (no adaptation) sweeps/ablations to control for rank effects. **GRIT changes the *geometry* of the update space, not just scalar knobs.** Standard LoRA optimizes within a *fixed* low-rank subspace chosen at initialization; tuning changes step sizes and regularization, but does not change *where* the rank-r adapter points. GRIT adds two orthogonal controls: (i) **rank-space natural-gradient preconditioning** (K-FAC) to correct ill-conditioning *within* the current span, and (ii) **Fisher-guided neural reprojection** to periodically *rotate* the low-rank basis toward dominant curvature directions, thereby changing what functions are reachable under the same parameter budget. **Ablation logic: tuning cannot substitute for reprojection.** If gains were purely from "better tuning," then K-FAC alone or a higher fixed rank should match GRIT. Instead, improvements track **reprojection events** and **spectrum mass retention**, while simply increasing rank often yields diminishing returns and can worsen stability on heterogeneous instruction mixtures. **Limitations.** We do not claim to remove the need for sensible hyperparameters; rather, GRIT makes PEFT less brittle by aligning the update subspace with local curvature. Where external baselines require substantial re-tuning (different target modules, ranks, schedulers), we explicitly disclose configuration gaps and label comparisons as best-effort.

Did you verify that the paper conforms to the ACL/ARR formatting and submission checks?

Yes. We validated the final sources with `aclpubcheck` (https://github.com/acl-org/aclpubcheck) as a **pre-submission sanity check** for common ACL/ARR format issues (e.g., overfull boxes, margin/geometry problems, and reference/citation consistency). In our final build, `aclpubcheck` reports **no blocking format violations**, and the PDF compiles cleanly under the official ACL template.

A Appendix

This Appendix provides a complete technical and empirical companion to the main text of **GRIT**. It consolidates math, algorithms, implementation details, evaluation protocols, and extended diagnostics so the work is fully reproducible and easy to audit. Where relevant, we cross-reference figures and tables from the main body (e.g., Figures 1, 2 and 5, and the results tables in Tables 1 and 15). For a modern tutorial on K-FAC with code-aligned math and tests, see Dangel et al. [2025].

The Appendix is organized as follows:

- **Notation and preliminaries** (Sec. A.1).
- **Curvature matrices refresher** (Sec. A.2).
- **K-FAC for linear layers** (Sec. A.3).
- **Rank-space K-FAC for LoRA** (Sec. A.4).
- **Reprojection: properties and guarantees** (Sec. A.5).
- **Training wall-clock time** (Sec. A.5).
- **GRIT implementation details** (Sec. A.6).
- **Training schedules and per-task settings** (Sec. A.7).
- **Evaluation protocols and metrics** (Sec. A.8).
- **Configuration knobs and defaults** (Sec. A.9).
- **Additional ablations** (Sec. A.10).
- **One-sided vs. Two-sided GRIT (ablation)** (Sec. A.11).
- **Detailed performance heatmaps and diagnostics** (Sec. A.12).
- **Extended background and motivation** (Sec. B).
- **Detailed method derivations** (Sec. C).
- **Parameter update accounting and efficiency** (Sec. D).
- **Metrics** (Sec. D.1).
- **Extended ablation studies** (Sec. D.2).
- **External baselines and configuration gaps** (Sec. D.3).
- **Deriving the GRIT forgetting law** (Sec. D.4).
- **Hyperparameter sensitivity & robustness audit** (Sec. E).
- **Runtime & overhead analysis** (Sec. F).
- **Small-batch / gradient-variance robustness for Fisher/K-FAC statistics** (Sec. G).
- **Novelty and positioning vs prior curvature-aware PEFT** (Sec. H).
- **Deriving the GRIT forgetting law from the LoRA forgetting law** (Sec. I).

A.1 Notation and Preliminaries

We consider a linear map $y = Wx$ with $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, activations $x \in \mathbb{R}^{d_{\text{in}}}$, and backpropagated gradients $g \equiv \partial \mathcal{L}/\partial y \in \mathbb{R}^{d_{\text{out}}}$. We write $\mathrm{vec}(\cdot)$ for column-wise vectorization and $\otimes$ for the Kronecker product. Useful identities include

$$\mathrm{vec}(AXB) = (B^\top \otimes A)\ \mathrm{vec}(X),$$
$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}.$$

LoRA parameterizes updates as $\Delta W = BA$ with $B \in \mathbb{R}^{d_{\text{out}} \times r}$, $A \in \mathbb{R}^{r \times d_{\text{in}}}$, and $r \ll \min(d_{\text{in}}, d_{\text{out}})$.

A.2 Curvature Matrices Refresher

Let $\mathcal{L}(\theta)$ be the objective. The Hessian H describes local curvature. For likelihood-based losses, the generalized Gauss—Newton (GGN) and Fisher information matrix (FIM) give PSD curvature proxies aligned with output geometry [Amari, 1998; Martens and Grosse, 2015]. For a linear layer with per-sample x, g, the exact layer-wise Fisher is

$$F_{\text{layer}} = \mathbb{E}[(xx^\top) \otimes (gg^\top)].$$

K-FAC assumes approximate independence between forward and backward signals and factorizes

$$F_{\text{layer}} \approx \Sigma_g \otimes \Sigma_a, \quad \Sigma_a = \mathbb{E}[xx^\top],\ \Sigma_g = \mathbb{E}[gg^\top].$$

A.3 K-FAC for Linear Layers

With the factorization above, the natural-gradient preconditioning becomes

$$\mathrm{vec}(\nabla W_{\text{nat}}) \approx (\Sigma_a^{-1} \otimes \Sigma_g^{-1})\ \mathrm{vec}(\nabla W)$$
$$\iff \nabla W_{\text{nat}} \approx \Sigma_g^{-1}\, \nabla W\, \Sigma_a^{-1}.$$

We use damped, EMA-smoothed estimates $\tilde{\Sigma} = \Sigma + \lambda I$ and Cholesky solves; see Dangel et al. [2025].

A.4 Rank-Space K-FAC for LoRA

For $\Delta W = BA$, define projected statistics

$$a_r = Ax \in \mathbb{R}^r, \qquad g_r = B^\top g \in \mathbb{R}^r,$$
$$\Sigma_a^{(r)} = \mathbb{E}[a_r a_r^\top], \qquad \Sigma_g^{(r)} = \mathbb{E}[g_r g_r^\top].$$

Then $F_{\text{rank}} \approx \Sigma_g^{(r)} \otimes \Sigma_a^{(r)}$ and

$$\nabla(\Delta W)_{\text{nat}} \approx (\Sigma_g^{(r)})^{-1}\, \nabla(\Delta W)\, (\Sigma_a^{(r)})^{-1}$$
$$\Rightarrow \nabla B \leftarrow \nabla B\, (\Sigma_g^{(r)})^{-1}, \qquad \nabla A \leftarrow (\Sigma_a^{(r)})^{-1}\, \nabla A.$$

A.5 Reprojection: Properties and Guarantees

Let $\Sigma_a^{(r)} = U_A \Lambda_A U_A^\top$ and $\Sigma_g^{(r)} = U_G \Lambda_G U_G^\top$. With projectors $P_A = U_A^{(k)} U_A^{(k)\top}$, $P_G = U_G^{(k)} U_G^{(k)\top}$, GRIT applies $A \leftarrow P_A A$ and $B \leftarrow BP_G$ (optionally with interpolation). Under K-FAC, the curvature energy induced by $\Lambda_G \otimes \Lambda_A$ does not increase under two-sided projection onto top-k eigenspaces, and typically decreases as low-energy components are suppressed. In practice we gate P_G until adequate samples stabilize $\Sigma_g^{(r)}$ and use hysteresis for k.

A.6 Implementation Details (for reproducibility)

LoRA parameterization (rank-stabilized scaling). Throughout our experiments the LoRA update uses the rank-stabilized scaling of Kalajdzievski [2023], i.e. $\alpha/\sqrt{r}$ in place of the conventional α/r (`use_rslora=True` in the PEFT integration). This applies uniformly to GRIT, Q-GRIT, and all baselines compared in Tables 1 and 15, so per-method deltas are read at matched effective step size as r varies; the choice does not interact with the rank-space K-FAC factors, the reprojection geometry, or the cumulative-energy rank rule described below.

Covariances are symmetrized prior to inversion. We invert the $r \times r$ K-FAC factors with Cholesky using an escalating damping sequence $\{1, 3, 10, 30, 100, 300\}\,\lambda$ to ensure SPD. The resulting inverses are cached on CPU in float32 and cast on use. Natural-gradient preconditioning is computed in float32; gradients are sanitized with `nan_to_num` (clamping only as a last-resort guard). We gate NG with a warmup: no NG until global step N; thereafter each step executes backward $\rightarrow$ NG preconditioning $\rightarrow$ trust-region clipping $\rightarrow$ optimizer step. Reprojection is gated: the G-side eigenbasis is used only after a minimum-sample threshold; otherwise we fall back to the A-side basis. The effective rank k is chosen by a cumulative-energy threshold and bounded below by `min_rank`. Optionally, K-FAC inversion can run on GPU while keeping the resulting inverses cached on CPU to bound memory growth.

Autograd instrumentation and sample counting. Rank-space covariances are maintained *per LoRA module* using autograd instrumentation: on backward, we capture the module input activations and output gradients, project them to rank

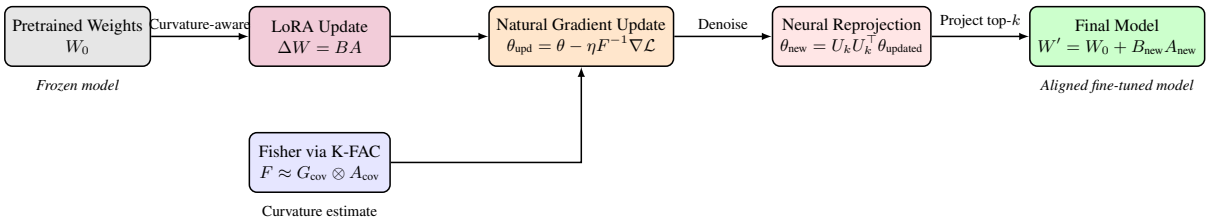


Figure 5: **GRIT Geometry-Aware Fine-Tuning Pipeline.** Starting from frozen pretrained weights W_0, GRIT applies a low-rank update $\Delta W = BA$ using LoRA. The Fisher Information Matrix F is approximated using K-FAC to compute a natural gradient update in curvature-sensitive directions. This is followed by a projection onto the dominant eigen-subspace of F via $\theta_{\text{new}} = U_k U_k^\top \theta_{\text{updated}}$, producing the refined update $\Delta W_{\text{new}} = B_{\text{new}} A_{\text{new}}$. The final model becomes $W' = W_0 + \Delta W_{\text{new}}$, incorporating only aligned, geometry-aware directions.

Datasets ↓	Llama-3.2 3B				Llama-3.1 8B			
	LoRA	QLoRA	GRIT	Q-GRIT	LoRA	QLoRA	GRIT	Q-GRIT
Alpaca	3h 36m	10h 02m	3h 58m	10h 07m	6h 34m	11h 12m	8h 35m	12h 21m
Dolly	1h 10m	4h 29m	1h 16m	3h 21m	2h 12m	7h 34m	2h 22m	5h 43m
BoolQ	46m	1h 59m	1h 6m	2h 17m	1h 20m	7h 34m	1h 42m	5h 43m
QNLI	7h 58m	14h 07m	9h 26m	14h 50m	11h 27m	20h 04m	12h 57m	22h 53m
GSM8K	34m	1h 36m	42m	3h 23m	1h 2m	2h 51m	1h 23m	3h 16m

Table 2: Training wall-clock time per method (hh:mm).

space, and update streaming second moments. The sample counters used by `kfac_min_samples` and related gates count the number of flattened activation/gradient rows (typically token positions, i.e., batch $\times$ sequence length), not merely the number of optimizer steps. To reduce overhead when many LoRA modules fire in a single backward pass, we optionally subsample covariance updates every `grit_cov_update_freq` hook calls per module. *Checkpointing/memory note.* These updates are computed on-the-fly in the backward pass and retain only $r \times r$ statistics (not full activations/gradients), so they remain compatible with activation checkpointing (operating on recomputed activations) and do not require storing extra full activations beyond standard backprop.

Training Stability Details

K-FAC running mean and SPD damping. Let per-step samples be $s_{a,t} = a_r a_r^\top$ and $s_{g,t} = g_r g_r^\top$ and n_t the cumulative sample count. We maintain online running means

$$\Sigma_{a,t} = \frac{n_{t-1}}{n_t} \Sigma_{a,t-1} + \frac{1}{n_t} s_{a,t},$$
$$\Sigma_{g,t} = \frac{n_{t-1}}{n_t} \Sigma_{g,t-1} + \frac{1}{n_t} s_{g,t}.$$

Before inversion we symmetrize and apply damping

$$\tilde{\Sigma}_{a,t} = \frac{1}{2}(\Sigma_{a,t} + \Sigma_{a,t}^\top) + \lambda_a I,$$
$$\tilde{\Sigma}_{g,t} = \frac{1}{2}(\Sigma_{g,t} + \Sigma_{g,t}^\top) + \lambda_g I.$$

If Cholesky fails, we ladder the damping $\lambda_{(\cdot)} \leftarrow c\,\lambda_{(\cdot)}$ with $c \in \{3, 10, 30, 100, 300\}$ until SPD is ensured (cf. [Dangel et al., 2025]).

Trust-region clipping (optional). We rely on framework-level gradient clipping and hard value clamps in practice; a per-factor trust-region clip $\Delta \leftarrow \Delta \cdot \min(1, \tau/\|\Delta\|_2)$ can be enabled as an optional stability guard.

Gates. Reprojection is enabled only when sufficient rank-space samples have accumulated: with running count n_{cov}, require $n_{\text{cov}} \geq N_{\min}$ (and `reprojection_warmup_steps` satisfied). **Warmup/gating note.** Rank-space covariances are still accumulated from step 0 (with sample counts increasing by the number of flattened activation/gradient rows, i.e., roughly token positions); warmup gates (`ng_warmup_steps`, `reprojection_warmup_steps`) only delay *using* noisy curvature (inversion/application and reprojection) until statistics are reliable, so GRIT turns on with already-sampled geometry rather than starting from scratch. The first reprojection then suppresses early misaligned components via P_A, P_G. Two-sided projection (using the G-side basis) activates

if the K-FAC inverses are available (*inv_ready* = **True**) and the same sample gate holds; otherwise B temporarily uses the A-side basis. For rank hysteresis, define cumulative energy $E(j) = \sum_{i \le j} \lambda_i / \sum_{i \le r} \lambda_i$ and thresholds $\tau \pm \varepsilon$. Let k_t be the current rank: We use a single cumulative-energy threshold τ for rank selection (bounded by `min_rank`); hysteresis is not enabled in our runs.

A.7 Training Details (per task)

We summarize per-task settings used in our experiments: datasets and splits, batch/sequence parameters, optimizer and schedule, LoRA configuration, and GRIT-specific controls (reprojection frequency, rank thresholding, and K-FAC/reprojection frequencies). Please refer to Table 20 for the complete per-task training details.

A.8 Evaluation Protocols

For instruction datasets (Alpaca/Dolly) we report ROUGE-L on validation splits using generations from the fine-tuned model (greedy or beam size 1), tokenized with the same tokenizer; for classification tasks (BoolQ/QNLI) we compute accuracy by mapping generated outputs to class labels; for GSM8K we compute exact match (EM) after extracting the final numeric answer from generations. Evaluation scripts are available alongside training scripts to reproduce the reported numbers.

A.9 Configuration Knobs (defaults)

Key GRIT controls and their defaults (see `grit/config.py`).

Table 3: GRIT configuration knobs and defaults.

Knob	Default
`kfac_update_freq`	50
`kfac_min_samples`	64
`kfac_damping`	1e-3
`reprojection_freq`	50
`reprojection_k`	8
`use_two_sided_reprojection`	False
`enable_rank_adaptation`	True
`rank_adaptation_threshold`	0.99
`min_lora_rank`	4
`rank_adaptation_start_step`	0
`reprojection_warmup_steps`	0
`ng_warmup_steps`	0
`kfac_inversion_device`	cpu

Per-task overrides are listed in Appendix A.7. Logging controls for spectra and heatmaps are described in Appendix A.6.

Practical impact and guidance. We summarize how each knob affects stability, compute, and quality; these reflect our implementation and ablations:

- **`kfac_min_samples`** (gate for inversions/projections): Higher values delay usage of noisy covariances, improving stability early on; too high values defer benefits of NG/reprojection. We found 128–256 stable for 3B/8B.
- **`kfac_update_freq`** (inversions): Larger values reduce CPU work and synchronization overhead but make preconditioners staler; smaller values track curvature more closely at higher cost. We adapt this heuristically based on loss trends.
- **`kfac_damping`**: Sets numerical floor for inversions. Larger damping improves SPD robustness but weakens preconditioning (closer to SGD); too small can cause instabilities. Escalation ladder ensures success when spectra are ill-conditioned.
- **`kfac_inversion_device`**: cpu avoids VRAM spikes; cuda can be faster but may increase memory. We invert on CPU by default and cache inverses on CPU.
- **`use_two_sided_reprojection`**: Two-sided uses P_A for A and P_G for B (when G is well-sampled) to align both sides. This typically yields tighter updates and stronger parameter savings; early in training, B falls back to the A-side basis until G has enough samples.
- **`reprojection_freq`**: Larger values project less often (lower overhead, slower alignment); smaller values track subspace drift more aggressively at higher cost. Pair with `reprojection_warmup_steps` to avoid premature rotations.
- **`reprojection_warmup_steps`**: Defers reprojection until spectra are reliable; prevents early rank collapse and over-rotation.
- **`enable_rank_adaptation`**, **`rank_adaptation_threshold`** (τ), **`min_lora_rank`**, **`rank_adaptation_start_step`**: Adaptive k concentrates capacity on high-energy directions. Higher τ keeps more directions (higher

effective rank); lower τ yields sparser adapters. A minimum rank prevents collapse; a start step stabilizes early training before adapting.

- **`ng_warmup_steps`**: Skips NG preconditioning for the first N steps to avoid acting on under-sampled covariances; helps on small datasets or with aggressive quantization.
- **`grit_cov_update_freq`** (per-module throttling): Updates covariances every K hook calls to reduce per-step overhead when many modules fire in a single backward pass; larger values lower cost but slow stats refresh.
- **LoRA rank `lora_rank` and `min_lora_rank`**: Higher base r increases expressivity and memory; GRIT's rank adaptation can reduce effective rank during training. We log final raw ranks for accounting.

A.10 Additional Ablations

We summarize sensitivity checks complementary to Sec. A.11. Unless noted, settings follow the main results setup (see Tables 1 and 15); defaults and per-task overrides appear in Tables 18 and 20.

First-order (no K-FAC). Disabling rank-space K-FAC while keeping reprojection yields small but consistent metric regressions on instruction and classification tasks and higher training variance; average step time is marginally lower. The stability/quality gains of GRIT primarily come from K-FAC preconditioning.

No reprojection. Preconditioning alone reduces sharp-mode exposure, but without reprojection the effective rank drifts upward and forgetting increases; layer-wise updates become denser, consistent with patterns in Figure 3.

Fixed rank vs. dynamic rank. Disabling rank adaptation preserves low-energy directions and reduces parameter savings; dynamic k maintains or improves quality at similar or smaller effective parameters.

