SLORR: SIMPLE AND EFFICIENT IN-TRAINING LOW-RANK REGULARIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Direct spectral regularization is a natural way to induce low-rank weights, but common objectives are costly at modern scale because they typically require repeated singular value decompositions (SVDs). Existing alternatives reduce this cost through reformulations that introduce architectural reparameterization, cached spectral state, predefined target ranks, or periodic decompositions, thus hindering practicality. We introduce SLORR to make direct spectral regularization practical without these restrictions while retaining competitive post-compression performance. Our SVD-free, stateless, architecture-preserving framework uses GPU-friendly approximations of the spectral quantities needed for regularizer forward and backward passes. SLORR supports different spectral objectives, instantiated here with Hoyer spectral sparsity and nuclear norm regularization. We evaluate induced compressibility and performance after post-training low-rank compression across different variants, using ImageNet-1K pretraining and continued training of ResNets and Vision Transformers, and LLM pretraining at 135M, 560M, and 1B parameters. Across vision settings and at 135M LLM scale, SLORR achieves competitive compression–quality tradeoffs with recent strong low-rank regularizers, often with greater robustness across settings, while retaining low training overhead and avoiding their limitations. We further scale SLORR to 560M and 1B LLMs: it substantially improves post-compression performance over matched unregularized training. These gains persist across different language model token budgets and four optimizers.

1 INTRODUCTION

Low-rank factorization reduces the computational and memory costs of neural networks by replacing dense weight matrices with products of smaller factors (Ben Noach & Goldberg, 2020; Denton et al., 2014; Jaderberg et al., 2014; Chen et al., 2021). However, compression quality depends on weight spectra: discarding meaningful spectral directions can substantially degrade performance, making modern models difficult to compress aggressively without accuracy loss.

This motivates training-time methods that encourage low-rank structure before compression, for example, by promoting sparsity in the singular values. The natural approach is direct spectral regularization, explored in prior work (Alvarez & Salzmann, 2017; Xu et al., 2020). However, it requires large singular value decompositions (SVDs) at every training step, making it prohibitively expensive at modern scale. Existing alternatives mostly avoid this by reformulating the problem, but they generally introduce other limitations that hinder their practicality: factorized parameterizations change the architecture and optimization dynamics (Yang et al., 2020a; Alkhouri et al., 2024), and methods based on periodically cached spectral quantities remain stateful, retaining some SVD cost (complicating scalability). They often introduce additional hyperparameters (Ghosh et al., 2025). A summary of directly relevant related works appears in Table 1, with a more extensive discussion in Section 2.

In this work, we ask whether direct spectral regularization can instead be made practical at modern scale without these restrictions while retaining competitive post-compression performance. We answer positively with SLORR, an SVD-free, stateless, architecture-preserving framework that operates directly on the original weight matrices and directly optimizes the spectral objective. The key idea is to approximate the spectral quantities required for the regularizer forward and backward passes

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

Table 1: **Comparison of different low-rank-inducing regularizers**. Detailed discussion in Section 2.

Method	Changes arch.	SVD	Efficiency/scalability	Stateful	Target rank
SVD-based (Alvarez & Salzmann, 2017)	No	Yes	Poor	No	No
Factorized reg. (Yang et al., 2020a)	Yes	No	Good	No	No
LoRITa (Alkhouri et al., 2024)	Yes	No	Good	No	No
Q3R (Ghosh et al., 2025)	No	Periodic	Moderate	Yes	Yes
SLORR (ours)	**No**	**No**	**Good**	**No**	**No**

using GPU-efficient polar-factor approximations. Importantly, SLORR supports different spectral objectives. Here, we instantiate it with Hoyer spectral sparsity (Hoyer, 2004) and nuclear norm regularization. For both instantiations, the resulting approximations admit convergence guarantees for the regularizer values and gradients (Proposition 3.1).

Empirically, we evaluate SLORR across vision and language modeling settings. On ImageNet-1K (Deng et al., 2009), we study short-horizon continued training of ResNet-50 (He et al., 2016), ViT-B/16, and ViT-L/16 (Dosovitskiy et al., 2021), as well as ResNet-18 pretraining (He et al., 2016). We evaluate SLORR-Hoyer and SLORR-Nuc in LLM pretraining at 135M, 560M, and 1B scales. Across the vision settings and at 135M LLM scale, SLORR achieves competitive compression-quality tradeoffs with recent low-rank regularizers, often with greater robustness across settings. At 560M and 1B, it substantially improves post-compression performance over matched unregularized training. Measured SLORR-Hoyer training overhead remains below 3% in our LLM experiments and below 8% in the vision ones. Additional experiments across compression procedures, token budgets, and three alternative optimizers demonstrate SLORR's generality.

Overall, our contributions are as follows. First, we introduce an SVD-free framework for low-rank-inducing regularization using GPU-friendly approximations and instantiate it with two objectives. Second, we show how this framework can be used to regularize the spectra of the original weight matrices without architectural changes, cached spectral quantities, or a pre-specified target rank. Third, we demonstrate across vision and language modeling experiments that SLORR improves post-training low-rank compressibility with small training overhead. Overall, SLORR generally maintains or surpasses the effectiveness of other baselines (often being more robust across settings) while avoiding their limitations and makes low-rank regularization feasible at larger scales.

2 Background and Related Work

Layer factorization. Various factorization approaches have been extensively studied over time (Denton et al., 2014; Jaderberg et al., 2014; Ben Noach & Goldberg, 2020). In this work, we focus on factorization schemes that separate a single matrix multiplication into two sequential ones:

$$f(\boldsymbol{X};\boldsymbol{W}) = \boldsymbol{X}\boldsymbol{W} \approx \hat{f}(\boldsymbol{X};\boldsymbol{W}_0,\boldsymbol{W}_1) = (\boldsymbol{X}\boldsymbol{W}_0)\boldsymbol{W}_1,$$

where $\boldsymbol{X} \in \mathbb{R}^{B\times D_i}$ is the input, $\boldsymbol{W} \in \mathbb{R}^{D_i\times D_o}$ the weight of the original layer f, and $\boldsymbol{W}_0 \in \mathbb{R}^{D_i\times P}$ and $\boldsymbol{W}_1 \in \mathbb{R}^{P\times D_o}$ are the weights of the new approximate layer $\hat{f}$. With sufficiently small P, we can achieve inference speedups and memory reduction. In particular, we focus on those that employ the SVD of weight matrices (Jaiswal et al., 2025; Ben Noach & Goldberg, 2020), which is optimal in weight space according to the Eckart–Young–Mirsky theorem (Golub & Van Loan, 2013). In particular, the factors can be obtained from the SVD of $\boldsymbol{W}$:

$$\boldsymbol{W}_0 = \boldsymbol{U}_{:,:P}\boldsymbol{\Sigma}^{1/2}_{:P,:P}, \qquad \boldsymbol{W}_1 = \boldsymbol{\Sigma}^{1/2}_{:P,:P}\boldsymbol{V}^{\top}_{:,:P}.$$

In convolutional neural networks (CNNs) (LeCun et al., 1998), SVD-based factorization can be performed by reshaping parameter tensors into matrices and subsequently decomposing them (normally called channel-wise decomposition) (Yang et al., 2020a; Alkhouri et al., 2024); we follow this approach throughout this work. Other recent works, such as activation-aware methods (Qinsi et al., 2025; Wang et al., 2025b;a; Chen et al., 2021; Yuan et al., 2025), introduce different ways of computing factors that can lead to better overall accuracy.

Inducing low-rank parameters during training. SVD truncation discards smaller singular values, motivating training-time regularization that concentrates the spectrum and makes weights approxi-

mately low-rank. Rank minimization is equivalent to promoting sparsity in the singular values:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda \sum_i \text{rank}(\boldsymbol{W}^i) = \mathcal{L}_{\text{task}} + \lambda \sum_i \|\boldsymbol{\sigma}(\boldsymbol{W}^i)\|_0,$$

where $\boldsymbol{\sigma}$ denotes the singular values of $\boldsymbol{W}^i$. This objective is considered to be intractable (Fazel et al., 2004). Prior work has addressed this by using proxy objectives, such as minimizing the nuclear norm (Alvarez & Salzmann, 2017; Xu et al., 2020); however, this generally requires computing the SVDs of each weight at each training step, which is computationally prohibitive for larger models. Several solutions have been proposed.

Reparameterization-based approaches. Some approaches rely on modifying the architecture. For example, Yang et al. (2020a) obtains the SVD $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{\sigma}$ of each layer and substitutes each layer with a factorized form (retaining all singular values). The authors then incentivize $\boldsymbol{\sigma}$ to be sparse through the Hoyer sparsity metric (Hoyer, 2004) (the ℓ_1 norm is also explored). An additional regularizer is required to keep $\boldsymbol{U}$ and $\boldsymbol{V}$ orthogonal to maintain SVD semantics, making it less practical. The recent LoRITa (Alkhouri et al., 2024) reparametrizes each layer weight $\boldsymbol{W}$ with an ordered product $\prod_i^N \boldsymbol{W}_i = \boldsymbol{W}$ and penalizes the Frobenius norm of each $\boldsymbol{W}_i$, which can be shown to work as a proxy for the nuclear norm minimization objective of the overall $\prod_i^N \boldsymbol{W}_i$. A similar approach was proposed in Kliegl et al. (2018) for depth-2 decompositions. These approaches generally increase the number of trainable parameters and can alter optimization dynamics.

Proxy objectives through cached spectral quantities. Another line of work reuses computed factors instead of recomputing them at every iteration. For example, the recent Q3R (Ghosh et al., 2025) uses a surrogate objective that can be approximated with periodic SVDs (hence masking the inefficiency at some scales but not solving the scalability problem). However, it is stateful (as it needs to store the temporary factors until they are recomputed). It also requires specifying a target rank in advance, making hyperparameter tuning challenging.

Other approaches. Other approaches for approximate computations exist: Li et al. (2025b) propose randomized differentiable surrogates for generalized low-rank regularization, applying them to tasks including matrix completion, video foreground/background separation, and denoising. While the method is SVD-free in theory, it is still impractical, requiring backpropagation through long chains of matrix multiplications. Moreover, its practical implementation[1] still requires a (smaller) SVD and matrix inverse. This makes it infeasible for weight regularization, even at moderate scales. Qin et al. (2025) propose a method to make models amenable to activation-aware methods (in particular, SVD-LLM (Wang et al., 2025b)), but it is focused on very short-horizon post-training, while we focus on longer training periods.

Other related works. Relatedly, low-rank induction can also happen (either explicitly or implicitly) at the optimizer level. For example, Zimmer et al. (2024) use a Frank–Wolfe-based optimizer to promote low rank in weight matrices using partial SVDs. Dolatabadi et al. (2026) find that, surprisingly, Muon-trained LLMs are more compressible and that truncated Muon updates can further improve compressibility. Here, we restrict our focus to explicit regularization techniques, typically formulated either as additional loss terms or as decoupled penalties applied outside the primary gradient update, analogous to decoupled weight decay (Loshchilov & Hutter, 2019).

Lastly, a related area, but fundamentally different in nature, is that of low-rank training, where the rank is generally pre-defined before training, and a low-rank architecture is directly trained. Different variants, such as full-rank warmup or combinations with sparse training, and aspects like initialization, have been investigated over time (Lialin et al., 2023; Khodak et al., 2021; Kamalakara et al., 2022; Wei et al., 2024; Han et al., 2024; Li et al., 2025a).

3 Our Framework

For a weight matrix $\boldsymbol{W} \in \mathbb{R}^{I \times O}$, we use $\boldsymbol{U} \in \mathbb{R}^{I \times R}, \boldsymbol{\Sigma} \in \mathbb{R}^{R \times R}, \boldsymbol{V} \in \mathbb{R}^{O \times R}$ to denote its thin SVD factors ($\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$), where $R = \text{rank}(\boldsymbol{W}) \leq \min(O, I)$. Additionally, we use $\boldsymbol{\sigma}$ to denote the vector of singular values ($\boldsymbol{\sigma} = \text{diag}(\boldsymbol{\Sigma})$). Since the SVD is not unique, when mentioning it or any factor, we implicitly refer to an arbitrary valid choice.

[1] https://github.com/naiqili/EDLRR

We view the problem from the perspective of inducing sparsity in singular values. SLORR separates the spectral objective being optimized from the computational mechanism used to evaluate it. In this work, we instantiate our framework with (1) the ℓ_1 norm of singular values, i.e., the nuclear norm, and (2) the squared ℓ_1/ℓ_2-ratio (related to the Hoyer metric (Hoyer, 2004)) of singular values, which is effective as a sparsity-inducing regularizer (Yang et al., 2020b) (the nonsquared version has also been shown to work well in factorization-based low-rank regularization (Yang et al., 2020a)). We have:

$$\mathcal{L}_{\text{hoyer}} = \sum_i \|\boldsymbol{\sigma}^i\|_1^2/\|\boldsymbol{\sigma}^i\|_2^2 = \sum_i \|\boldsymbol{W}^i\|_*^2/\|\boldsymbol{W}^i\|_F^2, \quad \mathcal{L}_{\text{nuc}} = \sum_i \|\boldsymbol{W}^i\|_*,$$

where i denotes the layer index. From now on, we will omit layer indices for brevity. We optimize these regularizers together with the task loss. Strictly speaking, the regularizers are not differentiable everywhere. We use the natural backpropagation rule, which corresponds to the minimum-Frobenius-norm element of the Clarke generalized gradient (see Appendix A for the derivations). We use gradient notation for simplicity. We have that:

$$\nabla\mathcal{L}_{\text{hoyer}}(\boldsymbol{W}) = 2\frac{\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F}\left(\frac{1}{\|\boldsymbol{W}\|_F}\boldsymbol{U}\boldsymbol{V}^\top - \frac{\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^3}\boldsymbol{W}\right), \quad \nabla\mathcal{L}_{\text{nuc}}(\boldsymbol{W}) = \boldsymbol{U}\boldsymbol{V}^\top. \tag{1}$$

Both gradients have terms that are generally computed through SVDs, which is prohibitively expensive to do at every iteration. Fortunately, iterative methods can approximate every term needed in Equation (1). The resulting approximations have provable convergence guarantees (Proposition 3.1).

Listing 1: Example PyTorch implementation of SLORR-Hoyer.

```
class LowRankRegularizer(torch.autograd.
    Function):
  @staticmethod
  def forward(ctx, W, eps, steps=6):
      frob = torch.linalg.norm(W, ord="fro")
    + eps
      UV = polar_express(W, steps=steps)
      nuc = torch.sum(W * UV)
      reg = nuc / frob
      ctx.save_for_backward(W, UV, frob, nuc
  , reg)
      return reg ** 2

  @staticmethod
  def backward(ctx, grad_output):
      W, UV, frob, nuc, reg = ctx.
    saved_tensors
      G = (UV / frob) - (nuc / (frob**3)) *
    W
      return grad_output * 2 * reg * G, None
    , None
```

Computing the polar factor. Notice that we do not need to compute the factors of $\boldsymbol{U}\boldsymbol{V}^\top$ separately; we only need their product[2], which is generally known as the generalized polar factor (we use "polar factor" for short) of $\boldsymbol{W}$. It can be computed efficiently on GPUs using iterative methods (e.g., (Higham, 1994); a similar approach is used in the Muon optimizer (Jordan et al., 2024) to obtain polar factors of gradients). We use Polar Express (Amsel et al., 2026) to approximate $\boldsymbol{U}\boldsymbol{V}^\top$ efficiently on GPUs.

Polar Express depends on several hyperparameters, including the number of iterations T, most of which we set to their recommended values. It first normalizes the matrix so that its largest singular value is at most 1. This is achieved by dividing by the Frobenius norm, i.e., it uses $\boldsymbol{W}/(\|\boldsymbol{W}\|_F + \varepsilon)$[3]. Moreover, it also needs a lower bound estimate of the smallest nonzero singular value of the normalized matrix, ℓ, which we set to 10^{-3} following their recommendations. We provide a more detailed discussion of our usage of the Polar Express in Appendix D.1.

Computing the nuclear and Frobenius norms. After we have (an approximation of) $\boldsymbol{U}\boldsymbol{V}^\top$, we can compute an estimate of the nuclear norm with elementwise multiplications and a summation. In particular, it holds that (for any exact $\boldsymbol{U}\boldsymbol{V}^\top$)

$$\text{sum}(\boldsymbol{W} \odot \boldsymbol{U}\boldsymbol{V}^\top) = \text{tr}(\boldsymbol{W}^\top\boldsymbol{U}\boldsymbol{V}^\top) = \text{tr}(\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^\top\boldsymbol{U}\boldsymbol{V}^\top) = \text{tr}(\boldsymbol{\Sigma}) = \|\boldsymbol{W}\|_*,$$

where $\odot$ denotes elementwise multiplication, and the intermediate steps follow from the cyclic property of the trace operator. The Frobenius norm is simply $\|\boldsymbol{W}\|_F = \sqrt{\sum_{i,j}\boldsymbol{W}_{i,j}^2}$.

Combining everything. We now have all the elements needed to compute approximate values and approximate backpropagation rules for both variants, which can be implemented, for example, as a custom PyTorch function. Forward and backward computations of SLORR-Hoyer are summarized in Listing 1[4]; SLORR-Nuc admits a similar implementation. Note that we do not backpropagate through

[2]We use the singular vectors corresponding to the nonzero singular values of the matrix, whose product is unique (Li & Sun, 2002).

[3]In Listing 1, normalization is performed inside the Polar Express function.

[4]As shown in Listing 1, in practice, we stabilize the forward and backward expressions separately by substituting $\|\boldsymbol{W}\|_F \mapsto \|\boldsymbol{W}\|_F + \varepsilon$.

the Polar Express iterations. Instead, the approximate polar factor is used in place of the $\boldsymbol{U}\boldsymbol{V}^\top$ term that appears in the regularizer values and in the gradient rules in Equation (1). During training, we compute the task loss and regularization loss, weighted by the regularization strength λ, and use gradient-based optimization as usual[5]. In the case of Adam (Kingma & Ba, 2017), one can also use a decoupled low-rank regularization term, similar to Q3R (Ghosh et al., 2025); see Algorithm 1.

Computational cost. In terms of matrix multiplication FLOPs (the main cost), each iteration of Polar Express (with our exact settings) with a matrix of shape (M, N) uses $4P^2Q + 2P^3$ FLOPs, where $P = \min\{M, N\}$ and $Q = \max\{M, N\}$, and we generally use 6 iterations. For convolutional layers, we reshape kernels into matrices (details in Appendix E.1); in which case we get $M = C_o$ and $N = C_i K_w K_h$, denoting the output and input channels, as well as the width and height of the convolutional kernel, respectively. We discuss practical SLORR overhead measures in Section 4.1 and Section 4.2, where we find that it generally remains below or around 8% in our ImageNet-scale settings and 3% in LLM pretraining at 135M, 560M, and 1B scales, whereas SVD remains prohibitively expensive.