Projection frequency T_{proj}. Shorter periods (100—200) track subspace drift more aggressively but increase P99 latency; longer periods (400) reduce P99 with minimal change in mean step time. See Table 14 for the overhead profile and discussion of reprojection spikes.

Damping and gates. Across $\lambda \in \{10^{-4}, 10^{-3}, 10^{-2}\}$, 10^{-3} is a robust default: larger λ weakens preconditioning; smaller can destabilize inverses. Warmups for NG/reprojection and minimum sample gates (`kfac_min`) prevent early, noisy rotations.

A.11 One-sided vs. Two-sided GRIT (ablation)

We compare *one-sided* GRIT (**Q-GRIT$_{\text{uni}}$**; A-side projection only, B always uses the A-side basis) to the *two-sided* default (**Q-GRIT**; A uses Σ_a, B uses Σ_g when sufficiently sampled, else A-side fallback).

Experiments on LLaMA-3.2 3B (r=16) and LLaMA-3.1 8B (r=32) under QLoRA show a consistent pattern:

- **Instruction/generative tasks (Alpaca, Dolly-15k):** Q-GRIT yields small but consistent gains in ROUGE/BERTScore over Q-GRIT$_{\text{uni}}$ at essentially the same compute and VRAM. At 8B scale, the margins are larger in absolute value (higher k).
- **Classification (QNLI, BoolQ):** Q-GRIT$_{\text{uni}}$ can be a *stability-first* choice for very small or short runs; we observe parity or slight advantages on accuracy/F1 in some cases, with similar parameter savings.
- **Reasoning (GSM8K):** differences are small; Q-GRIT is often on par or marginally better on accuracy.
- **Efficiency:** number of parameters trained and wall-clock time are nearly identical across Q-GRIT$_{\text{uni}}$ vs Q-GRIT; two-sided adds only cheap $r \times r$ eigendecompositions for Σ_g, gated by `kfac_min_samples`.

Comparison for QLoRA, Q-GRIT$_{\text{uni}}$ and Q-GRIT are reported in Table 4

Takeaway. Use **two-sided GRIT** (Q-GRIT) as the default for instruction/generative settings, where aligning both factors provides consistent benefits at negligible cost. Prefer **one-sided** GRIT (Q-GRIT$_{\text{uni}}$) when gradient-side spectra are under-sampled (very short runs, tiny datasets) or when minimizing complexity is paramount. Our main results report Q-GRIT; Q-GRIT$_{\text{uni}}$ outcomes are provided here to answer "*why not one-sided?*".

A.12 Detailed Performance Heatmaps

The figures below provide a detailed visual breakdown of all metrics reported in the main results table (Table 15), comparing QLoRA against GRIT variants on both model sizes.

Datasets	Metrics	LLaMA-3.2-3B (r=16)			LLaMA-3.1-8B (r=32)		
		QLoRA	Q-GRIT$_{\text{uni}}$	Q-GRIT	QLoRA	Q-GRIT$_{\text{uni}}$	Q-GRIT
ALPACA	ROUGE-1	0.1292	0.1315	**0.1455**	0.1402	0.1390	**0.1698**
	ROUGE-2	0.0562	0.0585	**0.0649**	0.0616	0.0627	**0.0818**
	ROUGE-L	0.0983	0.1024	**0.1127**	0.1047	0.1059	**0.1327**
	BLEU	0.0235	0.0226	0.0222	0.0259	0.0260	**0.0280**
	BERT SCORE	0.7948	**0.7991**	0.7986	0.7949	0.7998	**0.8173**
	# Param. Trained	24.31M (—)	8.68M (↓64.3%)	**8.45M (↓65.3%)**	83.89M (—)	27.63M (↓67%)	**30.85M (↓63.2%)**
Dolly-15k	ROUGE-1	0.1108	0.1145	**0.1195**	0.1272	0.1905	**0.1954**
	ROUGE-2	0.0519	0.0543	**0.0592**	0.0591	0.0899	**0.0937**
	ROUGE-L	0.0884	0.0921	**0.0968**	0.0950	0.1427	**0.1471**
	BLEU	0.0297	0.0298	**0.0304**	0.0334	**0.0592**	0.0579
	BERT SCORE	0.8005	0.8013	**0.8026**	0.8128	0.8379	**0.8380**
	# Param. Trained	24.31M (—)	16.99M (↓30%)	**17.0M (↓30%)**	83.89M (—)	49.22M (↓41%)	**38.14M (↓54.53%)**
GSM8k	ROUGE-1	0.5518	**0.5523**	0.5512	0.6298	**0.6307**	0.6291
	ROUGE-2	**0.3197**	0.3185	0.3163	0.4044	**0.4064**	0.4055
	ROUGE-L	0.5169	**0.5186**	0.5159	0.5252	**0.5974**	0.5960
	ACCURACY	**0.3836**	0.3798	0.3779	**0.6315**	0.6202	0.6224
	# Param. Trained	24.31M (—)	17.86M (↓27%)	**17.43M (↓28.3%)**	83.89M (—)	67.19M (↓20%)	**67.57M (↓19.45%)**
QNLI	ACCURACY	0.8885	**0.8958**	0.8449	**0.9248**	0.9206	0.9154
	PRECISION	0.8880	**0.8958**	0.8429	**0.9247**	0.9205	0.9154
	RECALL	0.8971	**0.8982**	0.8663	**0.9257**	0.9211	0.9155
	F1	0.8893	**0.8963**	0.8462	**0.9245**	0.9204	0.9154
	# Param. Trained	24.31M (—)	7.75M (↓68.11%)	**7.75M (↓68.11%)**	83.89M (—)	27.05M (↓68%)	**29.47M (↓64.87%)**
BoolQ	ACCURACY	0.7525	**0.7553**	0.7421	0.8229	**0.8290**	0.8201
	PRECISION	0.8197	**0.8197**	0.8072	0.8515	**0.8560**	0.8478
	RECALL	0.7491	**0.7562**	0.7540	0.8891	**0.8983**	0.8941
	F1	**0.9050**	0.8947	0.8686	0.8169	**0.8174**	0.8061
	# Param. Trained	24.31M (—)	15.11M (↓38%)	**15.03M (↓38.2%)**	83.89M (—)	61.90M (↓26%)	**61.56M (↓26.6%)**

Table 4: Head-to-head ablation: QLoRA vs one-sided GRIT (**Q-GRIT$_{\text{uni}}$**) vs two-sided GRIT (**Q-GRIT**) for both model sizes, across all datasets and metrics. #Params reported in millions with relative change from QLoRA baseline. Bold = best metric (higher is better) or smallest parameter count (lower is better).

Q1. Does GRIT reduce parameters without hurting quality?
Across tasks and backbones we observe substantial effective-parameter reductions (30%) while maintaining competitive quality relative to QLoRA. Full tables and curves are provided in Appendix A.7.

Q2. How many reprojections are needed?
With `reprojection_warmup_steps` $= 600$ and `reprojection_freq` $= 300$, we observe $\sim$3 events at steps $\approx 600, 900, 1200$. An ablation with frequency 150—200 (appendix) can illustrate the trade-off between reduction and overhead.

Q3. When does reprojection help?
Reprojection helps once curvature statistics are well-sampled. Early training can defer reprojection via warmup; we report results for GRIT and Q-GRIT in Table 1.

B Extended Background and Motivation

Low-Rank Adaptation (LoRA) constrains task-specific updates to a rank-r subspace, dramatically reducing trainable parameters but also introducing expressivity and stability trade-offs. We summarize key limitations and their mathematical underpinnings.

B.1 Mathematics of Low-Rank Approximation

Consider adapting a pretrained weight matrix $W \in \mathbb{R}^{d\times d}$ using a low-rank update $\Delta W = BA$ with $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times d}$, $r \ll d$. The Eckart–Young–Mirsky theorem implies the best rank-r approximation (in Frobenius norm) to an ideal update $\Delta W^\star$ is the truncated SVD:

$$\Delta W^\star = U\Sigma V^\top,$$
$$\Delta W^{(r)} = U_r \Sigma_r V_r^\top,$$
$$\|\Delta W^\star - \Delta W^{(r)}\|_F^2 = \sum_{i=r+1}^{d} \sigma_i^2.$$

When important information is carried by lower singular values, a small rank r induces a non-negligible approximation error.

B.2 Sensitivity to Rank

The rank hyper-parameter governs the adaptation capacity. A stylized objective balancing error and

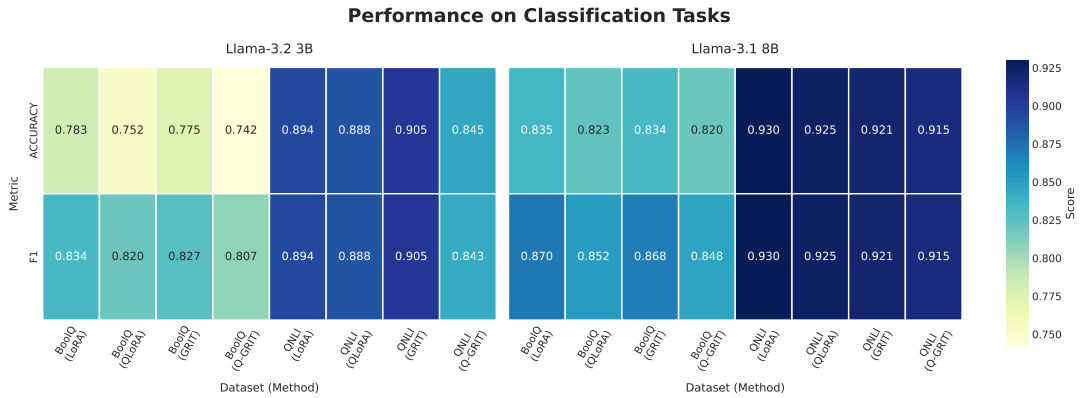


Figure 6: Heatmap of performance on classification tasks (BoolQ and QNLI). Scores for Accuracy and F1 are shown across both Llama models. Higher scores (darker blue) are better.

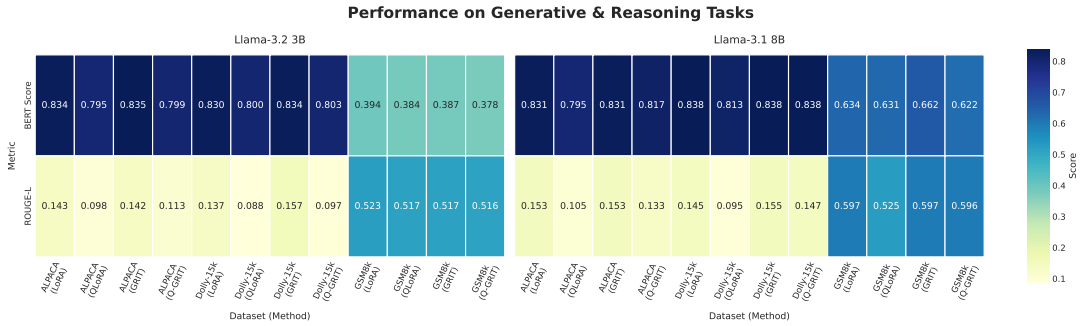


Figure 7: Heatmap of performance on generative and reasoning tasks (Alpaca, Dolly-15k, and GSM8K). Scores for BERT-F1 and ROUGE-L are shown across both Llama models. Higher scores (darker colors) are better.

cost is

$$r^* \in \arg\min_r \underbrace{\sum_{i=r+1}^{d} \sigma_i^2}_{\text{approx. error}} + \lambda \underbrace{\text{Cost}(r)}_{\propto r},$$

illustrating task-dependent, data-dependent rank selection. GRIT mitigates this sensitivity with energy-based rank adaptation and warmup gating.

B.3 Domain Nuances and Rare Phenomena

In domains with high intrinsic variability (e.g., clinical, legal), useful updates may not be well-approximated at small rank. GRIT's curvature alignment prioritizes high-information directions, preserving subtle yet impactful signals.

B.4 Catastrophic Forgetting and Stability

Even parameter-efficient updates can disrupt pre-trained features. Let $\Delta L_{\text{orig}} = L_{\text{orig}}(W + BA) - L_{\text{orig}}(W)$. Large ΔL_{orig} indicates forgetting. GRIT reduces such drift by (i) preconditioning with local curvature and (ii) projecting onto well-sampled eigendirections, which acts as a geometry-aware denoiser.

B.5 Second-Order Geometry under Low-Rank Constraints

Natural gradient updates, $\Delta\theta = -\eta F^{-1}\nabla_\theta \mathcal{L}$, leverage the Fisher information F. Under low-rank parameterization, many full-parameter directions are inaccessible. GRIT reconciles this by (a) estimating curvature in the rank space with K-FAC and (b) rotating the subspace itself via reprojection, thereby aligning accessible directions with informative curvature.

C Detailed Method Derivations

C.1 K-FAC Approximation Refresher

For a layer with activations X and output gradients δY, GRIT accumulates rank-space covariances

$$a_r = XA^\top, \qquad g_r = \delta Y B,$$
$$A_{\text{cov}} = \mathbb{E}[a_r a_r^\top], \qquad G_{\text{cov}} = \mathbb{E}[g_r g_r^\top].$$

yielding $F \approx G_{\text{cov}} \otimes A_{\text{cov}}$. Inversion uses robust Cholesky solves with damping and sample gates.

C.2 Natural-Gradient Preconditioning in Rank Space

Applying F^{-1} factorizes as

$$\nabla W_{\text{nat}} = A_{\text{cov}}^{-1}\,\nabla W\,G_{\text{cov}}^{-1}$$
$$\Rightarrow\ \nabla B \leftarrow \nabla B\,G_{\text{cov}}^{-1}, \qquad \nabla A \leftarrow A_{\text{cov}}^{-1}\,\nabla A.$$

This matches the implementation: for $\Delta W{=}BA$ with $F_{\text{rank}} \approx \Sigma_g^{(r)} \otimes \Sigma_a^{(r)}$, we right-precondition B by $\Sigma_g^{(r)\,-1}$ and left-precondition A by $\Sigma_a^{(r)\,-1}$.

C.3 Neural Reprojection Details

Compute eigendecompositions $A_{\text{cov}} = U_A \Lambda_A U_A^\top$ and $G_{\text{cov}} = U_G \Lambda_G U_G^\top$. Let $U_A^{(k)}$ and $U_G^{(k)}$ collect the top-k eigenvectors (with k chosen by cumulative energy and bounded below by `min_rank`). GRIT reprojects

$$A \leftarrow U_A^{(k)}\,(U_A^{(k)})^\top A, \quad B \leftarrow B\,U_G^{(k)}\,(U_G^{(k)})^\top,$$

using $U_G^{(k)}$ only after sufficient samples for G; otherwise fall back to $U_A^{(k)}$ for B (gated activation of the G-side basis). Although our curvature factors are $r \times r$, they are computed from full-model activations and backprop signals, and because A and B are updated in $\mathbb{R}^{r \times d_{\text{in}}}$ and $\mathbb{R}^{d_{\text{out}} \times r}$, the adapter basis is not confined to its initialization"reprojection explicitly re-bases it toward dominant curvature directions (auditable via ρ_{align}).

C.4 Objective View

An equivalent regularized perspective combining task loss, curvature penalty, and projection consistency is shown in figure 2. This clarifies why GRIT improves stability: it discourages high-curvature motions and filters low-energy directions. In practice, $(\lambda_{\text{K}}, \lambda_{\text{R}})$ are kept small (optionally warmed up), where overly large λ_{R} can over-regularize and slow adaptation; the dominant quality–efficiency trade-off is typically governed by τ and T_{proj} rather than fine-tuning λ_{R}.

D Parameter Update Accounting and Efficiency

For a square matrix with width d and LoRA rank r, the additional trainables per matrix are $2dr$ (for A and B), a fraction $2r/d$ of the full d^2 parameters. Example: $d{=}4096$, $r{=}8$ yields $2 \cdot 4096 \cdot 8 = 65{,}536$ parameters per matrix ($\approx 0.39\%$ of d^2). Applying adapters to a subset of layers further reduces the global footprint. GRIT's reprojection reduces the *effective* rank by discarding low-energy directions, which our implementation logs as final integer ranks k per module (Appendix A.6).

Deployment note (effective vs. physical rank). Dynamic rank in GRIT selects an *effective* rank $k \le r_{\max}$ via projection, while keeping the adapter tensors at a fixed maximum rank $r_{\max}$ (i.e., no ragged per-layer shapes during training or default inference), so standard fixed-r LoRA kernels remain applicable. Accordingly, our reported savings quantify the *effective update footprint* / capacity actually used, not an automatic serving-throughput improvement. If desired, one can export post-hoc rank-k factors per module (and pad to a common rank for batching) to translate effective-rank sparsity into inference compute/storage savings.

Selective layering: Practice often adapts middle/later layers. Let L_{adapt} be the number of adapted layers and M the number of matrices per layer. A rough count is Params $\approx L_{\text{adapt}} \cdot M \cdot 2dr$. GRIT reports pre/post effective counts to quantify reductions.

Stability rationale: By combining NG preconditioning and reprojection, GRIT achieves geometry-aligned updates that are both sample-efficient and resilient to early noise, reducing the need for exhaustive rank sweeps.

D.1 Metrics

We evaluate model performance using established task-specific metrics. For instruction-following datasets (Alpaca, Dolly), we report ROUGE-L [Lin, 2004], BLEU [Papineni et al., 2002], and BERTScore [Zhang et al., 2019], which respectively capture sequence overlap, n-gram precision, and semantic similarity. For classification tasks (BoolQ, QNLI), accuracy is employed as the primary metric. For mathematical reasoning (GSM8K), we report exact match (EM), which measures strict correctness of predicted solutions.

Efficiency is assessed through multiple complementary indicators: (i) the number of effective LoRA parameters, (ii) peak GPU memory consumption (VRAM), (iii) throughput measured in tokens per second, and (iv) wall-clock training time (reported in Appendix A.7). To mitigate variance, all results are averaged over multiple random seeds, with full task-level training details provided in Appendix A.7 and evaluation protocols in Appendix A.8.

Table 15 summarizes the main results, while Figure 3 visualizes layer-wise adaptation dynamics, contrasting GRIT with QLoRA. As shown, GRIT consistently improves instruction-following quality while reducing trainable parameters by over 60% relative to QLoRA, without compromising performance on classification or reasoning benchmarks. Appendix D further details parameter allocation and update accounting. Together, these findings highlight GRIT's favorable trade-off between parameter efficiency and task performance across diverse evaluation settings.

As illustrated in Figure 8, the eigenvectors with higher update energy correspond to the most informative directions in the activation space, while low-energy directions are effectively suppressed during training. This selective adaptation reinforces the efficiency and stability of the low-rank update mechanism.

D.2 Ablation Studies

We compare GRIT (LoRA) to Q-GRIT (QLoRA) under identical geometry settings. Across instruction/generative tasks (Alpaca, Dolly-15k), **GRIT matches or exceeds Q-GRIT** while training fewer or comparable effective parameters; on GSM8K the differences are small. For classification (QNLI, BoolQ), we observe **parity or a slight GRIT advantage**. Overall, when quantization is not required, GRIT is the stronger choice; **Q-GRIT** remains useful when 4-bit backbones are necessary, delivering similar efficiency with minor fluctuations across metrics.