Approximation guarantees. We remark that the computed forward and backward passes of both variants are approximate in practice. Polar Express has worst-case approximation guarantees, which can be used to provide guarantees on the values and gradients of both SLORR variants (Proposition 3.1). We provide further discussion of this result and its proof in Appendix C.

Proposition 3.1 (Approximation guarantees)**.** *Let $\boldsymbol{W} \neq \boldsymbol{0}$ have thin SVD $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, and assume the nonzero singular values of a normalized version of $\boldsymbol{W}$ lie in $[\ell, 1]$. Let $\widehat{\boldsymbol{P}}$ denote the Polar Express approximation of the polar factor $\boldsymbol{U}\boldsymbol{V}^\top$ after T steps, and define $\delta = \left|1 - \ell^2\right|^{(q+1)^T}$, where $d = 2q + 1$ is the Polar Express polynomial degree. Additionally, let $\hat{n} := \operatorname{tr}(\boldsymbol{W}^\top \widehat{\boldsymbol{P}})$ denote the approximate nuclear norm estimate, and let $n := \|\boldsymbol{W}\|_*$ and $f := \|\boldsymbol{W}\|_F$. Moreover, let $\boldsymbol{R}$ denote the exact SLORR-Hoyer gradient and $\widehat{\boldsymbol{R}}$ its approximation. Then*

$$\frac{|\hat{n} - n|}{n} \leq \delta, \quad \|\widehat{\boldsymbol{P}} - \boldsymbol{U}\boldsymbol{V}^\top\|_2 \leq \delta \qquad \textit{(SLORR-Nuc)},$$

$$\frac{|\hat{n}^2/f^2 - n^2/f^2|}{n^2/f^2} \leq \delta(2+\delta), \quad \|\boldsymbol{R} - \widehat{\boldsymbol{R}}\|_2 \leq 2\delta(2+\delta)\left(\frac{n}{f^2} + \frac{n^2}{f^4}\|\boldsymbol{W}\|_2\right) \qquad \textit{(SLORR-Hoyer)}.$$

In particular, errors vanish as $T \to \infty$.

Practical remarks. The singular value range in Proposition 3.1 is a theoretical assumption on the normalized matrix, inherited from the Polar Express analysis in Amsel et al. (2026). In practice, Polar Express treats ℓ as an approximate lower bound parameter; the authors report that inaccurate guesses are typically not severe. Therefore, the displayed worst case bound should be interpreted as applying to normalized nonzero singular values lying in $[\ell, 1]$, while our choice $\ell = 10^{-3}$ follows their recommended value. We also emphasize that this is a worst case bound in an idealized setting. In practice, a smaller number of iterations works well even when the bound is not tight, and floating point error and implementation details may introduce additional approximation error. We follow the practical implementation choices of Amsel et al. (2026), including their recommended parameter settings and numerical stabilizations. Further discussion is provided in Appendix D.1 and Appendix C.

3.1 Understanding the Hoyer Regularizer

Our analysis is performed in an exact regularizer-only gradient descent setting. In this section, $\eta > 0$ denotes the step size of the regularizer-only update, with the training-time regularization strength absorbed into it. Additional comments and proofs appear in Appendix B. Let $\boldsymbol{W}$ be a weight matrix with thin SVD $\boldsymbol{W} = \boldsymbol{U}\,\mathrm{diag}\,(\boldsymbol{\sigma})\boldsymbol{V}^\top$. After a regularizer-only gradient step, denote the updated matrix by $\boldsymbol{W}^+$, with thin SVD $\boldsymbol{W}^+ = \boldsymbol{U}^+\,\mathrm{diag}(\boldsymbol{\sigma}^+)(\boldsymbol{V}^+)^\top$. We assume ordered singular values in decreasing order ($\boldsymbol{\sigma}_1 \geq \cdots \geq \boldsymbol{\sigma}_r > 0$) and $\boldsymbol{W} \neq \boldsymbol{0}$. We first remark that SLORR-Hoyer is scale-invariant (in the same way that the original version for vector sparsity is (Hoyer, 2004; Yang et al., 2020b)).[6] Intuitively, this means it aims to redistribute the spectral energy rather than grow or

[5]The autograd-based implementation is generally easy to integrate into existing codebases. An alternative memory-efficient implementation (used in our vision experiments) is shown in Appendix D.3.

[6]This is easy to see: $\|\alpha\boldsymbol{W}\|_*^2/\|\alpha\boldsymbol{W}\|_F^2 = \alpha^2\|\boldsymbol{W}\|_*^2/\alpha^2\|\boldsymbol{W}\|_F^2 = \|\boldsymbol{W}\|_*^2/\|\boldsymbol{W}\|_F^2$.

shrink it. Note, however, that in a practical discretized optimization setting, the overall norm of the weight tends to increase (note that the gradient is orthogonal to the weight matrix):

$$\|\boldsymbol{W} - \eta\nabla\mathcal{L}_{\text{hoyer}}(\boldsymbol{W})\|_F^2 = \|\boldsymbol{W}\|_F^2 - 2\eta\left\langle \boldsymbol{W}, \nabla\mathcal{L}_{\text{hoyer}}(\boldsymbol{W})\right\rangle_F + \eta^2\|\nabla\mathcal{L}_{\text{hoyer}}(\boldsymbol{W})\|_F^2 \geq \|\boldsymbol{W}\|_F^2.$$

In preliminary experiments, we tried a norm-correction step that did not seem to help in practice.

Proposition 3.2 (SLORR-Hoyer shrinkage and growth of singular values)**.** *For a sufficiently small regularizer-only gradient step, it holds that*

$$\boldsymbol{\sigma}_i^+ \diamond \boldsymbol{\sigma}_i \iff \boldsymbol{\sigma}_i \diamond \frac{\|\boldsymbol{W}\|_F^2}{\|\boldsymbol{W}\|_*}, \quad \textit{for} \diamond \in \{<, >, =\}.$$

Proposition 3.2 suggests that SLORR-Hoyer enlarges "large" singular values and shrinks "small" ones, where "large" and "small" are determined by the dynamic quantity $\|\boldsymbol{W}\|_F^2/\|\boldsymbol{W}\|_*$, which varies as the weights evolve. Intuitively, this concentrates the spectrum onto the already large singular values. In turn, under certain assumptions, it induces low-rank compressibility (Proposition 3.3).

Proposition 3.3 (Sufficiently small steps of SLORR-Hoyer induce compressibility)**.** *Let* $\tau = \|\boldsymbol{W}\|_F^2/\|\boldsymbol{W}\|_*$*. For a sufficiently small regularizer-only gradient step, if* $\boldsymbol{\sigma}_k > \tau$*, then*

$$\frac{\|\boldsymbol{U}^+_{:,:k}\operatorname{diag}(\boldsymbol{\sigma}^+_{:k})\boldsymbol{V}^{+\top}_{:,:k} - \boldsymbol{W}^+\|_F^2}{\|\boldsymbol{W}^+\|_F^2} < \frac{\|\boldsymbol{U}_{:,:k}\operatorname{diag}(\boldsymbol{\sigma}_{:k})\boldsymbol{V}^{\top}_{:,:k} - \boldsymbol{W}\|_F^2}{\|\boldsymbol{W}\|_F^2}.$$

4 Experiments

We evaluate the effectiveness of SLORR in inducing compressibility during training. Our experiments cover two main domains: image classification on ImageNet-1K (Deng et al., 2009), including both pretraining and continued training, and LLM pretraining. Our general pipeline consists of first training regularized and unregularized models, then compressing them and assessing their retained performance. Regularization and compression are applied to all linear and convolutional layers, except for the first and last layers. SLORR-Hoyer-D denotes our decoupled variant (Appendix D.2). In total, we perform more than 240 runs across a broad array of settings.

4.1 Image Classification Experiments

We first evaluate SLORR on ImageNet-1K classification. We cover two settings, namely pretraining and continued training, across four architectures. We compare against recent baselines, which have largely focused on the image classification domain. Further details regarding baselines and settings are given in Appendix E.

Baselines and comparisons. We compare SLORR against Q3R (Ghosh et al., 2025) and LoRITa (Alkhouri et al., 2024). These baselines are used because: (1) they are the most recent relevant methods, to the best of our knowledge, and (2) they allow us to compare different objectives, whereas factorization-based approaches (Yang et al., 2020a) generally have similar objectives to ours but optimize them after altering the architecture.

For Q3R, we apply regularization through a decoupled gradient update, following the original experiments. We jointly sweep the regularization strength and target rank, with the refresh interval fixed to 5. Our initial experiments favored stronger regularization than the originally recommended range, $\lambda \in [0.001, 0.01]$, so we extend the search beyond that range. We additionally examine longer refresh intervals, which reduce overhead and can remain effective in continued training, in Appendix F; overhead measurements are reported in Table 10.

For LoRITa, we fix the expansion factor to $N = 2$ and vary the regularization strength. We follow the method described in the original paper while using a more efficient implementation and adapting its initialization to support pretrained checkpoints, as required for continued training. For SLORR, we use 6 Polar Express iterations and sweep the regularization strength λ. We include an unregularized baseline trained with the same recipe in every comparison.

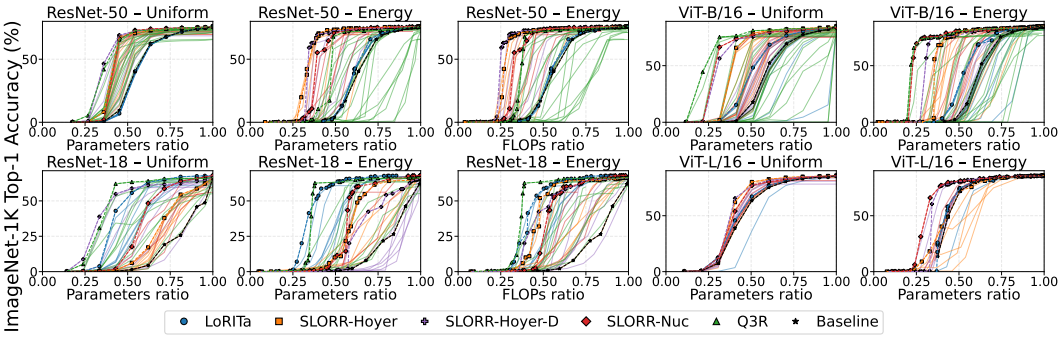

Figure 1: **Accuracy–compression curves of regularized ResNet-50, ViT-B/16, ViT-L/16, and ResNet-18 models on ImageNet-1K.** Factorization is performed using the energy and uniform rank-selection criteria (see Section 4 for details). Faint curves denote different hyperparameter selections. Highlighted curves indicate Pareto frontiers. For uniform compression and ViT energy compression, parameter and FLOPs ratios are equal up to a small shift, so separate panels are omitted.

Hyperparameter selection follows a best-effort manual search. With $N = 2$ fixed, LoRITa and SLORR each have one regularization hyperparameter, whereas Q3R requires jointly selecting regularization strength and target rank. Implementation details and the search procedure are provided in Appendix E, and all included configurations are reported in Appendix G. Separate ablations examine the Q3R refresh interval (Appendix F) and the number of Polar Express iterations (Appendix D.1).

Compression. After training, we perform compression using two different approaches. Uniform compression retains approximately the same fraction of parameters and FLOPs in each compressed layer. For energy-based truncation, we select the minimum rank for each layer such that the relative squared Frobenius approximation error of its weight matrix is at most $1 - \rho$, where ρ is the energy-retention threshold.

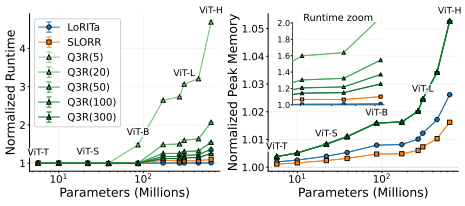

Figure 2: **Regularization overhead scaling across ViT sizes on B200 GPUs.** Runtime (left) and peak memory (right) are normalized to unregularized training. For Q3R, parentheses indicate the SVD refresh interval. The inset zooms runtime near ViT-H. Full overhead measurements for our classification experiments are reported in Table 10.

ImageNet-1K training details. For continued training, we train pretrained ImageNet-1K (Deng et al., 2009) ViT-B/16 and ViT-L/16 (Dosovitskiy et al., 2021) and ResNet-50 (He et al., 2016) checkpoints for 30 epochs using AdamW (Loshchilov & Hutter, 2019) with standard settings. We use batch size 1024 for ViT-B/16 and ResNet-50 and 896 for ViT-L/16[7]. We use learning rates of 10^{-5} and 5×10^{-5} for ViT-B/16, 10^{-5} for ViT-L/16, and 5×10^{-5} for ResNet-50, chosen to preserve unregularized pretrained performance. Q3R performs poorly on ViT-B/16 at the smaller rate, motivating the additional larger-rate comparison. We compare against Q3R and LoRITa on ViT-B/16 and ResNet-50, and against LoRITa and the unregularized baseline on ViT-L/16. For pretraining, we train ResNet-18 from scratch for 110 epochs with otherwise similar settings to ResNet-50 and learning rate 10^{-3}, comparing against Q3R and LoRITa.

Compression results. We report results for the different settings in Figure 1. The accuracy–compression Pareto frontier is highlighted, with fainter lines showing trajectories for each hyperparameter configuration. In general, for all methods, there appears to be an accuracy/compressibility tradeoff. Under our settings, we find that at least one variant of SLORR generally performs close to Q3R, sometimes surpassing it. LoRITa is relatively weak in most of our continued training settings, though it is more effective in the ResNet-18 pretraining setting. Note that, in some circumstances, some runs of different methods underperform the baseline in terms of compressibility (in particular, Q3R seems not to perform well on the ViT-B/16 runs with smaller learning rates; see Table 7 in

[7]LoRITa runs out of memory on ViT-L/16 at batch size 1024, so we use 896 for all methods for a matched comparison.

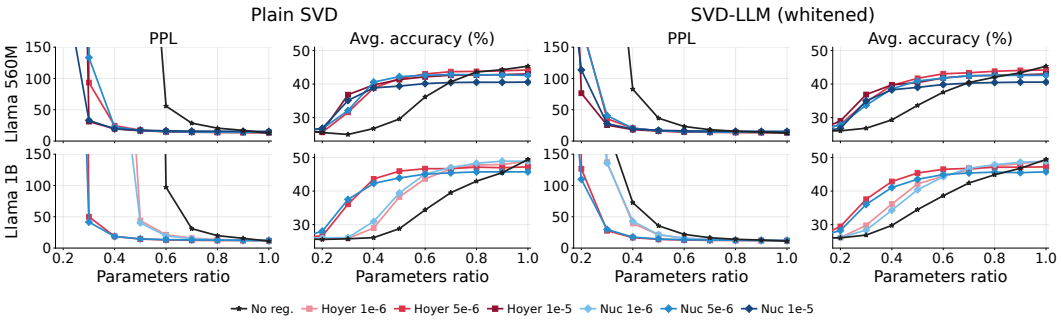

Figure 4: **LLM compression results with SLORR-Hoyer and SLORR-Nuc regularization.** Color intensity indicates the regularization strength λ used during pretraining. $\lambda = 0$ corresponds to the unregularized baseline. The perplexity axis is limited to 150 for visibility; uncapped results are shown in Figure 12. Downstream accuracy denotes the unweighted mean across six tasks. Per-task results are shown in Figures 8 to 11.

Appendix G). Interestingly, each exact setting appears to favor different methods, including different SLORR variants, suggesting that there might not be a single one-size-fits-all objective. We additionally assess sensitivity to the training seed in Appendix G.1, finding that for the configuration tried the seed doesn't significantly affect results.

Classification model training overhead. We measure training overhead under the settings used in our classification experiments; the complete measurements, including absolute timings, peak memory, and variability across repeated measurements, are reported in Table 10, with the protocol in Appendix H. SLORR maintains less than 8% average runtime overhead across the evaluated variants and the smallest memory overhead among the compared methods.

Separately, Figure 2 provides an illustrative scaling benchmark from ViT-T to ViT-H on B200 GPUs, rather than additional accuracy–compression experiments. It shows that maintaining low Q3R overhead at larger scales requires longer SVD refresh intervals.

4.2 LLM Pretraining

We now study regularized LLM pretraining using SLORR. In particular, we train standard Llama-like (Touvron et al., 2023) transformer models on FineWeb-Edu (Penedo et al., 2024), using the AdamW optimizer and distributed data parallelism (DDP), following standard practice. Exact training details are provided in Appendix I. Our main models are trained on approximately 20 training tokens per parameter, following the compute-optimal scaling rule of Hoffmann et al. (2022). We train 135M (where we compare against LoRITa and Q3R), 560M and 1B models using standard unregularized optimization, as well as both SLORR-Hoyer and SLORR-Nuc for different regularization strengths λ. We find that, across scales, SLORR incurs less than 3% training overhead.

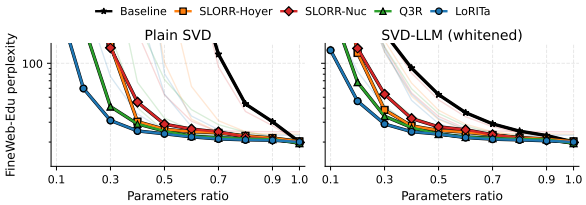

Figure 3: Comparison against LoRITa and Q3R.

After training, we compress models using both plain SVD and the recent SVD-LLM (Wang et al., 2025b), which uses activation statistics to compute a whitening transformation before compressing each layer, generally enabling stronger compression. Plain SVD is known to be impractical for LLMs, as degradation is significant even when removing a small fraction of parameters (Wang et al., 2025b). After compression, we evaluate the models. Across scales, we report validation perplexity on a held-out FineWeb-Edu set of approximately 2M tokens. For the 560M and 1B models, we additionally report average zero-shot downstream task accuracy using the `lm-evaluation-harness` library (Gao et al., 2024). Details are provided in Appendix I.

Compression results. Parameter ratios are computed over transformer blocks. At 135M, SLORR is competitive with Q3R and LoRITa (Figure 3); at 560M and 1B, it improves compression under both

plain SVD and SVD-LLM in perplexity and downstream accuracy (Figure 4). Gains are particularly pronounced for plain SVD, where unregularized models are very brittle.

Longer training. As noted above, our main models are trained to the compute-optimal token budget (Hoffmann et al., 2022). We also train 135M models with four and eight times the compute-optimal token budget. Compressibility appears to decrease with the training budget for both regularized and unregularized models, but regularized models retain their substantial advantage. These results are shown in Appendix J.

SLORR works well with other optimizers. We also assess the effectiveness of SLORR on other strong optimizers. In particular, aside from our main experiments with AdamW, we also evaluate SLORR with SOAP (Vyas et al., 2025), Muon (Jordan et al., 2024) and Kron (Li, 2018; Pooladzandi & Li, 2024)[8]. The exact results are shown in Appendix K. We find that SLORR produces more compressible models across all optimizers.