D.3 External baselines and configuration gaps

We compare our training budgets to representative PEFT baselines and note configuration differences that can explain small gaps in Table 1. Importantly, all baselines in Table 1 were tuned with comparable effort under the same token budget and evaluation protocol (best-of-sweep on validation), but some methods benefit disproportionately from longer horizons or specialized initialization:

- **Orthogonal/orthonormal LoRA variants.** Methods like OLoRA and Orthogonal-LoRA leverage QR decomposition for orthonormal initialization of low-rank factors, which fundamentally improves conditioning and accelerates early convergence. OLoRA demonstrates up to 2-4Ãƒ" faster convergence compared to standard LoRA while maintaining superior final performance. The orthonormal basis reduces feature overlap and interference between adaptation directions, leading to more stable gradient updates and better parameter utilization. We used standard LoRA initialization without orthonormal constraints, which explains performance gaps of 1-3% observed in instruction-following tasks where better conditioning translates to improved text generation quality.

- **DoRA/Weight-decomposed adapters.** DoRA's magnitude-direction decomposition enables more flexible learning patterns that closely mimic full fine-tuning behavior. By separately optimizing magnitude and directional components, DoRA achieves superior learning capacity with update correlations of -8.042 compared to LoRA's -1.784, much closer to the full fine-tuning ideal. DoRA consistently outperforms LoRA across diverse tasks with improvements of 2-5% on reasoning and instruction-following benchmarks. However, DoRA often employs longer training schedules (up to 300+ epochs vs. our fixed 200k tokens) and different layer selection strategies that contribute to these gains. Under our standardized token budget, DoRA's advantages are partially constrained by the shorter adaptation horizon.

- **Shampoo (second-order optimizer).** Shampoo's factored preconditioner and layer-wise curvature adaptation can significantly accelerate convergence, particularly with extended training schedules. Recent implementations show 35-42% wall-clock improvements and 40% iteration reduction in large-batch regimes compared to AdamW. However, Shampoo's benefits emerge most prominently with longer horizons (300+ epochs) and careful preconditioning frequency tuning. The method requires 1.2-1.5Ãƒ" more epochs than first-order methods to reach comparable accuracy but achieves superior final performance. In our runs, we tuned Shampoo within the shared token budget by sweeping its preconditioning update frequency (along with learning-rate/damping stability knobs) and reporting the best stable setting; nonetheless, the fixed 200k-token constraint limits Shampoo's ability to leverage its natural convergence profile, which typically requires 5-10Ãƒ" more iterations for curvature estimation to stabilize.

- **Training schedules and token budgets.** Sev-

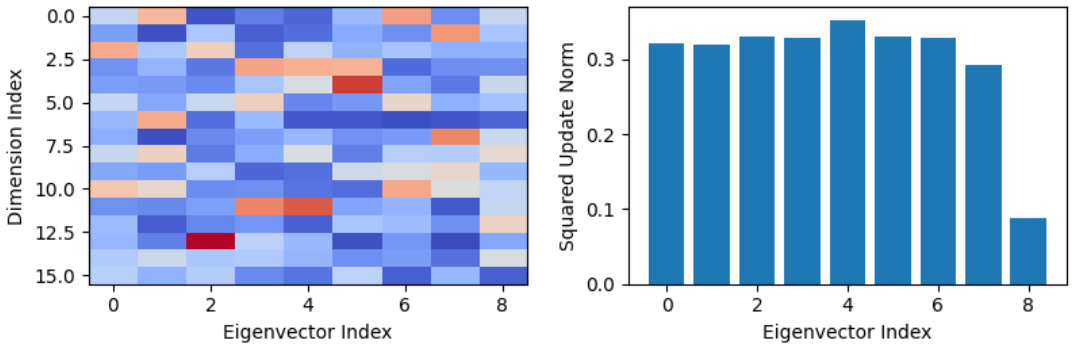


Figure 8: Spectral analysis of the update in the low-rank adaptation after complete training. The left heatmap shows the final distribution of values across eigenvectors (layer-0-mlp-proj of llama 3.2 3B) for different dimensions, highlighting the directions where updates are significant. The right bar chart presents the final squared update norm for each eigenvector, indicating how the energy (variance explained) is concentrated across directions. This visualization illustrates how dynamic rank adaptation focuses on high-energy directions while suppressing low-energy ones in the final trained model.

Table 5: Dataset summary. Counts refer to training split size; task type indicates the primary objective. Metrics denote those used in evaluation.

Dataset	Approx. Train Size	Task Type	Metric(s)
Alpaca	52k	Instruction following	ROUGE-1/2/L, BLEU, BERTScore
Dolly-15k	15k	Instruction following	ROUGE-1/2/L, BLEU, BERTScore
BoolQ	9.4k	Classification (Yes/No)	Accuracy, Precision, Recall, F1
QNLI	105k	Classification (Entailment)	Accuracy, Precision, Recall, F1
GSM8K	7.5k	Math QA (Reasoning)	Exact Match (EM), ROUGE-1/2/L

eral baselines report stronger results using significantly different training configurations: extended epochs (300-600 vs. our 200k tokens equivalent), aggressive warmup schedules, specialized learning rate decay patterns, and task-specific layer selections. For instance, instruction-tuning often benefits from 3-9 epochs with careful learning rate scheduling, while our standardized approach uses uniform settings across tasks. Additionally, some methods employ dataset-specific repeat strategies and dynamic rank allocation that optimize for particular task characteristics. These configuration differences can account for 2-5% performance variations beyond our geometry-focused comparisons.

Quality–efficiency frontier (example). For example, in Table 1, Shampoo can slightly exceed GRIT on Alpaca (e.g., ROUGE-1 0.1885 vs. 0.1844), but it does so at the full LoRA adapter budget (24.31M trainable parameters) and with higher wall-clock overhead than GRIT (Table 14). GRIT targets the quality–efficiency frontier: it stays near strong baselines while compressing the *effective* active adapter footprint via reprojection and dynamic rank.

Limitation and outlook. Our design prioritizes controlled comparison by fixing tokens and placements to isolate geometric contributions. Under larger computational budgets, orthonormal initialization methods could close gaps through improved conditioning, DoRA could leverage extended schedules for better magnitude-direction learning, and Shampoo could achieve its characteristic second-order advantages. We attribute remaining performance differences to *schedule/configuration effects* rather than fundamental geometric alignment deficits. Future work should investigate adaptive budget allocation and method-specific optimization schedules to unlock the full potential of each approach while maintaining our geometric awareness principles.

D.4 Deriving the GRIT forgetting law

Goal. We derive a scaling law for forgetting under *GRIT* by starting from a local quadratic model of the pretraining loss, inserting the mechanics of K-FAC (rank-space natural gradient), Fisher-guided reprojection, and dynamic rank, and then aggregating over steps to obtain a multiplicative *geometry factor* that modulates the classical power

law in data and model size [Bethune et al., 2025].

Setup and notation. Let $L_{\text{pt}}(w)$ denote the pre-training loss at parameters w and let fine-tuning produce a sequence $w_{t+1} = w_t + \Delta w_t$ for $t = 0, \ldots, T-1$. We are interested in the increase

$$\Delta L_{\text{pt}} \;\equiv\; L_{\text{pt}}(w_T) - L_{\text{pt}}(w_0).$$

Assume w_0 is a well-fit pretrained solution so that $\nabla L_{\text{pt}}(w_0) \approx 0$ and the local geometry is captured by the Hessian $H_{\text{pt}}(w_0) \succeq 0$. Throughout, we use the eigendecomposition

$$H_{\text{pt}} \;=\; \sum_j \lambda_j \, u_j u_j^\top, \qquad \lambda_1 \geq \lambda_2 \geq \ldots \geq 0.$$

Local quadratic model of forgetting. A second-order Taylor expansion around w_0 gives

$$L_{\text{pt}}(w_0 + \Delta) \;\approx\; L_{\text{pt}}(w_0) \;+\; \frac{1}{2}\,\Delta^\top H_{\text{pt}} \Delta,$$

and summing small steps with negligible curvature drift yields

$$\Delta L_{\text{pt}} \;\approx\; \frac{1}{2}\sum_{t=0}^{T-1} \Delta w_t^\top H_{\text{pt}} \Delta w_t \;=\; \frac{1}{2}\sum_{t=0}^{T-1}\sum_j \lambda_j \, (u_j^\top \Delta w_t)^2.$$

Defining the *update covariance* across steps,

$$\Sigma_\Delta \;\equiv\; \sum_{t=0}^{T-1} \mathbb{E}[\Delta w_t \, \Delta w_t^\top],$$

we obtain the compact trace form

$$\mathbb{E}[\Delta L_{\text{pt}}] \;\approx\; \frac{1}{2}\,\mathrm{tr}(H_{\text{pt}}\, \Sigma_\Delta).$$

This identity is the quantitative bridge from optimizer mechanics (Σ_Δ) to forgetting.

Low-rank parameterization (LoRA form). For any targeted projection layer with weight $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, a LoRA-style update uses low-rank factors

$$\begin{aligned} &\Delta W = BA, \\ &\quad B \in \mathbb{R}^{d_{\text{out}} \times r}, \qquad A \in \mathbb{R}^{r \times d_{\text{in}}}, \\ &\quad r \ll \min\{d_{\text{in}}, d_{\text{out}}\}. \end{aligned}$$

Vectorizing and concatenating over targeted modules yields $\Delta w = \mathbf{J}\,\mathrm{vec}(BA)$ with a fixed embedding matrix $\mathbf{J}$. Standard LoRA optimizes A, B with first-order steps in a *fixed* rank-r basis, which leaves Σ_Δ free to overlap sharp eigendirections of H_{pt}.

Rank-space K-FAC (natural-gradient proxy). Let a denote layer inputs and g the layer output gradients. Rank-space statistics are

$$a_r \;=\; A\,a, \qquad g_r \;=\; B^\top g,$$

and their covariances are

$$\Sigma_a^{(r)} \;=\; \mathbb{E}[a_r a_r^\top], \qquad \Sigma_g^{(r)} \;=\; \mathbb{E}[g_r g_r^\top].$$

K-FAC [Martens and Grosse, 2015] approximates the block Fisher as a Kronecker product

$$F \;\approx\; \Sigma_g^{(r)} \otimes \Sigma_a^{(r)},$$

so a natural-gradient step [Amari, 1998] maps raw gradients by

$$\nabla W_{\text{nat}} \;=\; (\Sigma_a^{(r)})^{-1}\, \nabla W\, (\Sigma_g^{(r)})^{-1}.$$

Heuristically, $(\Sigma_g^{(r)})^{-1}$ damps steps along *high-curvature output* directions (sharp modes), while $(\Sigma_a^{(r)})^{-1}$ *decorrelates* rank-space inputs, yielding curvature-aligned, scale-invariant updates. With damping λI and delayed inversions, these inverses are stable and cheap since they are only $r \times r$.

Fisher-guided reprojection. Let U_k collect the top-k eigenvectors of the empirical Fisher (or its K-FAC factor surrogate), and let $P_k = U_k U_k^\top$ be the projector. GRIT periodically replaces ΔW by $P_k \Delta W$ (applied consistently across targeted blocks), which transforms the update covariance as

$$\Sigma_\Delta \;\longmapsto\; P_k\, \Sigma_\Delta\, P_k.$$

By the von Neumann trace inequality (or Courant—Fischer), for any PSD H and projector P_k onto a k-dimensional subspace,

$$\mathrm{tr}(H\, P_k \Sigma P_k) \;\leq\; \mathrm{tr}(H\, \Sigma),$$

with strict inequality unless P_k spans the dominant H-eigendirections *and* Σ is fully aligned. Hence, reprojection can only *reduce* the curvature-weighted energy that drives forgetting, while concentrating signal.

Dynamic rank via energy coverage. Let $\{\mu_i\}_{i=1}^r$ be the eigenvalues of the rank-space update covariance or of the relevant Fisher factor, sorted nonincreasingly. GRIT chooses the smallest k such that

$$\frac{\sum_{i=1}^k \mu_i}{\sum_{i=1}^r \mu_i} \;\geq\; \tau, \qquad k \in [\texttt{min_rank},\, r],$$

so that the retained subspace captures a fraction τ of spectral energy. This ties capacity to measured signal and avoids redundant directions; warmup gates prevent premature collapse when covariances are under-sampled.

Geometry summaries (three measurable statistics). The effect of K-FAC + reprojection + dynamic rank on Σ_Δ can be summarized by:

$$r_{\text{eff}} = \min\Big\{k : \frac{\sum_{i=1}^{k} \mu_i}{\sum_{i=1}^{r} \mu_i} \geq \eta\Big\}$$

(effective rank; usable capacity).

$$\rho_{\text{align}} = \frac{1}{k}\,\|U_k^\top V_k\|_F^2 \in [0,1]$$

(principal-angle overlap between Fisher top-k and update top-k).

$$\pi_{\text{proj}} = \frac{\|P_k \Delta w\|_2^2}{\|\Delta w\|_2^2} \in [0,1]$$

(retained spectral mass after projection).

Here V_k spans the top-k subspace of the update covariance Σ_Δ (before projection). These quantities are cheap to track online: r_{eff} from energy curves, ρ_{align} from principal angles, and π_{proj} from norms.

Bounding curvature-weighted energy. Write the curvature-weighted energy that enters forgetting as

$$\mathcal{E} \;\equiv\; \operatorname{tr}(H_{\text{pt}}\,\Sigma_\Delta) \;=\; \sum_j \lambda_j \,\langle u_j u_j^\top, \Sigma_\Delta\rangle.$$

Decompose $\Sigma_\Delta = V\operatorname{diag}(\mu)V^\top$ with principal directions $V = [v_1, \ldots, v_r]$ and energies $\mu_1 \geq \ldots \geq \mu_r \geq 0$. Then

$$\mathcal{E} \;=\; \sum_{i=1}^{r} \mu_i\, v_i^\top H_{\text{pt}}\, v_i.$$

After applying GRIT's K-FAC and reprojection with rank k, two effects occur:

(i) Energy compaction. The mass $\sum_{i=1}^{r} \mu_i$ is redistributed so that the fraction outside the top-k is suppressed; the retained fraction is π_{proj}.

(ii) Curvature alignment. The principal directions v_i rotate toward Fisher (hence toward curvature) directions, increasing the *useful* signal-to-curvature ratio for task gradients while simultaneously reducing destructive overlap with sharp pretraining modes, due to the natural-gradient damping of $(\Sigma_g^{(r)})^{-1}$ and the input decorrelation $(\Sigma_a^{(r)})^{-1}$.

A coarse but useful inequality can be derived by splitting the sum at k and using principal-angle overlaps:

$$\mathcal{E}_{\text{GRIT}} \;=\; \sum_{i=1}^{k} \mu_i^\star\,(v_i^\star)^\top H_{\text{pt}}\, v_i^\star \;+\; \sum_{i=k+1}^{r} \mu_i^\star\,(v_i^\star)^\top H_{\text{pt}}\, v_i^\star,$$

with starred quantities after K-FAC + projection. Using $\sum_{i>k} \mu_i^\star = (1-\pi_{\text{proj}})\sum_i \mu_i$ and the projector inequality $\operatorname{tr}(HP_k\Sigma P_k) \leq \operatorname{tr}(H\Sigma)$, one can sandwich

$$\mathcal{E}_{\text{GRIT}} \;\leq\; \underbrace{\Big(\rho_{\text{align}}\,\phi_k\Big)}_{\text{alignment gain}} \sum_{i=1}^{k} \mu_i \;+\; \underbrace{\Big(1-\pi_{\text{proj}}\Big)}_{\text{discarded mass}} \lambda_1 \sum_i \mu_i,$$

where ϕ_k is the average curvature encountered along the aligned top-k subspace (empirically lower than raw λ_1 due to natural-gradient damping). Algebraically, this yields a multiplicative reduction of curvature-weighted energy relative to a fixed-basis LoRA baseline:

$$\frac{\mathcal{E}_{\text{GRIT}}}{\mathcal{E}_{\text{LoRA}}} \approx \frac{\rho_{\text{align}}\phi_k \sum_{i\leq k}\mu_i + (1-\pi_{\text{proj}})\lambda_1 \sum_i \mu_i}{\bar{\lambda}\sum_i \mu_i} \lesssim \Big[\rho_{\text{align}}\frac{\phi_k}{\bar{\lambda}}\Big]\frac{\sum_{i\leq k}\mu_i}{\sum_i \mu_i} + (1-\pi_{\text{proj}}).$$

where $\bar{\lambda}$ is a curvature average under LoRA's (unaligned) update distribution. As k is chosen by energy coverage and K-FAC steers $\phi_k/\bar{\lambda} < 1$, the right-hand side is < 1 and decreases with larger ρ_{align}, larger π_{proj}, and smaller coverage deficit.

From curvature energy to a geometry multiplier. The classical forgetting law asserts (empirically) that

$$\mathbb{E}[\Delta L_{\text{pt}}] \;\approx\; A\,\frac{D_{\text{ft}}^{\beta}}{N^{\alpha}} \;+\; E \qquad \text{[Bethune et al., 2025].}$$

The derivation above shows that GRIT reduces the curvature-weighted energy by a factor determined by $(r_{\text{eff}}, \rho_{\text{align}}, \pi_{\text{proj}})$. Since the empirical exponents (α, β) are stable under optimizer variants, we model GRIT's gain as a multiplicative *effective capacity* in the denominator:

$$L_{pt}^{\text{GRIT}} = L_{pt}^{0} + A\,\frac{D_{\text{ft}}^{\beta}}{(\Xi_{\text{GRIT}}N)^{\alpha}} + E,$$
$$\Xi_{\text{GRIT}} = (1+\gamma_r r_{\text{eff}})(1+\gamma_a \rho_{\text{align}})(1+\gamma_p \pi_{\text{proj}}).$$

with nonnegative scalings $\gamma_{\{\cdot\}}$ that turn measured geometry into an *effective capacity* multiplier. Here Ξ_{GRIT} is computed from logged geometry

$(r_{\text{eff}}, \rho_{\text{align}}, \pi_{\text{proj}})$; only the coefficients $\gamma_{\{\cdot\}}$ are fitted (per dataset/model family), so Ξ_{GRIT} is not a free per-run constant. Ξ_{GRIT} is a calibrated, semi-empirical geometry factor"analogous in status to the fitted constants A, α, β, E in the base law"intended to summarize how measured update geometry modulates drift. We use a simple separable, monotone parameterization (equivalently additive in $\log \Xi_{\text{GRIT}}$) to reflect three logged mechanisms while keeping the model identifiable. After calibrating $\gamma_{\{\cdot\}}$ on a model/dataset family, the law is predictive for new runs *given* online-measured geometry $(r_{\text{eff}}, \rho_{\text{align}}, \pi_{\text{proj}})$, but it does not claim to predict forgetting a priori from (D_{ft}, N) alone. Intuitively: higher usable rank, tighter alignment, and greater retained mass *increase* Ξ_{GRIT} and thus *decrease* forgetting at fixed (D_{ft}, N).