Visualizing the effect of SLORR on weight spectra. SLORR promotes low rank with the objective of making models amenable to factorization. Naturally, this is directly reflected in the visual properties of weight spectra, where stronger regularization induces faster singular value decay. We include plots for all Llama 135M layers in the appendix (Figure 13), where the spectral concentration effect is apparent. Perhaps surprisingly, there are visible spectral differences in the embedding matrix, but not in the final linear layer. Since the embedding layer was not regularized, this indicates that modifying the spectra of regularized layers can affect other parts of the model.

5 Discussion

Limitations. Weight low-rank regularization may not be effective in every setting. In exploratory experiments, we found that fine-tuning LLMs with low-rank regularization while preserving general capabilities was challenging, suggesting that the data and training setup are important, as one might expect. Additionally, there is generally a tradeoff between final accuracy and compressibility, and excessive regularization can cause training instabilities. Moreover, although our experiments are significantly more extensive than those of prior work (more than 240 runs in total across a broad array of settings), the performance of regularizers can vary depending on the training setting and hyperparameters. Similarly, we did not explore every SLORR variant in every setting. Therefore, although we provide a broad best-effort comparison, our results should not be interpreted as a definitive ranking of all methods. Finally, our theory, while informative, relies on specific assumptions that may not hold in every practical case.

Future work. One promising direction is to combine regularization with complementary techniques, such as periodic truncation, similar to Xu et al. (2020), or multiple rounds of retraining and compression. Regularization strength schedules, as in Dolatabadi et al. (2026), would likely also yield better results. Moreover, the need for orthogonalization in optimization has sparked research into more effective polar function approximations, e.g., (Zhang et al., 2026), which could be applied to SLORR. Finally, we believe our work could motivate broader research beyond the scope of this paper. Although the main application of this work is to directly regularize weights, the versatility of SLORR may open new avenues in settings where one cannot rely on architectural modifications or the validity of cached quantities. For example, low-rank regularization of activations via the nuclear norm has been shown to enhance generalization (Shi et al., 2024); while that work uses SVDs, an efficient version could potentially be computed using our approach. Similarly, although nontrivial, it would be interesting to extend low-rank regularization to encourage activations to lie in a subspace, enabling better use of activation-aware factorization methods. More broadly, an accessible way to regularize rank may motivate work on broader applications of rank regularization beyond those studied here, including settings where encouraging a higher rank may be beneficial.

Conclusion. We have introduced SLORR, a framework for directly regularizing weight matrices to be low rank. The idea is simple yet highly versatile: it does not require SVDs, remains efficient, operates directly on the matrix being regularized, and is stateless. Our experiments on ImageNet-1K image classification and LLM pretraining show that this regularizer is effective at producing more compressible models while maintaining training efficiency, enabling scalable low-rank regularization.

[8] In particular, the version at https://github.com/evanatyourservice/kron_torch

AI USE STATEMENT

We used generative AI tools to assist with literature search (all cited works were ultimately read and chosen by the authors), discussion of research ideas, suggestions for proof steps, and the presentation of technical arguments. Regarding these, their role was limited to the discussion of standard arguments, feedback, and refinement of our ideas. The main ideas were those of the authors, but the discussions with LLMs were generally helpful. We also used these tools to assist with method implementation and debugging, and for general feedback on experiments and parts of the analysis. Beyond that, they were used for general manuscript polishing and formatting. We take responsibility for the final content of this work, including the text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have taken several steps to ensure the reproducibility of our work. First, beyond the low-rank regularization, our training procedure is standard, and we provide the relevant hyperparameters and settings in the appendix. We further provide our codebase in the supplementary materials, with code to reproduce the different experiments presented here. For theoretical proofs, all details can be found in the appendix.

REFERENCES

Ismail Alkhouri, Xitong Zhang, and Rongrong Wang. Structure-preserving network compression via low-rank induced training through linear layers composition. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=1KCrVMJoJ9.

Jose M. Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 856–867, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Noah Amsel, David Persson, Christopher Musco, and Robert M. Gower. The polar express: Optimal matrix sign methods and their application to the muon algorithm. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=yRtgZ1K8hO.

Matan Ben Noach and Yoav Goldberg. Compressing pre-trained language models by matrix decomposition. In Kam-Fai Wong, Kevin Knight, and Hua Wu (eds.), *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pp. 884–889, Suzhou, China, December 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.aacl-main.88. URL https://aclanthology.org/2020.aacl-main.88/.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

Patrick Chen, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. Drone: Data-aware low-rank compression for large nlp models. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 29321–29334. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/f56de5ef149cf0aedcc8f4797031e229-Paper.pdf.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Frank H. Clarke. *Optimization and Nonsmooth Analysis*. Society for Industrial and Applied Mathematics, 1990. doi: 10.1137/1.9781611971309. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611971309.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pp. 1269–1277, Cambridge, MA, USA, 2014. MIT Press.

Hadi Mohaghegh Dolatabadi, Thalaiyasingam Ajanthan, Sameera Ramasinghe, Chamin P Hewa Koneputugodage, Shamane Siriwardhana, Violetta Shevchenko, Karol Pajak, James Snewin, Gil Avraham, and Alexander Long. Numuon: Nuclear-norm-constrained muon for compressible llm training, 2026. URL https://arxiv.org/abs/2603.03597.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

M. Fazel, H. Hindi, and S. Boyd. Rank minimization and applications in system theory. In *Proceedings of the 2004 American Control Conference*, volume 4, pp. 3273–3278 vol.4, 2004. doi: 10.23919/ACC.2004.1384521.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Ipsita Ghosh, Ethan Nguyen, and Christian Kümmerle. Q3r: Quadratic reweighted rank regularizer for effective low-rank training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=ZtzWvNKOCr.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations - 4th Edition*. Johns Hopkins University Press, Philadelphia, PA, 2013. doi: 10.1137/1.9781421407944. URL https://epubs.siam.org/doi/abs/10.1137/1.9781421407944.

David González-Martínez. Balf: Budgeted activation-aware low-rank factorization for fine-tuning-free model compression, 2025. URL https://arxiv.org/abs/2509.25136.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour, 2018. URL https://arxiv.org/abs/1706.02677.

Andi Han, Jiaxiang Li, Wei Huang, Mingyi Hong, Akiko Takeda, Pratik Jawanpuria, and Bamdev Mishra. SLTrain: a sparse plus low rank approach for parameter and memory efficient pretraining. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=MXze4H7opg.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Nicholas J. Higham. The matrix sign decomposition and its relation to the polar decomposition. *Linear Algebra and its Applications*, 212-213:3–20, 1994. ISSN 0024-3795. doi: https://doi.org/10.1016/0024-3795(94)90393-X. URL https://www.sciencedirect.com/science/article/pii/002437959490393X.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre.

Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Patrik O. Hoyer. Non-negative matrix factorization with sparseness constraints. *J. Mach. Learn. Res.*, 5:1457–1469, December 2004. ISSN 1532-4435.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014. doi: http://dx.doi.org/10.5244/C.28.88.

Ajay Kumar Jaiswal, Yifan Wang, Lu Yin, Shiwei Liu, Runjin Chen, Jiawei Zhao, Ananth Grama, Yuandong Tian, and Zhangyang Wang. From low rank gradient subspace stabilization to low-rank weights: Observations, theories, and applications. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=fsf7LhbYdf.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Siddhartha Rao Kamalakara, Acyr Locatelli, Bharat Venkitesh, Jimmy Ba, Yarin Gal, and Aidan N. Gomez. Exploring low rank training of deep neural networks, 2022. URL https://arxiv.org/abs/2209.13569.

Mikhail Khodak, Neil A. Tenenholtz, Lester Mackey, and Nicolo Fusi. Initialization and regularization of factorized neural layers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=KTlJT1nof6d.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Markus Kliegl, Siddharth Goyal, Kexin Zhao, Kavya Srinet, and Mohammad Shoeybi. Trace norm regularization and faster inference for embedded speech recognition rnns, 2018. URL https://arxiv.org/abs/1710.09026.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86:2278–2324, 1998. URL https://api.semanticscholar.org/CorpusID:14542261.

Jiaxi Li, Lu Yin, Li Shen, Jinjin Xu, Liwu Xu, Tianjin Huang, Wenwu Wang, Shiwei Liu, and Xilu Wang. Lost: Low-rank and sparse pre-training for large language models, 2025a. URL https://arxiv.org/abs/2508.02668.

Naiqi Li, Yuqiu Xie, Peiyuan Liu, Tao Dai, Yong Jiang, and Shu-Tao Xia. Efficient differentiable approximation of generalized low-rank regularization. In James Kwok (ed.), *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pp. 5616–5624. International Joint Conferences on Artificial Intelligence Organization, 8 2025b. doi: 10.24963/ijcai.2025/625. URL https://doi.org/10.24963/ijcai.2025/625. Main Track.

Wen Li and Weiwei Sun. Perturbation bounds of unitary and subunitary polar factors. *SIAM Journal on Matrix Analysis and Applications*, 23(4):1183–1193, 2002. doi: 10.1137/S0895479801394623. URL https://doi.org/10.1137/S0895479801394623.

Xi-Lin Li. Preconditioned stochastic gradient descent. *IEEE Transactions on Neural Networks and Learning Systems*, 29(5):1454–1466, May 2018. ISSN 2162-2388. doi: 10.1109/tnnls.2017.2672978. URL http://dx.doi.org/10.1109/TNNLS.2017.2672978.

Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. Relora: High-rank training through low-rank updates, 2023. URL https://arxiv.org/abs/2307.05695.

Lucas Liebenwein, Alaa Maalouf, Dan Feldman, and Daniela Rus. Compressing neural networks: Towards determining the optimal layer-wise decomposition. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=BvJkwMhyInm.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 olmo 2 furious, 2024. URL https://arxiv.org/abs/2501.00656.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Omead Pooladzandi and Xi-Lin Li. Curvature-informed sgd via general purpose lie-group preconditioners, 2024. URL https://arxiv.org/abs/2402.04553.

Haoran Qin, Shansita Sharma, Ali Abbasi, Chayne Thrash, and Soheil Kolouri. Low-rank prehab: Preparing neural networks for svd compression, 2025. URL https://arxiv.org/abs/2512.01980.

Wang Qinsi, Jinghan Ke, Masayoshi Tomizuka, Kurt Keutzer, and Chenfeng Xu. Dobi-SVD: Differentiable SVD for LLM compression and some new perspectives. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=kws76i5XB8.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

Zhenmei Shi, Yifei Ming, Ying Fan, Frederic Sala, and Yingyu Liang. Domain generalization via nuclear norm regularization. In *Conference on Parsimony and Learning (Proceedings Track)*, 2024. URL https://openreview.net/forum?id=hJd66ZzXEZ.

TorchVision maintainers and contributors. TorchVision: PyTorch's Computer Vision library, November 2016. URL https://github.com/pytorch/vision.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham M. Kakade. SOAP: Improving and stabilizing shampoo using adam for language modeling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=IDxZhXrpNf.

Xin Wang, Samiul Alam, Zhongwei Wan, Hui Shen, and Mi Zhang. SVD-LLM v2: Optimizing singular value truncation for large language model compression. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4287–4296, Albuquerque, New Mexico, April 2025a. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.217. URL https://aclanthology.org/2025.naacl-long.217/.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. SVD-LLM: Truncation-aware singular value decomposition for large language model compression. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=LNYIUouhdt.

Xiuying Wei, Skander Moalla, Razvan Pascanu, and Caglar Gulcehre. Building on efficient foundations: Effective training of LLMs with structured feedforward layers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=WxLVYZbIew.

Ross Wightman. PyTorch Image Models, 2019. URL https://github.com/huggingface/pytorch-image-models.

Yuhui Xu, Yuxi Li, Shuai Zhang, Wei Wen, Botao Wang, Yingyong Qi, Yiran Chen, Weiyao Lin, and Hongkai Xiong. TRP: trained rank pruning for efficient deep neural networks. *CoRR*, abs/2004.14566, 2020. URL https://arxiv.org/abs/2004.14566.

Huanrui Yang, Minxue Tang, Wei Wen, Feng Yan, Daniel Hu, Ang Li, Hai Li, and Yiran Chen. Learning low-rank deep neural networks via singular vector orthogonality regularization and singular value sparsification. *CoRR*, abs/2004.09031, 2020a. URL https://arxiv.org/abs/2004.09031.

Huanrui Yang, Wei Wen, and Hai Li. Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=rylBK34FDS.

Zhihang Yuan, Yuzhang Shang, Yue Song, Dawei Yang, Qiang Wu, Yan Yan, and Guangyu Sun. Asvd: Activation-aware singular value decomposition for compressing large language models, 2025. URL https://arxiv.org/abs/2312.05821.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Jack Zhang, Noah Amsel, Berlin Chen, and Tri Dao. Gram newton-schulz, 2026. URL https://dao-ailab.github.io/blog/2026/gram-newton-schulz/.

Max Zimmer, Christoph Spiegel, and Sebastian Pokutta. Compression-aware training of neural networks using frank-wolfe, 2024. URL https://arxiv.org/abs/2205.11921.

A BACKPROPAGATING THROUGH THE REGULARIZERS

As noted before, the SLORR regularizers are not differentiable everywhere. In practice, one can informally use "natural" backpropagation rules. We note that, for SLORR-Hoyer, the computed rule simply corresponds to backpropagating through the whole expression (and, when backpropagating through the nuclear norm, using the generalized polar factor). These correspond to minimum-norm choices in the generalized gradient of the corresponding functions, in the sense of Clarke (Clarke, 1990, Chapter 2), which we denote by

$$\partial f(\boldsymbol{W}).$$

Throughout this section, when $\boldsymbol{U}$ and $\boldsymbol{V}$ appear, they refer to the thin SVD of $\boldsymbol{W}$, as in the rest of the paper. We also assume $\boldsymbol{W} \neq \boldsymbol{0}$, so that $\|\boldsymbol{W}\|_F > 0$.

First, note that both $\|\boldsymbol{W}\|_*$ and $\|\boldsymbol{W}\|_F$ are locally Lipschitz. Moreover, $\|\boldsymbol{W}\|_F^2$ is continuously differentiable, with

$$\nabla \|\boldsymbol{W}\|_F^2 = 2\boldsymbol{W}.$$

By (Clarke, 1990, Proposition 2.2.4), this implies

$$\partial \|\boldsymbol{W}\|_F^2 = \{2\boldsymbol{W}\}.$$

We use Clarke's notion of regularity (Clarke, 1990, Definition 2.3.4). Continuously differentiable functions and convex functions are regular (Clarke, 1990, Proposition 2.3.6). Therefore, $\|\boldsymbol{W}\|_F^2$, $\|\boldsymbol{W}\|_*$, and $\|\boldsymbol{W}\|_*^2$ are regular.

The nuclear norm is convex. Hence, by (Clarke, 1990, Proposition 2.2.7), its Clarke generalized gradient coincides with the subdifferential from convex analysis.

Generalized gradient of SLORR-Nuc.

The SLORR-Nuc regularizer is the nuclear norm,

$$\mathcal{L}_{\text{nuc}}(\boldsymbol{W}) = \|\boldsymbol{W}\|_*.$$

Let

$$\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$$

be the thin SVD of $\boldsymbol{W}$. The generalized gradient is

$$\partial \|\boldsymbol{W}\|_* = \left\{ \boldsymbol{U}\boldsymbol{V}^\top + \boldsymbol{Q} : \boldsymbol{U}^\top \boldsymbol{Q} = \boldsymbol{0}, \boldsymbol{Q}\boldsymbol{V} = \boldsymbol{0}, \|\boldsymbol{Q}\|_2 \leq 1 \right\}.$$

Since $\boldsymbol{Q}$ is Frobenius-orthogonal to $\boldsymbol{U}\boldsymbol{V}^\top$, the minimum-norm element is obtained by taking $\boldsymbol{Q} = \boldsymbol{0}$. Thus, the backpropagation choice is

$$\nabla \mathcal{L}_{\text{nuc}}(\boldsymbol{W}) = \boldsymbol{U}\boldsymbol{V}^\top \in \partial \mathcal{L}_{\text{nuc}}(\boldsymbol{W}).$$

Generalized gradient of SLORR-Hoyer.

The Hoyer-type SLORR regularizer is not convex. It can be written as

$$\mathcal{L}_{\text{hoyer}}(\boldsymbol{W}) = \frac{a(\boldsymbol{W})}{b(\boldsymbol{W})},$$

where

$$a(\boldsymbol{W}) = \|\boldsymbol{W}\|_*^2, \qquad b(\boldsymbol{W}) = \|\boldsymbol{W}\|_F^2.$$

Since $\boldsymbol{W} \neq \boldsymbol{0}$, we have $a(\boldsymbol{W}) > 0$ and $b(\boldsymbol{W}) > 0$.

Moreover, since $\|\boldsymbol{W}\|_*^2$ is convex (and locally Lipschitz), its generalized gradient corresponds to the convex subdifferential (Clarke, 1990, Proposition 2.2.7):

$$\partial a(\boldsymbol{W}) = 2\|\boldsymbol{W}\|_* \partial \|\boldsymbol{W}\|_*.$$

Also,

$$\nabla b(\boldsymbol{W}) = 2\boldsymbol{W}.$$

By the quotient rule for generalized gradients (Clarke, 1990, Proposition 2.3.14),

$$\partial \left[\frac{a(\boldsymbol{W})}{b(\boldsymbol{W})}\right] = \frac{b(\boldsymbol{W})\partial a(\boldsymbol{W}) - a(\boldsymbol{W})\nabla b(\boldsymbol{W})}{b(\boldsymbol{W})^2}.$$

Note that equality holds because a and $-b$ are regular, and because $a(\boldsymbol{W}) > 0$ and $b(\boldsymbol{W}) > 0$.

Substituting, we obtain

$$\partial \mathcal{L}_{\text{hoyer}}(\boldsymbol{W}) = \left\{ 2\frac{\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^2}\left(\boldsymbol{U}\boldsymbol{V}^\top + \boldsymbol{Q}\right) - 2\frac{\|\boldsymbol{W}\|_*^2}{\|\boldsymbol{W}\|_F^4}\boldsymbol{W} : \boldsymbol{U}^\top\boldsymbol{Q} = \boldsymbol{0},\, \boldsymbol{Q}\boldsymbol{V} = \boldsymbol{0},\, \|\boldsymbol{Q}\|_2 \le 1 \right\}.$$

The $\boldsymbol{Q}$ term is again orthogonal to the remaining terms. Hence, the minimum-norm element is again obtained by taking $\boldsymbol{Q} = \boldsymbol{0}$. Thus, the backpropagation choice is

$$\nabla \mathcal{L}_{\text{hoyer}}(\boldsymbol{W}) = 2\frac{\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^2}\boldsymbol{U}\boldsymbol{V}^\top - 2\frac{\|\boldsymbol{W}\|_*^2}{\|\boldsymbol{W}\|_F^4}\boldsymbol{W} \in \partial \mathcal{L}_{\text{hoyer}}(\boldsymbol{W}).$$

Equivalently,

$$\nabla \mathcal{L}_{\text{hoyer}}(\boldsymbol{W}) = 2\frac{\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F}\left(\frac{1}{\|\boldsymbol{W}\|_F}\boldsymbol{U}\boldsymbol{V}^\top - \frac{\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^3}\boldsymbol{W}\right).$$

We then use these expressions when backpropagating through the regularizers. We remark that, intuitively, these correspond to backpropagating individually through the composition of functions in PyTorch, and setting the gradient of the nuclear norm to the generalized polar factor. We use this specific choice because it is the one that is generally (approximately) computed by polynomial approximations such as Polar Express.