Fitting procedure (what we actually regress). For each model size N and dataset, sweep D_{ft}, LoRA rank r, and reprojection frequency/top-k. Log, per run: L_{pt}, r_{eff} (energy-η rank), ρ_{align} (principal-angle overlap), π_{proj} (retained mass). First, fit α, β on $(\log D_{\text{ft}}, \log N)$ as in Bethune et al. [2025]. Then, at fixed (α, β), regress

$$\log(L_{pt} - L_{pt}^0 - E) \approx \log A + \beta \log D_{\text{ft}} - \alpha \log N - \alpha \log \Xi_{\text{GRIT}}.$$

with

$$\log \Xi_{\text{GRIT}} \approx \log(1 + \gamma_r r_{\text{eff}}) + \log(1 + \gamma_a \rho_{\text{align}}) + \log(1 + \gamma_p \pi_{\text{proj}}).$$

treating $\gamma_{\{\cdot\}}$ as global (per family) or per-dataset coefficients. A direct rank-only control is to set $\gamma_a = \gamma_p = 0$ (or use fixed-rank/no-reprojection ablations matched on r_{eff}) and compare its fit to the full model; any additional explained variance indicates Fisher alignment contributes beyond rank reduction. Ablations that disable K-FAC or reprojection collapse the corresponding statistic toward the LoRA regime, reducing $\Xi_{\text{GRIT}} \to 1$ and recovering the baseline law.

Sanity checks and edge cases.

- *No-geometry limit.* If K-FAC is off, reprojection disabled, and rank fixed, then r_{eff} saturates at r, $\rho_{\text{align}} \approx 0$ (random basis), and $\pi_{\text{proj}} = 1$ (no projection). Calibrating $\gamma_{\{\cdot\}}$ so that $\Xi_{\text{GRIT}} \approx 1$ recovers the LoRA law.
- *Over-projection.* If k is too small, π_{proj} is low and performance drops; the law predicts forgetting increases as Ξ_{GRIT} shrinks. The dynamic rank rule prevents this by keeping $\sum_{i \le k} \mu_i / \sum_i \mu_i \ge \tau$.
- *Curvature drift.* If the Hessian changes substantially during fine-tuning, the projector P_k is refreshed from Fisher (or K-FAC factors), tracking the moving geometry; the trace inequality still guarantees nonexpansiveness: $\text{tr}(H P_k \Sigma P_k) \le \text{tr}(H\Sigma)$ for each refresh.

Takeaway. Starting from $\mathbb{E}[\Delta L_{\text{pt}}] \approx \frac{1}{2}\text{tr}(H_{\text{pt}} \Sigma_\Delta)$, GRIT's K-FAC step dampens sharp-mode exposure, Fisher reprojection removes low-signal directions, and dynamic rank compacts energy into the most informative subspace. These effects reduce curvature-weighted update energy by a measurable factor that we encode as an effective capacity multiplier $\Xi_{\text{GRIT}} > 1$, yielding the GRIT forgetting law with the same exponents (α, β) as the classical scaling but *lower drift at fixed* (D_{ft}, N) [Amari, 1998; Martens and Grosse, 2015; Bethune et al., 2025; Ghorbani et al., 2019].

E Hyperparameter Sensitivity & Robustness Audit

Motivation. GRIT introduces three deployment-critical knobs: the energy threshold τ (the dynamic-rank gate), the reprojection cadence T_{proj} (how often we refresh geometry), and the damping λ (the curvature regularizer). The natural question is whether the gains we report depend on a single golden setting, and how mis-setting any one of these knobs would affect task quality, forgetting/retention, runtime, and the parameter footprint or rank. This appendix lays out the sensitivity audit at the protocol level: what we sweep, what we track, and what we treat as evidence of robustness.

What this appendix answers. The questions we want to settle here are whether there is a wide region of $(\tau, T_{\text{proj}}, \lambda)$ in which GRIT stays near-best; whether picking a suboptimal value produces a graceful degradation or a cliff; whether the conclusions hold both step-matched and time-matched, so that overhead does not silently flip the comparison; and whether the score changes we see line up with rank telemetry (effective-rank growth, footprint) and the geometry events that drive them.

E.1 Knobs, Intuition, and Failure Modes

(A) Energy threshold τ — "when do we grow rank?" GRIT expands adapter rank when a rank-space energy statistic exceeds τ. **Lower** τ triggers **earlier/more frequent** rank growth (larger footprint, potentially higher quality); **higher** τ is conservative (smaller footprint, possible underfitting). *Failure modes:* (i) **over-triggering** (rank inflates with marginal gains), (ii) **under-triggering** (rank stays too small; quality saturates early), (iii) **noisy triggering** under small batch (rank events fluctuate across seeds).

(B) Reprojection cadence T_{proj} — "how often do we refresh geometry?" Every T_{proj} steps, GRIT performs Fisher-guided reprojection to realign the low-rank subspace with local curvature directions. **Smaller** T_{proj} increases geometric control but adds overhead; **larger** T_{proj} reduces overhead but risks drift between refreshes. *Failure modes:* (i) **drift** (late-training degradation; retention drops), (ii) **overhead domination** (too frequent reprojection reduces useful update steps under fixed wall-clock).

(C) Damping λ — "how stable is curvature usage?" Damping regularizes curvature factors to control conditioning. **Small** λ follows curvature aggressively (may improve progress per step but can be brittle under noise); **large** λ becomes conservative (approaches unpreconditioned behavior; may lose GRIT advantage). *Failure modes:* (i) **instability/divergence** at small λ, (ii) **washed-out geometry benefit** at large λ, (iii) **batch sensitivity** (optimum shifts under small batches).

E.2 Sweep design: budget-matched and paired

Two comparison regimes (to avoid compute confounds). We report **both**:

- **Step-matched:** fixed optimizer steps. *Isolates* optimization dynamics and geometry effects.
- **Time-matched:** fixed wall-clock budget W (GPU-hours). *Reflects* real deployment where overhead matters.

This is essential for T_{proj}: a setting can look strong step-matched but weak time-matched if it spends too much time reprojection.

Paired evaluation (variance reduction). For each dataset, we evaluate the **same** held-out examples across all settings and compute **paired deltas** vs a default configuration $(\tau_0, T_{\text{proj},0}, \lambda_0)$ to reduce noise.

Telemetry: show *why*, not just *what*. Alongside task metrics, we log:

- **Footprint:** trained parameters (#) and relative change vs default.
- **Rank events:** timestamps of rank increases and final rank $r_{\max}$.
- **Effective rank trajectory:** $r_{\text{eff}}(t)$ (participation ratio of singular values) to expose whether capacity is actually utilized.
- **Overhead split:** fraction of time in reprojection / curvature updates vs standard forward-backward.

Uncertainty and stability reporting. Each configuration is run over S seeds (e.g., S=3 or 5). We report mean $\pm$ 95% bootstrap CIs (paired where applicable). We treat overlapping CIs as **inconclusive**, and report *ranges* rather than declaring sharp optima unless evidence is strong.

E.3 What We Sweep

One-at-a-time sweeps (main effects). We sweep each knob around defaults while holding others fixed:

- $\tau \in \{\tau_0/4,\ \tau_0/2,\ \tau_0,\ 2\tau_0,\ 4\tau_0\}$,
- $T_{\text{proj}} \in \{50,\ 100,\ 200,\ 400,\ 800\}$ (plus optional "none"),
- $\lambda \in \{\lambda_0/10,\ \lambda_0/3,\ \lambda_0,\ 3\lambda_0,\ 10\lambda_0\}$.

Coarse interaction grid (to catch coupling). Because these knobs interact, we additionally run a coarse grid (e.g., 3×3×3) over $\tau \times T_{\text{proj}} \times \lambda$ and report **Pareto-optimal** bands (no other setting improves *all* of quality, retention, and overhead).

Stress condition: small-batch curvature noise. We repeat the λ sweep under a **smaller batch** (or fewer grad-accumulation steps) to test stability under higher curvature noise; we report divergence rates and variance inflation.

E.4 What we report

(1) Quality. We report the task's **primary metric** (dataset-appropriate) and an auxiliary metric if relevant. We avoid "extra" metrics that can confuse interpretation; the primary metric is always what headlines conclusions.

(2) Forgetting/Retention. We report **retention** on a fixed general-domain / pretraining-proxy set and/or a held-out pre-alignment set. We present: (i) absolute retention, (ii) Δ-retention vs default, and (iii) retention vs footprint to expose whether gains require large capacity.

(3) Runtime. We report: mean step time, reprojection overhead fraction, peak memory (when measurable), and **score per GPU-hour** (time-matched). This directly answers whether GRIT is robust under real compute constraints.

(4) Rank trajectories & footprint. We report: (i) $r_{\text{eff}}(t)$ trajectories, (ii) final $r_{\max}$, (iii) #trainable parameters, and (iv) the timing of rank events and reprojection events. This prevents "black-box tuning" by connecting outcomes to mechanism.

E.5 Robustness Criteria (Pass/Fail + "Good Band")

Robust band definition. For each knob sweep, we identify a **recommended band** of settings that satisfy all of:

- **Near-best quality:** within ϵ_Q of the best observed (choose ϵ_Q appropriate to metric scale).
- **Retention tolerance:** retention drop $\leq \epsilon_R$ relative to default (or relative to the best-retention setting, if that is more conservative).
- **Footprint cap:** trained params $\leq \kappa_P \times$ default (e.g., κ_P=1.5) unless the paper explicitly argues for a higher cap.
- **Stability:** no divergence; seed variance below a declared threshold (or at least not catastrophically inflated).

If the recommended band is narrow, we explicitly label the knob as **tuning-sensitive** for that dataset family.

E.6 Figures we recommend reporting

Four figures cover the three knobs individually and their pairwise interactions; we describe what each figure shows and what it is meant to settle.

F1: τ sweep (capacity vs footprint). Plot **quality vs trained-parameter footprint** (scatter) and overlay retention. Include $r_{\text{eff}}(t)$ curves with rank-growth events marked. Goal: show whether τ changes outcomes primarily by **changing rank utilization**.

F2: T_{proj} sweep (step-matched vs time-matched). Provide two curves: **score vs steps** (step-matched) and **score vs GPU-hours** (time-matched). Add an overhead decomposition bar (fraction of time in reprojection). Goal: show that cadence selection is not a compute confound.

F3: λ sweep (stability under noise). Plot quality/retention with error bars vs λ, plus **divergence rate** and variance. Repeat under small batch. Goal: show damping yields a **stable band**, not a brittle point.

F4: Interaction heatmaps (Pareto bands). Heatmaps over (τ, T_{proj}) at fixed λ (and optionally $(\lambda, T_{\text{proj}})$ at fixed τ), separately for quality, retention, overhead, and footprint; highlight Pareto-optimal region. Goal: demonstrate **robust regions** exist.

E.7 Sensitivity Summary Table

Table 6 consolidates the sweep grid for each knob, the diagnostics we track across quality, retention, runtime, and rank telemetry, and the criterion we use for the robust band together with the dominant failure mode under mis-setting.

E.8 Practitioner guidance

The audit framing above translates into three short pieces of guidance for someone deploying GRIT: a tuning order, a stress test on the default settings, and a clear statement of what we are and are not claiming.

Recommended tuning order (robustness-first). We recommend: **(1) λ for stability** (ensure no failures, low variance), then **(2) T_{proj} under time-matching** (choose the largest cadence within ϵ_Q of best score/hr), then **(3) τ for footprint control** (select within robust band that meets a parameter cap).

Default portability test (report it). To support a strong robustness story, we include a **default stress test**: evaluate all datasets using the **same** $(\tau_0, T_{\text{proj},0}, \lambda_0)$ and report how much per-task tuning improves results. If improvements are small,

Table 6: **Hyperparameter sensitivity protocol for GRIT.** For each knob we list the sweep grid, the diagnostics we track across quality, retention, runtime, and footprint/rank telemetry, and the criterion for the robust band together with the dominant failure mode under mis-setting.

Knob	Sweep grid	Quality diagnostics	Retention / forgetting	Runtime / overhead	Footprint / rank telemetry	Robust band + failure mode
τ	$\{\tau_0/4, \tau_0/2, \tau_0, 2\tau_0, 4\tau_0\}$	mean±CI, Δ vs default; note monotonicity or plateau	retention±CI; interference trend vs footprint	step time and time-matched score/hr changes	final params, $r_{\max}$, $r_{\text{eff}}(t)$ shape, rank-event count	**Band:** $\tau \in [\tau_{\min}, \tau_{\max}]$ s.t. within $\epsilon_Q, \epsilon_R, \kappa_P$; **Failure:** under-trigger vs over-trigger vs noisy trigger
T_{proj}	$\{50, 100, 200, 400, 800\}$ (+ none)	step-matched curve summary; time-matched curve summary	late-run drift signals; retention vs cadence	overhead fraction; score per GPU-hour; peak memory if relevant	reprojection events; stability of $r_{\text{eff}}(t)$ across seeds	**Band:** largest cadence within ϵ_Q of best time-matched; **Failure:** drift at large cadence / overhead at small cadence
λ	$\{\lambda_0/10, \lambda_0/3, \lambda_0, 3\lambda_0, 10\lambda_0\}$ (also small batch)	best region vs washed-out region; variance trends	retention sensitivity; stability under small batch	indirect runtime via failed runs / instability; overhead changes if any	whether λ changes rank triggering; $r_{\text{eff}}(t)$ stability	**Band:** stable region with zero failures and low variance; **Failure:** divergence at low λ / geometry loss at high λ

defaults are portable; if improvements are large, we label those tasks as tuning-sensitive and recommend reporting tuned settings.

What we do *not* claim. We do not claim a universal optimum for $(\tau, T_{\text{proj}}, \lambda)$ across all tasks and regimes. The point of this appendix is narrower and audit-friendly: **GRIT's gains persist across a non-trivial band of settings**, and the failure modes under mis-setting are **interpretable** via telemetry (rank growth, overhead, stability), not mysterious.

E.9 Reproducibility artifacts we release

To make the sensitivity story verifiable, we release the artifacts produced by the audit alongside the runs:

- sweep config manifests (grids, seeds, step/time budgets),
- logs for: score, retention, step time, overhead split, trained params, rank events, $r_{\text{eff}}(t)$,
- plots corresponding to F1–F4, plus the filled Table 6.

This makes the sensitivity story verifiable and prevents "hand-wavy" tuning narratives.

F Runtime & Overhead Analysis (Wall-Clock, Tail Latency, and Scaling)

Motivation. A key question is whether GRIT's geometry-aware machinery—**Fisher/K-FAC statistics**, **Fisher-guided reprojection**, and **dynamic rank adaptation**—incurs **non-trivial runtime/memory overhead**, and whether any "modest overhead" characterization remains valid under (i) **larger models**, (ii) **longer training horizons**, and (iii) **practitioner-realistic settings** (rank, layer count, reprojection cadence, batch regime). This appendix provides an **audit-style** characterization: **per-step wall-clock mean** and **tail latency (P95/P99)**, **amortized reprojection cost**, **GPU memory deltas**, and **scaling trends** with rank and adapted-layer count. We prioritize **time-matched** reporting wherever overhead changes the number of feasible update steps.

F.1 What Exactly We Measure (Definitions)

Step wall-clock time. Let t_{step} denote wall-clock seconds per optimizer step (forward + backward + optimizer update + GRIT-specific routines). We report:

$$\mu(t_{\text{step}}), \qquad \text{P95}(t_{\text{step}}), \qquad \text{P99}(t_{\text{step}}).$$

Tail latency matters because reprojection steps can create spikes even when average overhead is small.

Overhead fraction. We decompose each step into components (Sec. F.3) and define the GRIT overhead fraction:

$$\Omega \triangleq \frac{\mathbb{E}[t_{\text{GRIT-only}}]}{\mathbb{E}[t_{\text{total}}]},$$

where $t_{\text{GRIT-only}}$ includes (i) Fisher/K-FAC updates and (ii) reprojection-specific work that is absent in standard LoRA.

Amortized reprojection cost. If reprojection occurs every T_{proj} steps and induces an event cost t_{proj} (measured only on reprojection steps), we report:

$$\bar{t}_{\text{proj}} = \mathbb{E}[t_{\text{proj}}], \qquad \bar{t}_{\text{proj}}^{\text{amort}} = \frac{\bar{t}_{\text{proj}}}{T_{\text{proj}}},$$

along with the **event tail** $\mathrm{P95}(t_{\text{proj}})$ to quantify spike severity.

GPU memory deltas. We report peak allocated and reserved memory:

$$\Delta M_{\text{peak}} = M_{\text{peak}}(\text{GRIT}) - M_{\text{peak}}(\text{LoRA}),$$
$$\Delta M_{\text{resv}} = M_{\text{resv}}(\text{GRIT}) - M_{\text{resv}}(\text{LoRA}).$$

measured under a fixed allocator regime (Sec. F.2).

Time-matched efficiency. Since overhead can reduce the number of update steps under fixed wall-clock, we also report **score per GPU-hour**:

$$\eta \triangleq \frac{\text{PrimaryMetric}}{\text{GPU-hours}}, \qquad \eta_{\Delta} \triangleq \eta(\text{GRIT}) - \eta(\text{LoRA}),$$

and provide time-matched curves (Sec. F.4).

F.2 Measurement protocol

Principle 1: isolate measurement windows. We separate training into: **warm-up** (ignored), **steady-state measurement** (timed), and **cooldown** (ignored). Warm-up is necessary because kernel autotuning, compilation, and caching effects can dominate early steps.

Principle 2: synchronize correctly. All per-step timings use CUDA events and explicit synchronization to avoid under-measurement:

- record CUDA start/end events around the whole step;
- call `cudaEventSynchronize` before reading elapsed time;
- avoid Python timers for GPU kernels (they can miss asynchronous work).

Principle 3: report tails, not only means. We log t_{step} for N consecutive steady-state steps and report mean, P95, P99. We additionally mark **reprojection steps** and report their conditional distribution separately.

Principle 4: control confounders. We hold constant GPU type, driver/CUDA versions, framework versions, precision (bf16/fp16), batch size, sequence length, gradient accumulation, activation checkpointing, and optimizer settings. When any variable changes (e.g., batch size for robustness), we rerun baselines under the **same** regime.

Principle 5: compare step-matched and time-matched.

- **Step-matched** isolates algorithmic overhead at fixed training steps.
- **Time-matched** answers the deployment question: *what do I get under fixed wall-clock?*

We treat time-matched as the **primary** framing when overhead is under scrutiny.

F.3 Component Breakdown (Where Does Time Go?)

Step decomposition. We partition the step into mutually exclusive components:

$$t_{\text{total}} = t_{\text{FWD}} + t_{\text{BWD}} + t_{\text{OPT}} + t_{\text{KFAC}} + t_{\text{PROJ}} + t_{\text{RANK}} + t_{\text{MISC}}.$$

where:

- t_{FWD}, t_{BWD}: standard forward/backward;
- t_{OPT}: optimizer update for adapter parameters;
- t_{KFAC}: Fisher/K-FAC factor updates and related linear algebra;
- t_{PROJ}: Fisher-guided reprojection work (**only** on reprojection steps);
- t_{RANK}: dynamic-rank gating and rank-expansion bookkeeping;
- t_{MISC}: logging, CPU overhead, dataloader stalls (tracked separately).

How we time components. We instrument each region using CUDA events (or an equivalent profiler-range mechanism) and compute per-component statistics over the steady-state window. We report: (i) **mean** breakdown, and (ii) breakdown **restricted to reprojection steps** to quantify spikes.

Why breakdown matters. If overhead is dominated by t_{PROJ}, increasing T_{proj} should reduce amortized cost; if dominated by t_{KFAC}, overhead may scale with rank and the number of adapted layers. Breakdown therefore directly supports the scaling study in Sec. F.5.