B Analysis of SLORR-Hoyer

Here, we provide proofs and additional discussion of the results shown in Section 3.1. We restate the results for the convenience of the reader.

We remark, following Section 3.1, that our analysis is performed in a regularizer-only gradient descent setting. $\eta > 0$ denotes the step size of the regularizer-only update, with the training-time regularization strength absorbed into it. We let $\boldsymbol{W} = \boldsymbol{U}\,\mathrm{diag}\,(\boldsymbol{\sigma})\boldsymbol{V}^\top$ be a weight matrix and its thin SVD, with singular values $\boldsymbol{\sigma}$. Additionally, let $\boldsymbol{W}^+$ be the weight matrix after a gradient step from $\boldsymbol{W}$, and define $\boldsymbol{\sigma}^+$, $\boldsymbol{U}^+$, and $\boldsymbol{V}^+$ as its thin SVD. We assume ordered singular values in decreasing order ($\boldsymbol{\sigma}_1 \ge \cdots \ge \boldsymbol{\sigma}_r > 0$) and $\boldsymbol{W} \neq \boldsymbol{0}$.

Proposition 3.2 (SLORR-Hoyer shrinkage and growth of singular values)**.** *For a sufficiently small regularizer-only gradient step, it holds that*

$$\boldsymbol{\sigma}_i^+ \diamond \boldsymbol{\sigma}_i \iff \boldsymbol{\sigma}_i \diamond \frac{\|\boldsymbol{W}\|_F^2}{\|\boldsymbol{W}\|_*}, \quad \textit{for} \diamond \in \{<, >, =\}.$$

Proof. Define $C = 2\|\boldsymbol{W}\|_*/\|\boldsymbol{W}\|_F$ for brevity. First, write our gradient update as

$$\begin{aligned}
\boldsymbol{W}^+ &= \boldsymbol{W} - \eta\nabla\mathcal{L}_{\text{hoyer}}(\boldsymbol{W}) \\
&= \boldsymbol{W} - \eta\left[\frac{C}{\|\boldsymbol{W}\|_F}\boldsymbol{U}\boldsymbol{V}^\top - \frac{C\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^3}\boldsymbol{W}\right] \\
&= \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top - \eta\frac{C}{\|\boldsymbol{W}\|_F}\boldsymbol{U}\boldsymbol{V}^\top + \eta\frac{C\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^3}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top \\
&= \boldsymbol{U}\left[\boldsymbol{\Sigma} - \eta\frac{C}{\|\boldsymbol{W}\|_F}\boldsymbol{I} + \eta\frac{C\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^3}\boldsymbol{\Sigma}\right]\boldsymbol{V}^\top.
\end{aligned}$$

Since the matrix in the "sandwich" is diagonal (as it is a sum of two diagonal matrices), we can conclude that, for small enough updates such that no inner term becomes negative,

$$\boldsymbol{\sigma}_i^+ = \boldsymbol{\sigma}_i - \eta\frac{C}{\|\boldsymbol{W}\|_F} + \eta\frac{C\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^3}\boldsymbol{\sigma}_i.$$

Moreover, note that the order of singular values is preserved.

Now,

$$\begin{aligned}
\boldsymbol{\sigma}_i^+ \diamond \boldsymbol{\sigma}_i &\iff \boldsymbol{\sigma}_i - \eta\frac{C}{\|\boldsymbol{W}\|_F} + \eta\frac{C\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^3}\boldsymbol{\sigma}_i \diamond \boldsymbol{\sigma}_i \\
&\iff -\eta\frac{C}{\|\boldsymbol{W}\|_F} + \eta\frac{C\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^3}\boldsymbol{\sigma}_i \diamond 0 \\
&\iff -\frac{C}{\|\boldsymbol{W}\|_F} + \frac{C\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^3}\boldsymbol{\sigma}_i \diamond 0 \quad (\eta > 0) \\
&\iff -\frac{1}{\|\boldsymbol{W}\|_F} + \frac{\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^3}\boldsymbol{\sigma}_i \diamond 0 \quad (C > 0) \\
&\iff -\|\boldsymbol{W}\|_F^2 + \|\boldsymbol{W}\|_*\boldsymbol{\sigma}_i \diamond 0 \\
&\iff \|\boldsymbol{W}\|_*\boldsymbol{\sigma}_i \diamond \|\boldsymbol{W}\|_F^2 \\
&\iff \boldsymbol{\sigma}_i \diamond \frac{\|\boldsymbol{W}\|_F^2}{\|\boldsymbol{W}\|_*},
\end{aligned}$$

for $\diamond \in \{<, >, =\}$.

This concludes the proof.

□

Proposition 3.3 (Sufficiently small steps of SLORR-Hoyer induce compressibility)**.** *Let* $\tau = \|\boldsymbol{W}\|_F^2/\|\boldsymbol{W}\|_*$. *For a sufficiently small regularizer-only gradient step, if* $\boldsymbol{\sigma}_k > \tau$, *then*

$$\frac{\|\boldsymbol{U}^+_{:,:k}\operatorname{diag}(\boldsymbol{\sigma}^+_{:k})\boldsymbol{V}^{+\top}_{:,:k} - \boldsymbol{W}^+\|_F^2}{\|\boldsymbol{W}^+\|_F^2} < \frac{\|\boldsymbol{U}_{:,:k}\operatorname{diag}(\boldsymbol{\sigma}_{:k})\boldsymbol{V}^{\top}_{:,:k} - \boldsymbol{W}\|_F^2}{\|\boldsymbol{W}\|_F^2}.$$

Proof. For brevity, define

$$\varepsilon = \eta\frac{2\|\boldsymbol{W}\|_*^2}{\|\boldsymbol{W}\|_F^4}.$$

Note that (see the proof of Proposition 3.2)

$$\boldsymbol{\sigma}_i^+ = \boldsymbol{\sigma}_i - \eta\frac{2\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^2} + \eta\frac{2\|\boldsymbol{W}\|_*^2}{\|\boldsymbol{W}\|_F^4}\boldsymbol{\sigma}_i.$$

Then

$$\eta\frac{2\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^2} = \eta\frac{2\|\boldsymbol{W}\|_*^2}{\|\boldsymbol{W}\|_F^4}\cdot\frac{\|\boldsymbol{W}\|_F^2}{\|\boldsymbol{W}\|_*} = \varepsilon\tau.$$

Therefore,

$$\begin{aligned}
\boldsymbol{\sigma}_i^+ &= \boldsymbol{\sigma}_i - \eta\frac{2\|\boldsymbol{W}\|_*}{\|\boldsymbol{W}\|_F^2} + \eta\frac{2\|\boldsymbol{W}\|_*^2}{\|\boldsymbol{W}\|_F^4}\boldsymbol{\sigma}_i \\
&= \boldsymbol{\sigma}_i - \varepsilon\tau + \varepsilon\boldsymbol{\sigma}_i \\
&= \boldsymbol{\sigma}_i + \varepsilon(\boldsymbol{\sigma}_i - \tau).
\end{aligned}$$

For convenience, now define

$$A^+ = \sum_{i\le k} (\boldsymbol{\sigma}_i + \varepsilon(\boldsymbol{\sigma}_i - \tau))^2 = \sum_{i\le k} \left[\boldsymbol{\sigma}_i^2 + 2\varepsilon\boldsymbol{\sigma}_i(\boldsymbol{\sigma}_i - \tau) + \varepsilon^2(\boldsymbol{\sigma}_i - \tau)^2\right]$$
$$= \underbrace{\sum_{i\le k} \boldsymbol{\sigma}_i^2}_{A} + \sum_{i\le k} \left[2\varepsilon\boldsymbol{\sigma}_i(\boldsymbol{\sigma}_i - \tau) + \varepsilon^2(\boldsymbol{\sigma}_i - \tau)^2\right] = A + \sum_{i\le k} \left[2\varepsilon\boldsymbol{\sigma}_i(\boldsymbol{\sigma}_i - \tau) + \varepsilon^2(\boldsymbol{\sigma}_i - \tau)^2\right],$$

$$B^+ = \sum_{i>k} (\boldsymbol{\sigma}_i + \varepsilon(\boldsymbol{\sigma}_i - \tau))^2 = \sum_{i>k} \left[\boldsymbol{\sigma}_i^2 + 2\varepsilon\boldsymbol{\sigma}_i(\boldsymbol{\sigma}_i - \tau) + \varepsilon^2(\boldsymbol{\sigma}_i - \tau)^2\right]$$
$$= \underbrace{\sum_{i>k} \boldsymbol{\sigma}_i^2}_{B} + \sum_{i>k} \left[2\varepsilon\boldsymbol{\sigma}_i(\boldsymbol{\sigma}_i - \tau) + \varepsilon^2(\boldsymbol{\sigma}_i - \tau)^2\right] = B + \sum_{i>k} \left[2\varepsilon\boldsymbol{\sigma}_i(\boldsymbol{\sigma}_i - \tau) + \varepsilon^2(\boldsymbol{\sigma}_i - \tau)^2\right].$$

The A term. Note that for the A term,

$$(\boldsymbol{\sigma}_i - \tau) > 0,$$

as required in the statement of this result. Hence, $A^+ - A = \varepsilon a_1 + \varepsilon^2 a_2$ (as every other term is nonnegative), where $a_1, a_2 > 0$.