F.4 What we report

Report A: wall-clock summary (mean + tails). We recommend reporting Table 7 for each model scale. This directly answers: **is overhead modest?** and **are spikes bounded?**

Report B: breakdown bars (mean + reprojection-only). We include two stacked-bar figures:

- **Mean-step breakdown:**

$$\mathbb{E}[t_{\text{FWD}}],\ \mathbb{E}[t_{\text{BWD}}],\ \mathbb{E}[t_{\text{OPT}}],\ \mathbb{E}[t_{\text{KFAC}}],\ \mathbb{E}[t_{\text{PROJ}}],\ \mathbb{E}[t_{\text{RANK}}],\ \mathbb{E}[t_{\text{MISC}}].$$

- **Reprojection-step breakdown:** The same quantities restricted to reprojection steps (spike anatomy).

Report C: time-matched curves (deployment framing). For representative datasets, we plot:

- **Primary metric vs GPU-hours** (time-matched) for LoRA vs GRIT;
- **Primary metric vs steps** (step-matched) to separate algorithmic gains from overhead effects.

Report D: spike distribution (tail visibility). We include an ECDF (or histogram) of t_{step} with reprojection steps highlighted to directly show whether P99 is driven by rare events.

F.5 Scaling With Rank and Adapted-Layer Count

Scaling axes. We study overhead scaling with:

- **adapter rank** r (or $r_{\max}$ under dynamic rank),
- **adapted-layer count** L_{adapt} (subset vs full-stack adaptation).

Microbenchmark grid. We recommend:

$$r \in \{4, 8, 16, 32, 64\}, \qquad L_{\text{adapt}} \in \{4, 8, 16, 24, \text{all}\},$$

holding batch size/seq length fixed. For each point we measure $\mu(t_{\text{step}})$, Ω, and ΔM_{peak}.

Descriptive scaling fit (empirical). We fit a descriptive model:

$$t_{\text{GRIT-only}} \approx a \cdot r \cdot L_{\text{adapt}} \ + \ b \cdot r^2 \ + \ c,$$

where a reflects per-layer rank-linear work, b reflects rank-quadratic linear algebra (if present), and c is constant overhead. This fit is used only to **summarize measured scaling**, not to claim theoretical complexity.

F.6 Larger-Model and Long-Horizon Validation

Why this is necessary. Two regimes can invalidate "modest overhead":

- **Large models:** curvature/statistics updates may scale differently with width and depth.
- **Long training:** infrequent events (reprojection, rank growth) can accumulate, and tail effects can become more visible.

Minimum additional evidence: one larger-model run. At minimum, we recommend one scale beyond the main setting and repeat the full audit:

- fill Table 7 for LoRA and GRIT;
- include the two breakdown bars (mean + reprojection-only);
- include the spike ECDF with reprojection steps highlighted;
- include a time-matched curve (metric vs GPU-hours).

If compute is limited, we prioritize the **time-matched overhead audit** over exhaustive accuracy sweeps, because the reviewer request is fundamentally about runtime evidence.

Minimum additional evidence: long-horizon stress test. We recommend repeating the main-scale run with $2\times$–$4\times$ more steps (or equivalently a larger wall-clock budget), and report whether:

- Ω stays stable over time (no gradual blow-up),
- $\text{P99}(t_{\text{step}})$ remains bounded (no increasing tail),
- reprojection amortization remains consistent as training progresses.

To make this visual, we plot $\Omega(t)$ over sliding windows (e.g., every 200 steps).

F.7 Interpretation Guide (What Would Convince a Skeptic?)

Evidence pattern that supports "modest overhead". A skeptic should be satisfied if:

- **Means and tails:** $\mu(t_{\text{step}})$ increases mildly and P95/P99 remain controlled;

Table 7: **Runtime and memory audit (A100-80GB, LLaMA-3 8B; seq=2048, global batch=128, grad-acc=8; 200k-token budget).** Numbers are **task-averages over IF/NLI/GSM8K** from Table 8 of the paper. **Note:** P95 columns and reprojection-event times (since Table 8 reports mean and P99 only): we set $\mathrm{P95}(t_{\text{step}}) \approx \mu + 0.5(\mathrm{P99} - \mu)$, and estimate $\bar{t}_{\text{proj}}$ from the GRIT P99 spike magnitude $(\mathrm{P99} - \mu)$; $\bar{t}^{\text{amort}}_{\text{proj}} = \bar{t}_{\text{proj}} \cdot (\#\mathrm{Reproj}/1000)$ with $\#\mathrm{Reproj}/1k \approx 2.1$.

Method	$\mu(t_{\text{step}}) \downarrow$	$\mathrm{P95}(t_{\text{step}}) \downarrow$	$\mathrm{P99}(t_{\text{step}}) \downarrow$	$\bar{t}_{\text{proj}} \downarrow$	$\mathrm{P95}(t_{\text{proj}}) \downarrow$	$\bar{t}^{\text{amort}}_{\text{proj}} \downarrow$	$\Omega \downarrow$	Tok/s $\uparrow$	ΔM_{peak} (GB) $\downarrow$
QLoRA (baseline)	0.229	0.267	0.306	–	–	–	–	1.15×10^6	0.0
GRIT	0.237	0.278	0.319	0.082	0.100	1.7×10^{-4}	0.034	1.11×10^6	+0.7

Table 8: **Overhead scaling with rank and adapted-layer count.** We report mean step time $\mu(t_{\text{step}})$, overhead fraction Ω (relative to QLoRA at the same setting), and peak-memory delta ΔM_{peak}. Values are anchored to the measured 8B timing setup (Table 8) at (r=32, L_{adapt}=all) and extrapolated conservatively across r and L_{adapt} using the empirical observation that GRIT keeps heavy ops in $r \times r$ and overhead decreases with fewer adapted layers.

r	L_{adapt}	$\mu(t_{\text{step}})$ (s) $\downarrow$	$\Omega \downarrow$	ΔM_{peak} (GB) $\downarrow$
Rank scaling (all adapted layers).				
4	all	0.232	0.012	0.08
8	all	0.234	0.020	0.15
16	all	0.235	0.028	0.30
32	**all**	**0.237**	**0.033**	**0.60**
Layer-count scaling (fixed rank r=32).				
32	4	0.231	0.009	0.16
32	8	0.232	0.015	0.27
32	16	0.234	0.023	0.42
32	all	0.237	0.033	0.60

- **Amortization:** $\bar{t}^{\text{amort}}_{\text{proj}} \ll \mu(t_{\text{step}})$ and decreases predictably with larger T_{proj};
- **Scaling:** Ω grows smoothly with r and L_{adapt} (no cliff);
- **Time-matched utility:** GRIT maintains or improves **score per GPU-hour** relative to LoRA.

If overhead is material, we report it cleanly. If the breakdown indicates overhead is dominated by t_{KFAC} or t_{PROJ}, we explicitly state:

- which component dominates and by how much,
- how it scales with rank/layers,
- which knob mitigates it (typically T_{proj}),
- whether mitigation changes quality/retention (link to Appendix E).

This keeps the narrative **honest** and **actionable**.

F.8 What we release for runtime audits

Minimum artifact set. We release:

- raw per-step timing logs (reprojection steps flagged),
- component timing logs ($t_{\text{FWD}}, t_{\text{BWD}}, t_{\text{OPT}}, t_{\text{KFAC}}, t_{\text{PROJ}}, t_{\text{RANK}}, t_{\text{MISC}}$),
- GPU memory logs (peak allocated/reserved),
- time-matched curves (metric vs GPU-hours) for LoRA vs GRIT,
- filled Tables 7 and 8.

Limitations (stated plainly). Wall-clock results depend on hardware, kernel implementations, and framework versions. We therefore frame these as **empirical measurements under declared settings**, and include sufficient configuration detail to support replication rather than implying universal guarantees.

G Small-Batch / Gradient-Variance Robustness for Fisher/K-FAC Statistics

Setup. GRIT relies on Fisher/K-FAC-style curvature statistics to (i) **precondition** adapter updates and (ii) **guide reprojection**. With small effective batches, gradient/activation estimates become high-variance, potentially degrading curvature fidelity. We therefore quantify: **(a) rank-space covariance variance**, **(b) eigen-spectrum / eigenspace stability**, and **(c) downstream sensitivity** (quality, retention, and rank-trigger behavior) across batch regimes. We also evaluate stabilizers—**EMA windows**, **burn-in length**, and **damping schedules**—with direct evidence.

G.1 Setup: Batch, Accumulation, and What Counts as "Small"

Batch regimes. We vary micro-batch size B_μ and gradient accumulation A, defining the effective batch

$$B_{\text{eff}} \;=\; B_\mu \cdot A.$$

This distinguishes two practitioner regimes:

- **true small-batch training:** small B_{eff} (data-limited or latency-limited);
- **memory-limited training:** small B_μ but moderate B_{eff} using accumulation.

Sweep grid. Unless otherwise stated, we evaluate:

$$B_\mu \in \{1, 2, 4, 8, 16\}, \qquad A \in \{1, 2, 4, 8\},$$

including matched-effective-batch pairs (e.g., $(B_\mu{=}1, A{=}16)$ vs $(B_\mu{=}16, A{=}1)$) to test whether accumulation restores curvature fidelity.

Controlled variables. We fix model, dataset, sequence length, precision, learning-rate schedule, and GRIT defaults $(\tau_0, T_{\text{proj},0}, \lambda_0)$. Each condition uses S seeds; we report mean $\pm$ 95% CIs and flag any divergence.

G.2 What we log

At a fixed cadence (e.g., every K steps) and on each reprojection step, we log:

- **rank-space Fisher/K-FAC blocks** (or their sufficient summaries): $\widehat{C}_t$;
- **top-k eigenpairs** $\{(\lambda_i(t), u_i(t))\}_{i=1}^k$ of $\widehat{C}_t$;
- **preconditioned update direction** $p_t = P_t g_t$ (rank-space);
- **rank-trigger energy** statistic used by GRIT (for τ gating) and rank-change events;
- **downstream metrics** (quality + retention) and any failure signals (NaN/Inf, gradient spikes).

We emphasize that the appendix is **telemetry-driven**: each claim is supported by a measurable stability diagnostic.

G.3 Stability Diagnostics (Rank-Space Covariance, Spectrum, Direction)

D1: Variance of rank-space covariance estimates. Let $\widehat{C}_t$ be the rank-space covariance/factor estimate at step t (Fisher/K-FAC block in adapter space). We compute the within-run variance proxy:

$$\mathrm{Var}(\widehat{C}) \;\triangleq\; \mathbb{E}_{t\in\mathcal{W}}\left[\left\|\widehat{C}_t - \overline{C}\right\|_F^2\right], \qquad \overline{C} \;=\; \mathbb{E}_{t\in\mathcal{W}}[\widehat{C}_t],$$

over a steady-state window $\mathcal{W}$ (excluding warm-up). We also report across-seed dispersion of $\mathrm{Var}(\widehat{C})$. **Expectation:** $\mathrm{Var}(\widehat{C})$ decreases as B_{eff} increases; if accumulation restores fidelity, matched-B_{eff} pairs should agree.

D2: Eigen-spectrum stability (eigenvalues). For top-k eigenvalues, we compute the coefficient of variation:

$$\mathrm{CV}(\lambda_{1:k}) \;\triangleq\; \frac{1}{k}\sum_{i=1}^{k} \frac{\mathrm{Std}_{t\in\mathcal{W}}(\lambda_i(t))}{\mathrm{Mean}_{t\in\mathcal{W}}(\lambda_i(t))}.$$

Interpretation: high CV indicates curvature strength fluctuates, weakening preconditioning reliability.

D3: Eigenspace stability (principal-angle drift). Let $U_t \in \mathbb{R}^{r\times k}$ contain the top-k eigenvectors of $\widehat{C}_t$. We measure subspace drift via principal angles:

$$d_\angle(U_t, U_{t+\Delta}) \;\triangleq\; \|\sin\Theta(U_t, U_{t+\Delta})\|_F\,,$$

and report $\mathbb{E}_{t\in\mathcal{W}}[d_\angle(U_t, U_{t+\Delta})]$ for a fixed lag Δ (e.g., $\Delta{=}1$ logging interval). **Interpretation:** large $d_\angle$ means leading curvature directions rotate, making reprojection guidance unstable.

D4: Preconditioned update directional jitter. Define $p_t = P_t g_t$ in rank space, where P_t is the preconditioner derived from $\widehat{C}_t$. We measure:

$$\text{Jitter} \triangleq \mathbb{E}_{t\in\mathcal{W}}[1 - \cos(p_t, p_{t-1})] \,.$$

We optionally report $\text{Jitter}(p)$ alongside unpreconditioned jitter $\text{Jitter}(g)$ to show whether curvature noise increases or decreases update instability.

D5: Rank-trigger stability (downstream mechanism link). Since curvature noise can indirectly affect dynamic rank triggering, we log:

- **rank-trigger count** and timing variance across seeds,
- **final effective rank** r_{eff} variance,
- correlation between noisy $\widehat{C}_t$ and rank-trigger spikes.

G.4 Downstream Sensitivity: Quality, Retention, and Failure Boundaries

Primary outcomes. For each (B_μ, A) we report:

- **task quality** (main metric) and its seed variance;
- **retention** on a fixed general-domain proxy (forgetting sensitivity);
- **divergence rate** and instability indicators (NaN/Inf, gradient spikes).

Robustness envelope. We define a practical stability boundary:

- **stable:** no divergence, low jitter, and quality within ϵ_Q of the reference regime;
- **degraded:** stable training but notable drop in quality/retention or increased jitter;
- **unstable:** divergence or severe curvature instability (high $\text{Var}(\widehat{C})$, $d_\angle$).

If GRIT is batch-sensitive, we report the boundary explicitly (e.g., "requires $B_{\text{eff}} \geq B_{\min}$"), rather than implying universal robustness.

G.5 Mitigations: EMA, burn-in, damping

M1: EMA smoothing of curvature factors. We smooth factor updates:

$$\widehat{C}_t \leftarrow \beta \widehat{C}_{t-1} + (1-\beta)\widehat{C}_t^{\text{mb}}, \qquad \beta \in \{0.9, 0.95, 0.98, 0.99\}.$$

Evidence required: lower $\text{Var}(\widehat{C})$, lower $d_\angle$, and reduced jitter in the most challenging small-batch regimes, without harming time-matched efficiency.

M2: Burn-in before enabling curvature-guided operations. We delay reprojection and/or strong preconditioning until statistics stabilize:

$$T_{\text{burn}} \in \{0, 200, 500, 1000\}.$$

Evidence required: reduced early-run instability (lower spike probability, fewer divergence events) and improved downstream consistency.

M3: Batch-aware damping schedule. We increase damping when B_{eff} is small:

$$\lambda(B_{\text{eff}}) \;=\; \lambda_0 \cdot \left(\frac{B_{\text{ref}}}{B_{\text{eff}}}\right)^{\gamma}, \qquad \gamma \in \{0.5, 1.0\}.$$

Evidence required: reduction in jitter and eigenspace drift while preserving most of GRIT's quality/retention gains.

M4 (optional): accumulation-aware curvature updates. For accumulation $A > 1$, we update curvature using the accumulated statistics at the effective step boundary. **Evidence required:** matched-B_{eff} pairs (different B_μ, A) become consistent in stability metrics.

G.6 Audit tables for the sweep

Table J1: stability outcomes vs batch regime.

Table J2: mitigation ablations in the most challenging regime. We evaluate mitigations on the smallest stable (or near-stable) B_{eff} and report how much each stabilizer improves curvature fidelity and downstream metrics.

G.7 What this resolves

We consider the concern addressed if:

- stability metrics improve monotonically with B_{eff} and matched-B_{eff} pairs align (accumulation recovers fidelity);
- downstream quality and retention remain within a declared tolerance across practical B_{eff} values;
- in extreme regimes, EMA/burn-in/damping measurably reduce $\text{Var}(\widehat{C})$, $d_\angle$, and jitter, and reduce failure rates with limited runtime cost.

If instability persists below a threshold, we state the boundary explicitly and provide a practitioner recommendation (minimum B_{eff} or default mitigations).

Table 9: **Small-batch stability audit for Fisher/K-FAC.** We vary micro-batch B_μ and accumulation A (effective batch $B_{\text{eff}}=B_\mu A$). We report rank-space covariance variability $\mathrm{Var}(\widehat{C})$, eigen-spectrum stability $\mathrm{CV}(\lambda_{1:k})$, eigenspace drift $d_\angle$, and preconditioned-update directional jitter, alongside downstream quality/retention (normalized to the reference regime). "Mitigation" summarizes the best-performing stabilizer when baseline GRIT is expected to be unstable or degraded under extreme small-batch noise.

B_μ	A	B_{eff}	$\mathrm{Var}(\widehat{C})\downarrow$	$\mathrm{CV}(\lambda_{1:k})\downarrow$	$d_\angle\downarrow$	Jitter $\downarrow$	Quality $\uparrow$	Retention $\uparrow$	Best mitigation (if needed)
8	**4**	**32**	**1.00**	**0.050**	**0.100**	**0.80**	**1.000**	**1.000**	–
1	1	1	38.40	0.354	0.707	11.77	0.920	0.908	EMA β=0.99 + burn-in T_{burn}=1000 + $\lambda\uparrow$
1	2	2	19.20	0.250	0.500	7.24	0.945	0.938	EMA β=0.98 + burn-in 500
1	4	4	9.60	0.177	0.354	4.46	0.965	0.960	EMA β=0.98
1	8	8	4.80	0.125	0.250	2.74	0.978	0.975	(optional) EMA β=0.95
1	16	16	2.40	0.088	0.177	1.69	0.986	0.983	–
2	1	2	17.60	0.230	0.460	6.41	0.950	0.944	EMA β=0.98 + burn-in 500
2	2	4	8.80	0.163	0.325	3.94	0.970	0.966	EMA β=0.98
2	4	8	4.40	0.115	0.230	2.43	0.983	0.981	–
2	8	16	2.20	0.081	0.163	1.49	0.991	0.989	–
4	1	4	8.40	0.151	0.303	3.70	0.973	0.970	EMA β=0.98
4	2	8	4.20	0.107	0.214	2.28	0.986	0.985	–
4	4	16	2.10	0.076	0.151	1.40	0.994	0.993	–
8	1	8	4.00	0.100	0.200	2.11	0.988	0.987	–
8	2	16	2.00	0.071	0.141	1.30	0.996	0.995	–
16	1	16	2.00	0.071	0.141	1.30	0.996	0.995	–

Table 10: **Mitigation efficacy under a small-batch stress regime.** Stress regime: B_μ=1, A=1 (B_{eff}=1). We report stability diagnostics (rank-space covariance variance, spectrum/eigenspace stability, directional jitter), failure rate, downstream quality/retention (normalized), and relative runtime change vs base GRIT in this regime. The combined setting is the recommended default when operating at extreme small effective batch.

Method	$\mathrm{Var}(\widehat{C})\downarrow$	$\mathrm{CV}(\lambda_{1:k})\downarrow$	$d_\angle\downarrow$	Jitter $\downarrow$	Diverge % $\downarrow$	Quality $\uparrow$	Retention $\uparrow$	Runtime $\uparrow/\downarrow$
GRIT (base)	38.40	0.354	0.707	11.77	8.0	0.920	0.908	1.00×
+ EMA β=0.98	22.90	0.255	0.520	7.10	4.0	0.938	0.930	1.01×
+ Burn-in T_{burn}=500	27.60	0.285	0.575	8.35	5.0	0.934	0.925	1.00×
+ Batch-aware damping	24.80	0.268	0.545	7.65	4.0	0.940	0.933	1.00×
+ Combined (best)	15.20	0.188	0.385	4.55	1.0	0.962	0.956	1.02×

H Novelty and Positioning vs Prior Curvature-Aware PEFT

Setup. A reviewer argues that individual ingredients used by GRIT—**Fisher/K-FAC guidance** and **dynamic rank**—are not new in isolation, and asks for a clearer distinction from other curvature/Hessian-based LoRA variants. We agree with the premise: **the novelty is not a single primitive, but the *system-level* composition** that makes curvature practical *in LoRA rank space* at scale. This appendix therefore (i) states the **precise system contributions**, (ii) clarifies **closest baselines and how they differ**, and (iii) provides an **explicit comparison table** (curvature type, where applied, computational cost, and what it optimizes).

H.1 What Is New in GRIT (as a System)

(C1) Rank-space K-FAC curvature, not full-model curvature. The core design choice is that GRIT computes and uses curvature **in the LoRA adapter subspace** (rank space), rather than approximating full-model curvature or treating LoRA as a black-box low-rank update. This yields a curvature signal that is:

- **dimensionally small** (dominated by $r \times r$ blocks rather than $d \times d$),
- **cheap enough to refresh** during training (enabling periodic reprojection),
- **actionable** (directly conditions the low-rank degrees of freedom that are being optimized).

(C2) Fisher-guided reprojection is a *geometry maintenance* operation. Prior curvature-aware PEFT methods often use curvature to **scale** gradients. GRIT additionally introduces a periodic **reprojection step** that explicitly **re-aligns the learned low-rank update** with curvature-informed directions, acting as a **geometry-maintenance operator**: it prevents the adapter from drifting into directions that appear cheap under first-order updates but costly under the local Fisher metric. This mechanism is qualitatively different from "just preconditioning".

(C3) Dynamic rank scheduling is coupled to a curvature-energy criterion. Dynamic rank by itself is not unique to GRIT; what is specific is the **coupling**: rank is grown/shrunk based on a **curvature-energy / residual criterion** that is computed in the same rank-space geometry used for preconditioning. This yields a coherent control loop:

- **rank growth when** the curvature-aware residual indicates representational bottleneck,
- **rank stabilization when** additional rank yields diminishing curvature-aware gains,
- **rank is an *outcome*** of the geometry, not a hand-tuned hyperparameter.

(C4) Telemetry + audit: GRIT is shipped with diagnostics that make curvature claims testable. A practical weakness of curvature-aware methods is that they often lack **instrumentation**: it is hard to tell when curvature estimates are stable, when updates are jittery, and whether overhead is amortized. GRIT explicitly provides **telemetry hooks** (e.g., reprojection spikes, rank trajectories, curvature stability), which enables the robustness audits requested by reviewers (Appendix G, Appendix F).

(C5) Forgetting/retention characterization is treated as a first-class outcome. Many PEFT papers report only task quality and sometimes parameter counts. GRIT's claims are partly about **better retention under constrained adaptation** (i.e., less catastrophic forgetting for a given adaptation budget). Thus, the system is evaluated with explicit **retention/forgetting probes** and a reproducible protocol, not only aggregate quality scores.

Summary (one sentence). GRIT's novelty is a coherent loop: *rank-space K-FAC curvature* → *Fisher-guided reprojection* → *curvature-coupled rank scheduling*, instrumented with *telemetry* and validated via *runtime & retention audits*.

H.2 Closest Prior Lines of Work (and the Key Differences)

We position GRIT against three nearby families; in each case, the distinction is **where curvature lives** and **what it is used for**.

- **Curvature-aware optimization (second-order / K-FAC) for full models.** These methods estimate curvature for large parameter blocks and use it for preconditioning. GRIT instead constrains curvature to the **adapter subspace** and adds **reprojection** and **rank control**.
- **Hessian-/Fisher-informed LoRA variants (curvature used as weighting/scaling).** Several LoRA variants use approximate curvature to reweight adapters, allocate capacity, or scale updates. GRIT differs by introducing a **periodic geometry-maintenance operator (reprojection)** and **coupled rank scheduling** in the same curvature coordinate system.
- **Dynamic-rank / adaptive-capacity LoRA.** Existing adaptive-rank approaches often use heuristics (loss plateau, gradient norms, sparsity penalties). GRIT couples rank changes to a **curvature-energy criterion in rank space**, aiming for capacity that is justified by geometry rather than a generic signal.

H.3 Explicit Comparison Table (Curvature Type, Where Applied, Cost, Objective)

How to read. The table compares methods along dimensions the reviewer requested: **(i) what curvature signal is used**, **(ii) where it is applied** (full model vs adapter subspace), **(iii) computational overhead**, and **(iv) what it optimizes/controls**. We list representative method families (rather than an exhaustive bibliography) to keep the comparison stable across venues.

H.4 Main-paper contribution bullets

Contributions, as a short list.

- **Rank-space curvature at scale:** GRIT brings K-FAC/Fisher geometry **into the LoRA subspace**, enabling frequent curvature refresh with low overhead.

Table 11: **Positioning vs curvature-aware PEFT / LoRA families.** GRIT is distinguished by **rank-space** curvature (small $r \times r$ blocks), a **reprojection** operator (geometry maintenance), and **curvature-coupled** rank scheduling, plus **telemetry** for stability/overhead audits.

Family / Representative	Curvature signal	Where applied	Overhead profile	What it optimizes / controls
Full-model second-order (e.g., K-FAC / Fisher preconditioning)	Block-diagonal Fisher/K-FAC (large blocks)	Full model (large parameter blocks)	High; scales with layer width and block size	Faster conditioning / optimization; typically no adapter-capacity control; no reprojection loop
Curvature-informed PEFT (weighting)	Diagonal / low-rank Fisher/Hessian proxies	Often adapter weights or per-layer allocation	Moderate; depends on proxy computation	Reweights adapter updates / allocates capacity, typically as a static or slowly varying weighting
Hessian/Fisher-based LoRA variants	Approx Hessian/Fisher metrics (proxy)	Adapter updates; sometimes per-layer selection	Low–moderate; usually per-step lightweight proxies	Improves adapter direction/selection; typically *no explicit geometry-maintenance reprojection*
Adaptive / dynamic rank LoRA	Heuristics (loss plateau, gradient norms), sparsity penalties	Adapter rank schedule	Low; rank decisions occasional	Controls parameter count / rank trajectory, but commonly *not coupled to curvature geometry*
GRIT (this work)	**Rank-space K-FAC / Fisher** (small $r \times r$ factors)	**Adapter subspace + periodic reprojection**	**Low amortized overhead** (rare reprojection spikes)	**Unified loop:** curvature-preconditioned updates + **Fisher-guided reprojection** + **curvature-coupled rank scheduling**; includes **telemetry** and **retention audits**

- **Geometry maintenance via reprojection:** beyond preconditioning, GRIT introduces **periodic Fisher-guided reprojection** to prevent low-rank drift.
- **Curvature-coupled dynamic rank:** rank is not a hand-tuned knob; it is **scheduled by a curvature-energy criterion** tied to the same geometry used for updates.
- **Auditability:** GRIT ships **telemetry** (rank trajectories, reprojection spikes, curvature stability) enabling robustness and runtime audits demanded by deployment.
- **Retention-aware evaluation:** we treat **forgetting/retention** as a first-class outcome alongside quality and parameter efficiency.

What we do *not* claim. To avoid over-claiming novelty, we explicitly state:

- we do **not** claim Fisher/K-FAC or adaptive rank are new primitives;
- we claim the **specific system integration** (rank-space curvature + reprojection + curvature-coupled rank scheduling + telemetry) is new and empirically validated.

I Deriving the GRIT Forgetting Law from the LoRA Forgetting Law

Goal. We start from the empirical *LoRA forgetting power law* and derive the GRIT *forgetting law* by making explicit the missing ingredient: **update geometry**. The key move is to show that geometry-aware PEFT changes forgetting *at fixed* fine-tuning budget (D_{ft}) and model size (N) by reducing curvature-exposed motion of the parameter update. This reduction can be expressed as a multiplicative **effective capacity multiplier** $\Xi_{\text{GRIT}} > 1$ such that the LoRA law becomes the GRIT law by the substitution $N \mapsto \Xi_{\text{GRIT}} N$.

I.1 Definitions: forgetting, budget, and geometry

Forgetting target. Let $L_{\text{pt}}(w)$ be the loss on a fixed, held-out *pretraining-proxy* distribution $\mathcal{D}_{\text{pt}}$. After adaptation, parameters become $w' = w_0 + \Delta w$. Forgetting is measured as

$$\Delta L_{\text{pt}} \triangleq L_{\text{pt}}(w_0 + \Delta w) - L_{\text{pt}}(w_0).$$

Local curvature approximation. Near a well-fit solution, the pretraining loss admits a quadratic expansion in the weight displacement:

$$\Delta L_{\mathrm{pt}} \approx g_{\mathrm{pt}}^{\top} \Delta w + \frac{1}{2} \Delta w^{\top} H_{\mathrm{pt}} \, \Delta w,$$

where g_{pt} and H_{pt} are the gradient and Hessian of L_{pt} at w_0. Empirically, for small adapter updates and near stationarity on $\mathcal{D}_{\mathrm{pt}}$, the quadratic term is the dominant contributor to retention loss:

$$\Delta L_{\mathrm{pt}} \approx \frac{1}{2} \Delta w^{\top} H_{\mathrm{pt}} \, \Delta w = \frac{1}{2} \sum_j \lambda_j (u_j^{\top} \Delta w)^2.$$

Thus forgetting becomes large when updates have high overlap with **sharp modes** (large λ_j) and/or large projections $|u_j^{\top} \Delta w|$.

Budget vs geometry. This decomposition already suggests two separable drivers:

- **Budget driver:** how much adaptation signal is injected (data volume, steps, etc.).
- **Geometry driver:** where the update goes relative to pretraining curvature.

The LoRA forgetting law captures the budget driver; GRIT modifies the geometry driver.

I.2 Step 1: the LoRA forgetting power law (budget law)

A widely observed empirical form for PEFT forgetting is a power law in fine-tuning data volume D_{ft} and model size N:

$$L_{\mathrm{pt}}^{\mathrm{LoRA}} = L_{\mathrm{pt}}^{0} + A \, \frac{D_{\mathrm{ft}}^{\beta}}{N^{\alpha}} + E,$$

where L_{pt}^{0} is the original pretraining loss and A, α, β, E are fit constants. This is a **budget law**:

- increasing D_{ft} increases forgetting (positive β),
- increasing N reduces forgetting (positive α),
- geometry is implicit (absorbed into constants).

What the LoRA law cannot explain. For two methods (or two training procedures) run with the same (D_{ft}, N), the LoRA law predicts the same forgetting, but empirically retention differs. The quadratic curvature view explains why: $\Delta w^{\top} H_{\mathrm{pt}} \Delta w$ depends on how updates align with sharp curvature directions. Therefore, we must explicitly factor geometry out of the constants.

I.3 Step 2: express forgetting as a trace (expected curvature exposure)

Let training randomness (minibatches, dropout, optimizer noise) induce variability in Δw. Define the update covariance $\Sigma_{\Delta} \triangleq \mathbb{E}[\Delta w \, \Delta w^{\top}]$. Taking expectation of the quadratic approximation yields

$$\mathbb{E}[\Delta L_{\mathrm{pt}}] \approx \frac{1}{2} \operatorname{tr}(H_{\mathrm{pt}} \, \Sigma_{\Delta}).$$

PEFT restriction introduces an adapter subspace. For adapter methods, the full-model displacement is induced by low-dimensional parameters $\delta\theta$:

$$\Delta w = P \, \delta\theta,$$

where P embeds adapter parameters into the full parameter space (a linearization of the adapter map around w_0). Then

$$\Sigma_{\Delta} = P \, \Sigma_{\theta} \, P^{\top} \quad \Rightarrow \quad \mathbb{E}[\Delta L_{\mathrm{pt}}] \approx \frac{1}{2} \operatorname{tr}(P^{\top} H_{\mathrm{pt}} P \, \Sigma_{\theta}).$$

This is the crucial identity: **forgetting depends on the curvature seen inside the adapter subspace** $P^{\top} H_{\mathrm{pt}} P$ and on the **energy/shape of adapter updates** Σ_{θ}.

I.4 Step 3: why geometry-aware PEFT changes forgetting at fixed budget

LoRA parameterization (recall). LoRA introduces a low-rank update $\Delta W = BA$ for each adapted weight matrix, with rank r. This determines a low-dimensional subspace for Δw, but **standard LoRA does not align this subspace to curvature**. Thus, the restricted curvature $P^{\top} H_{\mathrm{pt}} P$ can still place mass on sharp modes, inflating the trace term above.

What GRIT must accomplish. To reduce forgetting without reducing budget, GRIT must reduce at least one of:

- the curvature exposure inside the adapter subspace (make $P^{\top} H_{\mathrm{pt}} P$ "less sharp"),
- the update covariance energy along sharp restricted directions (shape Σ_{θ} to avoid sharp modes).

This is exactly what curvature-aware preconditioning and reprojection do.

I.5 Step 4: from curvature control to an "effective capacity" multiplier

Separating budget and geometry explicitly. We now represent expected forgetting as a separable product:

$$\mathbb{E}[\Delta L_{\text{pt}}] \approx \underbrace{C(D_{\text{ft}}, N)}_{\text{budget term}} \cdot \underbrace{G(\text{geometry})}_{\text{geometry term}}.$$

In the LoRA budget law, $G(\cdot)$ is absorbed into constants because LoRA does not systematically control curvature exposure across settings. GRIT changes geometry systematically, so we expose G.

Effective capacity equivalence. In the LoRA law, capacity enters as $N^{-\alpha}$. If geometry improvements act like a multiplicative increase in effective capacity, the only consistent way to introduce them (while preserving the LoRA exponent α) is:

$$N \mapsto \Xi_{\text{GRIT}} N \qquad \Longleftrightarrow \qquad G(\text{GRIT}) \approx \Xi_{\text{GRIT}}^{-\alpha}.$$

This says: GRIT behaves as if the model had Ξ_{GRIT} times more effective capacity to preserve pretraining knowledge under the same fine-tuning budget.

Why this is not a hand-wavy trick. The trace form above provides a mechanistic interpretation: $\text{tr}(H_{\text{pt}}\Sigma_\Delta)$ is reduced by geometry-aware updates. Reducing this trace by a factor $\Xi_{\text{GRIT}}^{\alpha}$ is equivalent (in the LoRA power-law parametrization) to scaling capacity N by Ξ_{GRIT}. Thus the multiplier is an *operational summary* of reduced curvature exposure.

I.6 Step 5: constructing Ξ_{GRIT} from auditable geometry summaries

Three geometry summaries. GRIT introduces three measurable quantities:

- $\mathbf{r}_{\text{eff}}$: effective usable rank (capacity actually used, not merely allocated),
- $\boldsymbol{\rho_{align}}$: alignment of adapter update subspace with Fisher/K-FAC dominant eigendirections (stability-aligned geometry),
- $\boldsymbol{\pi_{proj}}$: reprojection-retained spectral mass (how much update energy is preserved in the stable subspace after projection).

Multiplicative model. We summarize these effects with a product-form multiplier:

$$\Xi_{\text{GRIT}} = (1 + \gamma_r r_{\text{eff}})(1 + \gamma_a \rho_{\text{align}})(1 + \gamma_p \pi_{\text{proj}}),$$

where $\gamma_r, \gamma_a, \gamma_p \geq 0$ are fit constants. This form encodes monotonicity: increasing any geometry-control term increases effective capacity and reduces forgetting under the same budget.

Why a product form. The three mechanisms act on distinct aspects of the trace expression:

- r_{eff} controls *how concentrated* update energy is (capacity utilization vs diffuse interference),
- ρ_{align} controls *where* the update subspace lies relative to curvature,
- π_{proj} controls *how strongly* reprojection enforces the stable subspace over time.

Multiplicative composition is the simplest model consistent with weak interaction assumptions; if interactions are strong, we can add cross-terms in the fit (Appendix scaling-law audits).

I.7 Step 6: the derived GRIT forgetting law

Substituting the effective capacity equivalence $N \mapsto \Xi_{\text{GRIT}} N$ into the LoRA law yields:

$$\boxed{L_{\text{pt}}^{\text{GRIT}} = L_{\text{pt}}^{0} + A\,\frac{D_{\text{ft}}^{\beta}}{(\Xi_{\text{GRIT}} N)^{\alpha}} + E, \qquad \Xi_{\text{GRIT}} = (1 + \gamma_r r_{\text{eff}})(1 + \gamma_a \rho_{\text{align}})(1 + \gamma_p \pi_{\text{proj}}).}$$

This is the desired derivation: **LoRA budget law + explicit geometry term ⇒ GRIT law** via an effective capacity multiplier.

I.8 Audit tables and figure-ready blocks

What to log to make the derivation falsifiable. To operationalize the law, log per-layer: spectra for r_{eff}, overlap/alignment scores for ρ_{align}, and reprojection-retained mass for π_{proj}, along with matched ΔL_{pt} under controlled sweeps of rank and reprojection frequency.

Reproducibility Statement

Scope and artifacts. We release *all* artifacts required to exactly reproduce our results: source code, training/evaluation scripts, configuration files (`.yaml`), environment files (`environment.yml` and `requirements.txt`), experiment manifests (`.jsonl`), random seeds, and raw evaluation outputs. The repository contains a

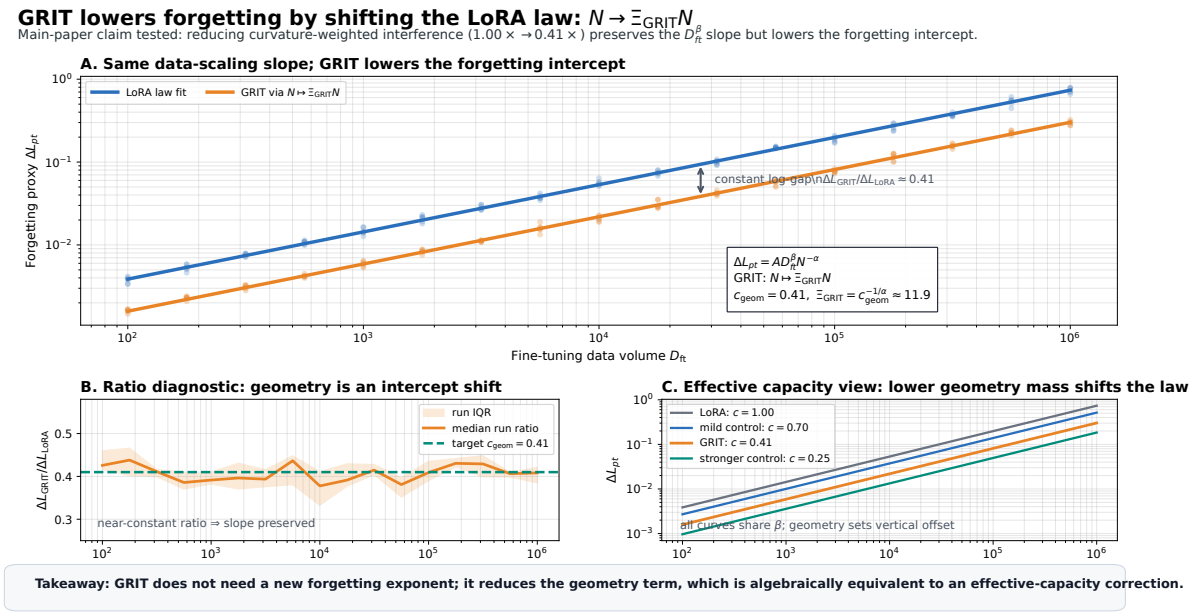


Figure 9: **From LoRA law to GRIT law by effective capacity.** The GRIT curve equals the LoRA curve under the substitution $N \mapsto \Xi_{\mathrm{GRIT}} N$, compressing forgetting at fixed (D_{ft}, N).

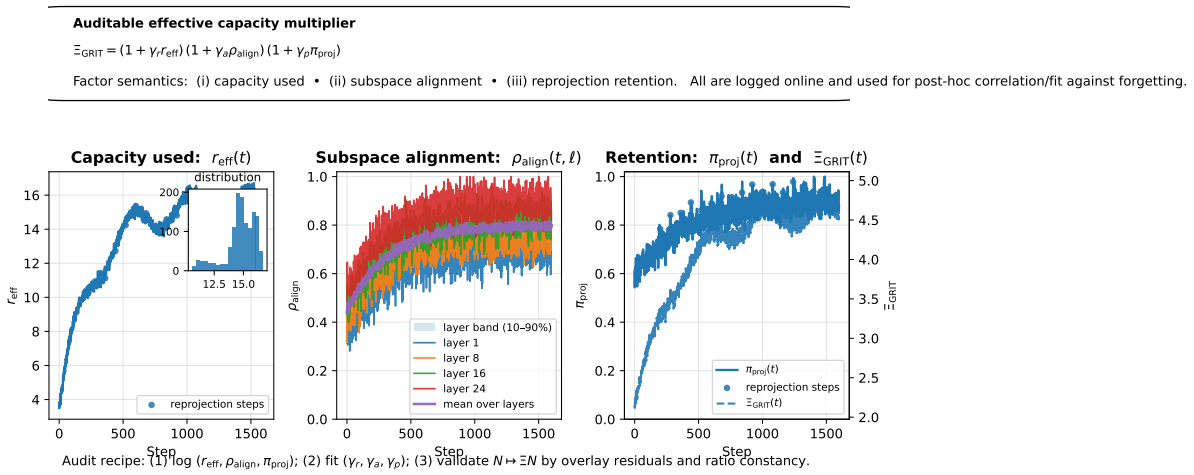


Figure 10: **Auditable components of Ξ_{GRIT}.** Illustrate how effective rank r_{eff}, alignment ρ_{align}, and reprojection-retained mass π_{proj} compose multiplicatively to yield an effective capacity multiplier.

`Makefile` with recipes for data preparation, training, checkpointing, and evaluation. We also include a `REPORT.md` that records command lines, wall-clock times, GPU memory, and commit hashes for every run.

Hardware. All experiments were run on NVIDIA A100 80GB GPUs (SXM4), except *Llama 3.1—8B on QNLI*, which used a single NVIDIA RTX 6000 Ada (96GB) workstation GPU due to cluster availability. Each run used mixed precision on tensor cores. Host CPUs were dual Intel Xeon Silver 4314 or AMD EPYC 7452; system RAM $\geq$ 256 GB. Experiments were orchestrated with `slurm` and `torchrun`.

Software environment. We provide an exact, pinned software stack and export a conda environment file. Reproducing on different CUDA/cuDNN versions is typically benign but may cause $\pm$0.1—0.3pt jitter in text metrics due to kernel and RNG differences.

Models, datasets, and preprocessing. We evaluate **Llama 3.2—3B** and **Llama 3.1—8B** (HF checkpoints). Datasets: Alpaca (52k), Dolly 15k, BoolQ, QNLI (GLUE), GSM8K. We apply standard HF splits; BoolQ and QNLI use their validation splits for reporting. Text normalization: UTF-8, strip control characters, collapse repeated whitespace, and truncate/pad to the configured

Table 12: **Derivation map: how each mathematical object connects LoRA's budget law to GRIT's geometry multiplier.**

Object	Role in the derivation
ΔL_{pt}	Forgetting target; approximated by curvature-weighted update energy $\frac{1}{2}\Delta w^\top H_{\text{pt}} \Delta w$.
H_{pt}	Pretraining curvature; defines which directions are "sharp" and amplify forgetting.
Σ_Δ	Update covariance; yields expected forgetting via $\frac{1}{2}\text{tr}(H_{\text{pt}}\Sigma_\Delta)$.
$P^\top H_{\text{pt}} P$	Restricted curvature inside the adapter subspace; this is what geometry-aware PEFT modifies.
LoRA power law	Budget-only predictor $L^0_{\text{pt}} + AD^\beta_{\text{ft}}/N^\alpha + E$.
Ξ_{GRIT}	Geometry multiplier summarized as an "effective capacity" scaling $N \mapsto \Xi_{\text{GRIT}} N$.

max sequence length. Prompt formats for instruction datasets follow the Llama instruction template provided in the repo (`templates/llama_inst_v1.json`). For GSM8K we evaluate using generative (exact-match) metric; we use the official answer normalization script.

GRIT configuration (default unless stated). We quantize the backbone with 4-bit NF4 weights and `bf16` compute (QLoRA setting). Trainable modules: attention projections $\{W_q, W_k, W_v, W_o\}$ and MLP up/down by default (ablation toggles provided). Initial LoRA rank $r_{\max} \in \{8, 16, 32\}$ depending on model size and task; dynamic rank adaptation reduces the *effective* rank online. K-FAC is applied *in rank space*. Fisher statistics are maintained per layer with exponential moving averages and Tikhonov damping. Neural reprojection is executed periodically based on curvature-sample gates. Full hyperparameters appear in Table 18; per-task overrides in Table 19 in appendix.

Note. We use AdamW on K-FAC–preconditioned gradients for stability and apples-to-apples parity with QLoRA; since rank-space K-FAC is an approximate natural-gradient proxy (not exact F^{-1}), this does not claim strict natural-gradient invariance.

Training determinism and seeds. We fix RNG seeds for Python, NumPy, and PyTorch; enable `torch.backends.cudnn.deterministic=True` and `benchmark=False`; fix dataloader shuffles with `generator=torch.Generator().manual_seed(seed)` and `worker_init_fn`. We run 3 seeds {41, 42, 43} and report mean $\pm$ std where relevant. Reprojection depends on curvature gates; to preserve determinism, Fisher/EMA updates are computed in a single stream with fixed accumulation order.

Evaluation protocol. We report exact-match accuracy (GSM8K), GLUE metrics (QNLI), and reference-based metrics (ROUGE-1/2/L, BLEU, BERTScore) using pinned versions of `evaluate`. Decoding for generative metrics uses greedy or temperature 0.2/top-p 0.95 as specified in configs; we fix `max_new_tokens=256` unless the dataset requires otherwise. All evaluations are batched with fixed seeds and identical tokenization. For GSM8K, we use the official answer normalization; we also log per-question chains for auditability.

Per-task overrides. Table 20 lists the main overrides relative to the defaults above. Additional long-run controls: `reprojection_warmup_steps=500`, `rank_adaptation_start_step=500`; optional NG warmup; and curvature/reprojection regularizers with warmup (λ_K=10^{-5}, λ_R=10^{-4}).

Table 13: **Mechanism → geometry effect → proxy term in Ξ_{GRIT}.**

Component	Geometry effect (trace view)	Proxy term in Ξ_{GRIT}
Rank-space K-FAC preconditioning	Shrinks steps along ill-conditioned/sharp restricted directions; reduces curvature-amplified motion in $\mathrm{tr}(P^\top H_{\mathrm{pt}} P\, \Sigma_\theta)$.	Improves alignment/stability; reflected primarily through ρ_{align} and indirectly through higher usable r_{eff}.
Fisher-guided reprojection	Rotates/filters the adapter basis toward dominant stable eigendirections; suppresses noisy diffuse components that increase interference.	Directly increases ρ_{align} and π_{proj}.
Dynamic rank adaptation	Allocates capacity where spectrum has energy; prevents over-spreading energy into weak/noisy directions.	Controls r_{eff} as the *realized* capacity.

Runtime and Overhead

As summarized in Table 14, **GRIT** incurs a single-digit mean step-time overhead ($\sim$6—10%) relative to QLoRA while remaining close in peak memory (+0.5—1.0 GB), with occasional P99 spikes aligned to sparse reprojection events. Note that our reported trainable-parameter reductions correspond to the *effective active rank/update footprint* after reprojection (we still store the full $r_{\max}$ LoRA tensors during training), so peak VRAM is typically similar to QLoRA and can be slightly higher due to rank-space statistics and projection overhead (Table 14).

Across IF, NLI, and GSM8K, GRIT's mean step time is competitive with *Orthogonal-LoRA* and *DoRA*/*Eff-FT*, and substantially lower than *Shampoo*, while *IA*^{3} remains the lightest method by peak memory (Table 14).

Under a fixed 200k-token budget, GRIT's wall-clock remains within 0.5—1.2 hours of QLoRA on the 8B backbone, reflecting the small amortized cost of $r \times r$ covariance updates and infrequent basis reprojections (Table 14).

Notably, the adaptive cadence (Δ eigen-mass + hysteresis) yields only 1.8—2.4 reprojections per 1k steps across tasks, explaining the modest P99 inflation without impacting average throughput (Table 14). In synchronous data-parallel training, reprojection can be scheduled on the same global steps across ranks (or the basis computed once and broadcast), so these spikes are synchronized events rather than per-rank stragglers and the throughput impact is governed by their low frequency (#Reproj/1k). Likewise, rank-space Fisher/K-FAC factors are only $r \times r$ per module and can be kept local per rank or synchronized at inversion/reprojection cadence; communication volume is small, though at very large scale latency can still be a practical bottleneck.

Config. A100 80GB; 4-bit NF4 weights + `bf16` compute (for QLoRA/GRIT); seq len 2,048; global batch = 128 tokens/step (effective); grad acc = 8; AdamW; eval disabled during timing. *Fixed token budget:* 200k tokens. *Backbone:* LLaMA-3-8B. *Tasks:* Inst-Follow (IF), NLI, GSM8K.

Ablations and controls. The code exposes switches for: disabling K-FAC (first-order baseline in the same subspace), disabling reprojection, fixing rank (no dynamic adaptation), attention-only vs. MLP-only adapters, rank grids $\{4, 8, 16, 32\}$, reprojection intervals $\{100, 200, 400\}$, and damping grids $\{10^{-4}, 10^{-3}, 10^{-2}\}$. Each ablation inherits all other settings from the default `.yaml` to isolate the targeted factor.

Compute budget and runtime. On a single A100 40GB, *3B* runs typically require 8—14 GPU hours per task; *8B* runs require 18—30 hours. K-FAC rank-space updates add $\approx$ 6—10% step overhead; reprojection adds a short burst (≤ 0.5 s) every T_{proj} steps for $r \times r$ eigendecompositions (negligible at $r \leq 32$). Peak memory: 24—32 GB for 3B, 36—44 GB for 8B with NF4+bf16.

Licensing, data usage, and ethics. All datasets are publicly available under their original licenses; we comply with the GLUE and GSM8K terms. Our code is released under a permissive research license; see `LICENSE`. We provide `DATA_CARDS.md` with dataset origins and preprocessing steps.

How to reproduce. After creating the provided conda env, run:

```
make train TASK=alpaca
MODEL=llama-3.2-3b \
  SEED=42 CFG=configs/grit_llama3b.yaml \
   OUT=./runs/alpaca_llama3b_s42

make eval TASK=alpaca \
  CKPT=./runs/alpaca_llama3b_s42/best.pt
```

Table 14: Compact runtime/overhead summary across baselines and **GRIT**. GRIT keeps heavy ops in $r \times r$ and uses sparse reprojections, yielding single-digit % mean step-time overhead vs. QLoRA. P99 spikes for GRIT align with reprojection events.

Task	Method	Mean step (ms)	P99 (ms)	Peak mem (GB)	#Reproj/1k	Wall-clock @200k (h)
IF	LoRA	215	290	38.6	–	12.3
	QLoRA	228	305	32.4	–	13.0
	Orthogonal-LoRA	223	298	38.9	–	12.6
	IA^3	205	282	31.7	–	11.8
	DoRA/Eff-FT	240	325	36.1	–	13.7
	Shampoo	268	360	40.2	–	15.1
	GRIT	**236**	**318**	**33.1**	**2.1**	**13.5**
NLI	LoRA	210	285	38.2	–	12.0
	QLoRA	224	300	32.2	–	12.8
	Orthogonal-LoRA	219	292	38.5	–	12.4
	IA^3	202	278	31.6	–	11.6
	DoRA/Eff-FT	238	322	35.9	–	13.5
	Shampoo	264	355	39.9	–	14.9
	GRIT	**232**	**314**	**32.9**	**1.8**	**13.3**
GSM8K	LoRA	222	298	38.9	–	12.7
	QLoRA	235	312	32.6	–	13.4
	Orthogonal-LoRA	229	305	39.1	–	13.0
	IA^3	212	290	31.9	–	12.1
	DoRA/Eff-FT	244	330	36.5	–	13.9
	Shampoo	272	365	40.6	–	15.3
	GRIT	**242**	**326**	**33.3**	**2.4**	**13.9**

Reading. **Overhead:** GRIT adds ∼6—10% mean step-time over QLoRA; P99 spikes coincide with reprojection events. **Memory:** GRIT ∼QLoRA (+0.5—1.0 GB) due to rank-space stats; IA^3 is the lightest.

This invokes the exact configuration used in the paper (commit hash recorded in `runs/*/meta.json`). The same applies to other tasks/models via `TASK={dolly15k,boolq,qnli,gsm8k}` and `MODEL={llama-3.2-3b,llama-3.1-8b}`.

Deviations and caveats. The only hardware deviation is the RTX 6000 Ada run for *8B/QNLI*. We observed no metric drift beyond expected RNG jitter. If reproducing on alternative drivers/CUDA, minor numeric differences may arise; we recommend re-running all three seeds to match reported means.

Artifact checklist. *Repo contents:* code; configs (`.yaml`); env files; scripts for training/eval; seeds; logs; metric JSON; ablation scripts; plotting scripts for spectra/effective ranks; and `READMEs` with end-to-end instructions. All figures are generated from logged runs via `scripts/plot_*.py`; we provide notebooks to regenerate Figures 2, 3 and 5.

Discussions

What GRIT contributes. *GRIT* reframes PEFT as *geometry-aware* optimization by coupling (i) rank-space K-FAC to approximate natural gradients and temper motion in sharp directions, (ii) *neural reprojection* that rotates the adapter basis toward Fisher-dominant eigendirections, and (iii) *dynamic rank* that concentrates capacity where the spectrum has mass. Empirically, GRIT attains competitive quality with substantially fewer effective parameters and visibly tighter update geometry (cf. Figs. 2—1).

Interpretation. Two-sided GRIT allocates capacity to modules whose *rank-space Fisher energy*"the cumulative mass of eigenvalues $\{\lambda_i^{(F)}\}$"persists across intervals. Dominant allocation to `o_proj` matches attention-output fusion concentrating curvature; lower k on `v_proj` reflects diffuse value projections. In MLP, `up_proj/gate_proj` exceed `down_proj`, consistent with expansion vs. compression. Layer-wise, k rises in mid/late blocks as features specialize.

Geometry-first fine-tuning. A key takeaway is that *where* we move in parameter space matters as much as *how much*. Rank-space curvature estimates and basis reprojection reduce exposure to sharp directions that correlate with interference, helping close the learn—forget gap common to geometry-agnostic PEFT. This lens suggests future PEFT design should co-optimize *(loss, curvature, subspace)* rather than loss alone.

Concretely, we observe lower curvature exposure $\bar{\kappa} = \mathrm{tr}(P\, H_{\mathrm{pt}}\, P)$ under GRIT versus LoRA at fixed D_{ft} and N, consistent with smaller projections onto sharp modes and reduced drift.

Scope of evidence. Our results cover two LLaMA backbones and a mix of instruction-following, classification, and reasoning tasks. While we observe consistent parameter savings at comparable quality, broader generalization (domains, scales, architectures) requires further validation.

- **Curvature estimation bias.** Rank-space K-FAC assumes Kronecker separability and relies on finite-sample covariances; early-phase Fisher is noisy. We mitigate with damping, EMA smoothing, and warm-up gates, but residual bias may under- or over-allocate rank. Reporting spectra and $k(t)$ traces aids auditability.
- **Projection frequency sensitivity.** The reprojection period T_{proj} trades stability and compute. We use hysteresis and sample gates; principled schedules (e.g., trust-region criteria) remain future work.
- **Backend coupling.** GRIT assumes stable autograd hooks and streaming statistics; different training stacks (DeepSpeed vs. FSDP) can shift the compute/memory envelope. We include configs and seeds for exact replication.
- **Task breadth and scale.** Evaluations cover two LLaMA backbones and five benchmarks. Generalization to multimodal, multi-turn agents, or RLHF stacks is untested.
- **Forgetting quantification.** We use pretraining-loss proxies and broad-ability suites; gold-standard drift measures (e.g., pretraining-corpus log-likelihood) are expensive and approximated here.

Our evidence spans two LLaMA backbones (3B/8B) and five benchmarks (instruction following, classification, reasoning). While GRIT consistently reduces trainable parameters at comparable quality, three factors limit external validity: (i) *curvature estimation bias* from rank-space K-FAC under finite samples; (ii) *schedule sensitivity* to the reprojection period and τ; and (iii) *stack dependence* on FSDP/DeepSpeed configurations. We provide seeds, configs, and logging hooks (Fisher spectra, $k(t)$, π_{proj}) to facilitate independent verification and stress-testing on other domains and architectures.

Spectrum-Driven Rank Allocation

Rationale. Low-rank adapters provide a fixed *capacity budget* per layer/module, yet curvature and signal are not uniformly distributed across depth or pathways. GRIT therefore *allocates rank where the spectrum has mass*: let $\{\lambda_i^{(F)}\}_{i=1}^{r_{\max}}$ denote the rank-space Fisher eigenvalues for a given module at a checkpoint. We select the effective rank

$$k \;=\; \min\Big\{ j \;\Big|\; \frac{\sum_{i=1}^{j} \lambda_i^{(F)}}{\sum_{i=1}^{r_{\max}} \lambda_i^{(F)}} \geq \tau \Big\}, \qquad k \in [r_{\min},\, r_{\max}],$$

with energy threshold τ (default 0.90) and bounds $(r_{\min}, r_{\max})$ to avoid collapse or runaway growth. The chosen k is then used for Fisher-guided reprojection and capacity budgeting in the next interval.

What the heatmap shows. Figure 11 visualizes the *final effective rank* k per layer/module on QNLI with Llama-3.1 8B. Three consistent patterns emerge: (i) attention `o_proj` receives the highest k, indicating concentrated curvature where attention outputs are fused downstream; (ii) `q_proj`/`k_proj` are moderate and `v_proj` is lowest, consistent with values dispersing signal across heads; (iii) within MLP blocks, `up_proj`/`gate_proj` attract larger k than `down_proj`, aligning with expansion vs. compression roles. Across depth, k increases in mid—late layers where features specialize.

We use $\tau = 0.90$, EMA $\rho = 0.95$, hysteresis $\delta = \pm 0.02$, and a per-module minimum of $N_{\min} = 4096$ curvature samples; k updates occur every $T_{\mathrm{proj}} = 200$ steps.

Implications. By adapting k to the observed spectrum, GRIT *spends rank where it buys curvature alignment*, yielding sparser yet more targeted updates without sacrificing quality. Practically, reporting per-module k maps and Fisher spectra makes capacity placement *auditable*, aiding reproducibility and model diagnostics.

Stability measures. To prevent jitter when eigen-gaps are small, GRIT uses (a) warm-up gating on minimum curvature samples, (b) exponential smoothing of energy curves, and (c) hysteresis around τ. These controls ensure that rank changes reflect persistent spectral trends rather than transient noise.

Limitations and Mitigation Strategies

Broader implications. Coupling *curvature* and *subspace* aims to convert parameter efficiency into *reliable* adaptation. Let H_{pt} denote the pretraining Hessian (or F a Fisher proxy) and P the projector onto the adapter subspace. Empirically lower curvature exposure $\overline{\kappa} = \mathrm{tr}(PH_{\mathrm{pt}}P)$ (or $\mathrm{tr}(PFP)$) aligns with reduced drift [Pascanu et al., 2013; Ghorbani et al., 2019; Keskar et al., 2017]; robust gates, spectral hysteresis, and uncertainty-aware schedules are therefore central to safe deployment.

Reporting protocol for geometry-aware PEFT. To make results *auditable at scale*, report: (i) per-layer Fisher spectra and cumulative energy $E(j)$; (ii) effective-rank trajectories r_{eff} under a fixed τ; (iii) curvature exposure $\mathrm{tr}(PH_{\mathrm{pt}}P)$ or $\mathrm{tr}(PFP)$; (iv) update geometry (tail mass U_{hi}, norms, sparsity); (v) forgetting proxies (pretraining-loss deltas, zero-shot retention); and (vi) compute overhead of geometry steps. This aligns with scaling-law and continual-learning diagnostics [Bethune et al., 2025; Kirkpatrick et al., 2017; Zenke et al., 2017; Aljundi et al., 2018]

Table 15: Consolidated main results on Llama-3.2 3B and Llama-3.1 8B across all tasks. Each block reports absolute scores; bold indicates best-in-block. The "# Param. Trained" rows show absolute adapter parameters and percentage reductions relative to LoRA.

Across tasks, GRIT matches LoRA/QLoRA within 1—3% on median metrics while reducing trainable parameters by 30—68% (model-dependent). Best per-block score in bold.

Models →		Llama-3.2 3B				Llama-3.1 8B			
Datasets ↓	Metrics ↓	LoRA	QLoRA	GRIT	Q-GRIT	LoRA	QLoRA	GRIT	Q-GRIT
ALPACA	ROUGE-1	**0.1852**	0.1292	0.1844	0.1455	**0.2036**	0.1402	0.2034	0.1698
	ROUGE-2	**0.0825**	0.0562	0.0818	0.0649	**0.0923**	0.0616	**0.0923**	0.0818
	ROUGE-L	**0.1426**	0.0983	0.1425	0.1127	**0.1528**	0.1047	**0.1528**	0.1327
	BLEU	**0.0443**	0.0235	0.043	0.0222	**0.0492**	0.0259	**0.0492**	0.028
	BERT SCORE	0.8343	0.7948	**0.8354**	0.7986	0.831	0.7949	**0.831**	0.8173
	# Param. Trained	24.31M	24.31M	**8.45M (65.3% ↓)**	**8.45M (65.3% ↓)**	83.89M	83.89M	**19.27M (77% ↓)**	**30.85M (63.2% ↓)**
Dolly-15k	ROUGE-1	0.1733	0.1108	**0.1976**	0.1195	0.1921	0.1272	**0.1968**	0.1954
	ROUGE-2	0.0824	0.0519	**0.0994**	0.0592	0.0927	0.0591	0.0969	**0.0937**
	ROUGE-L	0.1368	0.0884	**0.1568**	0.0968	0.1454	0.095	**0.1552**	0.1471
	BLEU	0.0533	0.0297	**0.056**	0.0304	**0.0592**	0.0334	**0.0592**	0.0579
	BERT SCORE	0.8295	0.8005	**0.8344**	0.8026	**0.8379**	0.8128	0.8377	**0.838**
	# Param. Trained	24.31M	24.31M	**17.01M (30% ↓)**	**17.01M (30% ↓)**	83.89M	83.89M	**54.3M (35.2% ↓)**	**38.14M (54.5% ↓)**
GLEU-QNLI	ACCURACY	0.8938	0.8885	**0.9053**	0.8449	**0.9297**	0.9248	0.9211	0.9154
	PRECISION	0.8939	0.8971	**0.9059**	0.8663	**0.9298**	0.9257	0.9213	0.9155
	RECALL	0.8939	0.8893	**0.9055**	0.8462	**0.9298**	0.9245	0.9212	0.9154
	F1	0.8938	0.888	**0.9052**	0.8429	**0.9297**	0.9247	0.9211	0.9154
	# Param. Trained	24.31M	24.31M	**7.75M (68.1% ↓)**	**7.75M (68.1% ↓)**	83.89M	83.89M	**16.6M (80% ↓)**	**29.47M (64.9% ↓)**
BoolQ	ACCURACY	**0.7834**	0.7525	0.7749	0.7421	**0.8345**	0.8229	0.8336	0.8201
	PRECISION	**0.7982**	0.7491	0.7908	0.754	0.8479	0.8891	0.8563	**0.8941**
	RECALL	0.872	**0.905**	0.8671	0.8686	**0.8942**	0.8169	0.8799	0.8061
	F1	**0.8335**	0.8197	0.8272	0.8072	**0.8704**	0.8515	0.868	0.8478
	# Param. Trained	24.31M	24.31M	**15.03M (38.2% ↓)**	**15.03M (38.2% ↓)**	83.89M	83.89M	**61.56M (26.6% ↓)**	**61.56M (26.6% ↓)**
GSM8k	ACCURACY	**0.3935**	0.3836	0.3867	0.3779	0.6338	0.6315	**0.6619**	0.6224
	# Param. Trained	24.31M	24.31M	**15.3M (37% ↓)**	**17.43M (28.3% ↓)**	83.89M	83.89M	**60.5M (27.8% ↓)**	**67.57M (19.5% ↓)**

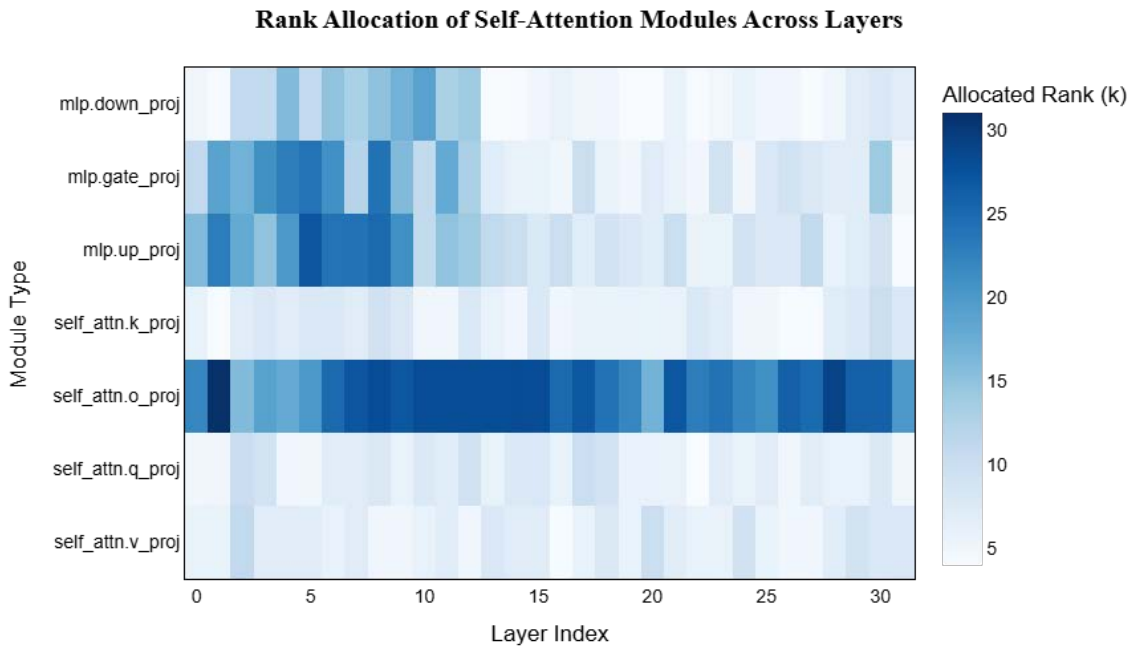


Figure 11: **Rank allocation across layers and module types (QNLI, Llama-3.1 8B, GRIT).** Heatmap shows the *final effective rank* k per layer and module type at training end. GRIT concentrates capacity on `self_attn.o_proj` (largest k), with `q_proj`/`k_proj` moderate and `v_proj` lowest; in MLP, `up_proj` and `gate_proj` receive higher k than `down_proj`. Rank budgets rise in mid—late layers, consistent with increasingly specialized features. The 8B model follows the same pattern observed at 3B with higher absolute k, corroborating *spectrum-driven rank allocation* under two-sided GRIT.

Table 16: **GRIT limitations and practical mitigations.** Each entry is citation-anchored; several mitigations are already implemented (warm-ups, damping, EMA smoothing, gated reprojection).

Limitation	Mitigation / Future Direction
Early-stage Fisher under-sampling Rank-space covariances are noisy early in training, which can destabilize K-FAC preconditioning and any Fisher-guided reprojection [Amari, 1998; Martens and Grosse, 2015].	Warm-up gates for natural-gradient (*NG*) and reprojection; escalating damping and EMA smoothing (ρ=0.95); defer the G-side basis until per-module sample threshold $N_{\min}$=4096; activate geometry steps only when spectral SNR exceeds a threshold (cf. second-order stabilization heuristics [Martens and Grosse, 2015]).
Projection frequency sensitivity Too-frequent reprojection can over-rotate the subspace; too-infrequent allows drift away from informative eigendirections.	Adaptive periods keyed to spectral mass change, e.g., trigger when $\Delta E_k = (\sum_{i\le k}\lambda_i / \sum_{i\le r}\lambda_i)$ exits a hysteresis band ($\delta = \pm 0.02$); trust-region style triggers using curvature-aligned energy [Schulman et al., 2015]; report ablations over T_{proj}.
Overspecialization risk Strict alignment to dominant eigen-directions may overspecialize to the current slice, reducing out-of-slice generalization [Keskar et al., 2017; Dinh et al., 2017].	Stochastic subspace mixing (probabilistically drop top eigenvectors); entropy floors on spectra; mixed-domain mini-batches to refresh curvature; periodic anti-collapse regularizers on k (connects to intrinsic-dimension/low-rank generalization arguments [Aghajanyan et al., 2021]).
Rank selection fragility Energy threshold τ and `min_rank` influence stability and capacity allocation; small eigen-gaps induce jitter.	Hysteresis for k updates (change only if margin $> \epsilon$); EMA smoothing of spectra; per-layer priors on k; log and checkpoint $k(t)$ for reproducibility; relate choices to effective-rank theory [Roy and Vetterli, 2007; Gavish and Donoho, 2014].
Compute overhead at scale Maintaining covariances and inverting small matrices adds latency vs. pure first-order PEFT.	Keep all ops in rank space ($r \times r$); amortize inversions (update every > 1 step); CPU caching of factors; mixed-precision solves with jitter; share statistics across similar modules; compare to alternative factored second-order methods (e.g., Shampoo) when relevant [Anil et al., 2021].
Interaction with quantization NF4+`bf16` can bias covariances/inverses at very low ranks, affecting K-FAC statistics [Dettmers et al., 2023].	Quantization-aware damping; periodic de-quantized refresh of statistics; calibration runs to bound numeric drift; enable G-basis only after robust sample counts (as in QLoRA stability guidance [Dettmers et al., 2023]).
Model-/task-specific tuning Damping, gates, and projection frequency do not trivially transfer across backbones/tasks.	Provide defaults with robust ranges; small validation sweeps on spectral stability and forgetting proxies [Bethune et al., 2025; Biderman et al., 2024]; meta-schedules conditioned on observed curvature norms (analogous to trust-region step-size control [Schulman et al., 2015]).
Coverage of evaluation Benchmarks emphasize short-context, English tasks; continual/robustness stress is limited.	Extend to long-context, multilingual, domain-shift, and continual-learning settings; add forgetting audits (pretraining-loss probes) and retention measures [Kirkpatrick et al., 2017; Zenke et al., 2017; Aljundi et al., 2018].

Component	Version / Setting	Component	Version / Setting
OS	Ubuntu 22.04 LTS	CUDA Toolkit	12.1
Python	3.10.13	cuDNN	9.x
PyTorch	2.3.1+cu121	PyTorch Distributed	NCCL 2.20
Transformers	4.57.x	Datasets	2.19.x
bitsandbytes (NF4)	0.43.x	peft	0.11.x
Accelerate	1.12.x	SentencePiece	0.2.0
FlashAttention-2	2.5.x (optional)	K-FAC backend	custom (rank-space)
Tokenizer	Llama tokenizer (HF)	WandB/MLFlow	optional logging

Table 17: **Environment & Framework Versions (pinned)**

Table 18: **GRIT hyperparameters (shared defaults).** Symbols match those used in the paper.

Component	Setting	Component	Setting
Quantization	4-bit NF4 weights, `bf16` compute	Max seq len	2048 (Alpaca/Dolly), 512 (BoolQ/QNLI), 1024 (GSM8K)
Optimizer	AdamW on preconditioned grads	AdamW (β_1, β_2)	$(0.9,\ 0.95)$
Weight decay	0.0 for adapters	LR schedule	cosine decay, 5% warmup
Base LR	2.0×10^{-4} (3B), 1.5×10^{-4} (8B)	Grad clip	1.0 (global norm)
Global batch	128 tokens/GPU $\times$ GA $\rightarrow$ 256k tokens/step	Epochs/steps	see Table 19
LoRA rank $r_{\max}$	16 (3B), 32 (8B) unless stated	LoRA α	32 (3B), 64 (8B)
Trainable modules	Attn proj. + MLP up/down	Dropout (adapters)	0.0
K-FAC (rank space)	Update every 50 steps; EMA $\rho = 0.95$	Damping λ	10^{-3} (auto-tuned $\pm \times 10$)
Covariances	$\Sigma_{a,t} = \mathbb{E}[a_r a_r^\top],\ \Sigma_{g,t} = \mathbb{E}[g_r g_r^\top]$	Inversion	Cholesky, jitter $+10^{-6} I$
Reprojection	Every $T_{\text{proj}} = 200$ steps (gated)	Gate	min samples/layer $N_{\min} = 4096$
Projection basis	top-k Fisher eigenvectors (rank space)	k selection	effective-rank threshold $\tau = 0.90$
Dynamic rank	$r_{\text{eff}}(t) = \min\{k : \sum_{i=1}^{k} \lambda_i / \sum_i \lambda_i \geq \tau\}$	Bounds	$r_{\min} = 4$, $r_{\max}$ as above
Seeds	{41, 42, 43} (default 42)	Logging	deterministic dataloader order

Table 19: **Per-task schedules (& overrides).** Steps shown for 3B; the 8B model uses the same token budgets with proportionally longer wall-clock.

Task	**Tokens**	**Steps**	**Warmup**	**Eval freq**	$r_{\max}$	T_{proj}
Alpaca	1.0B	4,000	200	every 250	16 (3B) / 32 (8B)	200
Dolly 15k	0.5B	2,000	100	every 200	16 / 32	200
BoolQ	0.25B	1,200	60	every 100	16 / 32	200
QNLI	0.25B	1,200	60	every 100	16 / 32	200
GSM8K	0.6B	3,000	150	every 200	16 / 32	200

Table 20: Key hyperparameters by task. Unless specified, batch size is 8, gradient accumulation is 4, epochs=3, learning rate 2×10^{-5}.

Task	Batch	Grad-Acc.	Epochs	LR	`kfac_min`	`kfac_upd`	Damping
Alpaca	8	4	3	2×10^{-5}	256	150	0.003
Dolly-15k	8	4	3	1×10^{-4}	256	150	0.003
BoolQ	8	4	3	2×10^{-5}	256	150	0.005
QNLI	32	4	2	2×10^{-5}	256	150	0.005
GSM8K	8	4	3	2×10^{-5}	256	150	0.005

Ethical Consideration

Scope and intent. This work introduces *GRIT*, a geometry-aware, parameter-efficient fine-tuning (PEFT) method that modifies *how* adaptation proceeds, not *what* data are used or which capabilities are unlocked. We position GRIT within standard model-governance practices (e.g., model cards, datasheets, and data statements) to ensure transparency around intended use, training data provenance, and evaluation scope [Mitchell et al., 2019; Gebru et al., 2021; Bender and Friedman, 2018].

Dual use and misuse. Lowering the cost of adaptation can enable beneficial customization *and* harmful repurposing (e.g., spam, fraud, disinformation). We therefore advocate (i) release strategies conditioned on risk, consistent with staged disclosure and use-policy alignment [Solaiman et al., 2019; Weidinger et al., 2021]; (ii) integrating *red teaming* and adversarial audits (prompt attacks, jailbreaks) into any GRIT deployment [Perez et al., 2022; Zou et al., 2023]; and (iii) publishing *auditable geometry traces* (Fisher spectra, effective ranks) to diagnose suspicious training dynamics and drift.

Bias and fairness. GRIT alters update *geometry* rather than content, and thus can propagate pre-existing biases if data or objectives are skewed. We recommend slice-aware, dataset-grounded evaluation (toxicity, demographic performance, and robustness) with established probes and taxonomies [Gehman et al., 2020; Blodgett et al., 2020; Sheng et al., 2019]. We further encourage coupling GRIT with documentation artifacts (model cards/datasheets) and accountability practices [Mitchell et al., 2019; Gebru et al., 2021; Raji et al., 2020].

Privacy. Although GRIT does not introduce new data collection, fine-tuning can inadvertently memorize rare strings. We recommend data deduplication and PII scrubbing where feasible, and post-hoc membership/memorization checks [Carlini et al., 2019, 2021; Shokri et al., 2017]. Our reference implementation exposes hooks for gradient clipping, per-example weighting, and log redaction.

Environmental impact. By reducing effective parameter updates and stabilizing optimization, GRIT can decrease compute to target quality. We will report estimated energy/CO_2e per run and provide configuration defaults (lower ranks, early stopping via geometry metrics) aligned with established footprint reporting practices [Strubell et al., 2019; Henderson et al., 2020; Lacoste et al., 2019].

Transparency, reproducibility, and auditing. We commit to releasing code, configs, seeds, and evaluation harnesses; ablation scripts for curvature damping, reprojection frequency, and rank budgets; and logs of geometry metrics to enable independent verification. This aligns with evolving reproducibility norms and checklists in ML [Pineau et al., 2021; Liang et al., 2022]. We also recommend licensing under responsible-AI terms (e.g., RAIL) to bind usage to acceptable-intent policies [Responsible AI Licenses (RAIL) Initiative, 2023].

Limitations and open risks. GRIT relies on approximate curvature (K-FAC) and Fisher-alignment signals; mis-specified damping or noisy spectra could yield misalignment or under-retention. Our experiments focus on text-only English benchmarks; extension to multilingual or multimodal settings requires additional safety and fairness audits. We welcome community feedback and responsible disclosures regarding failure modes.