The $A + B$ term. We will show that, for sufficiently small ε, the $A + B$ increases less than A. First note

$$\sum_i 2\varepsilon\boldsymbol{\sigma}_i(\boldsymbol{\sigma}_i - \tau) = 2\varepsilon\left(\sum_i \boldsymbol{\sigma}_i^2 - \tau\sum_i \boldsymbol{\sigma}_i\right) = 2\varepsilon\left(\sum_i \boldsymbol{\sigma}_i^2 - \frac{\|\boldsymbol{W}\|_F^2}{\|\boldsymbol{W}\|_*}\sum_i \boldsymbol{\sigma}_i\right) = 0.$$

Now,

$$A^+ + B^+ - (A + B) = \sum_i \left[2\varepsilon\boldsymbol{\sigma}_i(\boldsymbol{\sigma}_i - \tau) + \varepsilon^2(\boldsymbol{\sigma}_i - \tau)^2\right] = \sum_i \left[\varepsilon^2(\boldsymbol{\sigma}_i - \tau)^2\right] = b\varepsilon^2,$$

with $b > 0$.

As the A term has a strictly positive linear term in ε, while the total increment is only quadratic, taking ε small enough gives

$$A^+ - A > (A^+ + B^+) - (A + B).$$

Also note that $A^+ - A > 0$. Hence,

$$\frac{A^+}{A^+ + B^+} > \frac{A}{A + B}.$$

Final step. Now, separate the sums

$$\frac{\sum_{i\le k}(\boldsymbol{\sigma}_i^+)^2}{\sum_i(\boldsymbol{\sigma}_i^+)^2} = \frac{\sum_{i\le k}(\boldsymbol{\sigma}_i^+)^2}{\sum_{i\le k}(\boldsymbol{\sigma}_i^+)^2 + \sum_{i>k}(\boldsymbol{\sigma}_i^+)^2} = \frac{A^+}{A^+ + B^+},$$

$$\frac{\sum_{i\le k}\boldsymbol{\sigma}_i^2}{\sum_i \boldsymbol{\sigma}_i^2} = \frac{\sum_{i\le k}\boldsymbol{\sigma}_i^2}{\sum_{i\le k}\boldsymbol{\sigma}_i^2 + \sum_{i>k}\boldsymbol{\sigma}_i^2} = \frac{A}{A + B}.$$

Finally, it follows that

$$1 - \frac{\|\boldsymbol{U}^+_{:,:k}\operatorname{diag}(\boldsymbol{\sigma}^+_{:k})\boldsymbol{V}^{+\top}_{:,:k} - \boldsymbol{W}^+\|_F^2}{\|\boldsymbol{W}^+\|_F^2} = \frac{\sum_{i\le k}(\boldsymbol{\sigma}_i^+)^2}{\sum_i(\boldsymbol{\sigma}_i^+)^2} > \frac{\sum_{i\le k}\boldsymbol{\sigma}_i^2}{\sum_i \boldsymbol{\sigma}_i^2}$$
$$= 1 - \frac{\|\boldsymbol{U}_{:,:k}\operatorname{diag}(\boldsymbol{\sigma}_{:k})\boldsymbol{V}^{\top}_{:,:k} - \boldsymbol{W}\|_F^2}{\|\boldsymbol{W}\|_F^2}.$$

The result follows from subtracting 1 and swapping sides. □

C Approximation Error Guarantees

We restate the result for the convenience of the reader.

Proposition 3.1 (Approximation guarantees). *Let $\boldsymbol{W} \neq \boldsymbol{0}$ have thin SVD $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, and assume the nonzero singular values of a normalized version of $\boldsymbol{W}$ lie in $[\ell, 1]$. Let $\widehat{\boldsymbol{P}}$ denote the Polar Express approximation of the polar factor $\boldsymbol{U}\boldsymbol{V}^\top$ after T steps, and define $\delta = \left|1-\ell^2\right|^{(q+1)^T}$, where $d = 2q+1$ is the Polar Express polynomial degree. Additionally, let $\hat{n} := \operatorname{tr}(\boldsymbol{W}^\top \widehat{\boldsymbol{P}})$ denote the approximate nuclear norm estimate, and let $n := \|\boldsymbol{W}\|_*$ and $f := \|\boldsymbol{W}\|_F$. Moreover, let $\boldsymbol{R}$ denote the exact SLORR-Hoyer gradient and $\widehat{\boldsymbol{R}}$ its approximation. Then*

$$\frac{|\hat{n}-n|}{n} \leq \delta, \quad \|\widehat{\boldsymbol{P}} - \boldsymbol{U}\boldsymbol{V}^\top\|_2 \leq \delta \qquad \textit{(SLORR-Nuc)},$$

$$\frac{|\hat{n}^2/f^2 - n^2/f^2|}{n^2/f^2} \leq \delta(2+\delta), \quad \|\boldsymbol{R} - \widehat{\boldsymbol{R}}\|_2 \leq 2\delta(2+\delta)\left(\frac{n}{f^2} + \frac{n^2}{f^4}\|\boldsymbol{W}\|_2\right) \qquad \textit{(SLORR-Hoyer)}.$$

In particular, errors vanish as $T \to \infty$.

Proof. Given $\boldsymbol{W}$, by assumption, a normalized version $\boldsymbol{M}$ satisfies $\boldsymbol{\sigma}(\boldsymbol{M}) \subseteq [\ell, 1]$ (where only the nonzero singular values are taken into account). By Theorem 3.3 of Amsel et al. (2026),

$$\left\|\widehat{\boldsymbol{P}} - \boldsymbol{U}\boldsymbol{V}^\top\right\|_2 \leq \left|1-\ell^2\right|^{(q+1)^T},$$

where T is the number of Polar Express steps (6 in our main experiments) and $d = 2q+1$ is the degree of the Polar Express polynomial (5 in our experiments). We can now extend it to the approximation of the nuclear norm. In particular, fix the PE parameters and let $\delta = \left|1-\ell^2\right|^{(q+1)^T}$.

First, we have

$$\begin{aligned}
|\hat{n} - n| &= \left|\operatorname{tr}\left(\boldsymbol{W}^\top(\widehat{\boldsymbol{P}} - \boldsymbol{U}\boldsymbol{V}^\top)\right)\right| \\
&\leq \|\boldsymbol{W}\|_* \left\|\widehat{\boldsymbol{P}} - \boldsymbol{U}\boldsymbol{V}^\top\right\|_2 \\
&\leq \delta \|\boldsymbol{W}\|_*.
\end{aligned}$$

Therefore,

$$\frac{|\hat{n}-n|}{n} \leq \delta.$$

Similarly, we can bound the error in the SLORR-Hoyer value. Using

$$|\hat{n}| \leq n + |\hat{n} - n| \leq (1+\delta)n,$$

we obtain

$$\begin{aligned}
\left|\hat{n}^2 - n^2\right| &= |\hat{n} - n|\ |\hat{n} + n| \\
&\leq |\hat{n} - n|\ (|\hat{n}| + n) \\
&\leq \delta n\,((1+\delta)n + n) \\
&= n^2\delta(2+\delta).
\end{aligned}$$

Therefore, we get (dividing by n^2)

$$\frac{\left|\hat{n}^2 - n^2\right|}{n^2} \leq \delta(2+\delta),$$

or equivalently,

$$\frac{\left|\hat{n}^2/f^2 - n^2/f^2\right|}{n^2/f^2} \leq \delta(2+\delta).$$

Now, we can bound the error on the SLORR-Hoyer gradient. Recall that (for the exact polar factor):

$$\boldsymbol{R} = 2\frac{n}{f}\left(\frac{1}{f}\boldsymbol{U}\boldsymbol{V}^\top - \frac{n}{f^3}\boldsymbol{W}\right),$$

and note that its Polar Express approximation is

$$\widehat{\boldsymbol{R}} = 2\frac{\hat{n}}{f}\left(\frac{1}{f}\widehat{\boldsymbol{P}} - \frac{\hat{n}}{f^3}\boldsymbol{W}\right).$$

We have

$$\begin{aligned}
\left\|\boldsymbol{R} - \widehat{\boldsymbol{R}}\right\|_2 &= \left\|2\frac{n}{f}\left(\frac{1}{f}\boldsymbol{U}\boldsymbol{V}^\top - \frac{n}{f^3}\boldsymbol{W}\right) - 2\frac{\hat{n}}{f}\left(\frac{1}{f}\widehat{\boldsymbol{P}} - \frac{\hat{n}}{f^3}\boldsymbol{W}\right)\right\|_2 \\
&= \left\|\frac{2n}{f^2}\boldsymbol{U}\boldsymbol{V}^\top - \frac{2n^2}{f^4}\boldsymbol{W} - \frac{2\hat{n}}{f^2}\widehat{\boldsymbol{P}} + \frac{2\hat{n}^2}{f^4}\boldsymbol{W}\right\|_2 \\
&\le \left\|\frac{2n}{f^2}\boldsymbol{U}\boldsymbol{V}^\top - \frac{2\hat{n}}{f^2}\boldsymbol{U}\boldsymbol{V}^\top\right\|_2 + \left\|\frac{2\hat{n}}{f^2}\boldsymbol{U}\boldsymbol{V}^\top - \frac{2\hat{n}}{f^2}\widehat{\boldsymbol{P}}\right\|_2 + \left\|-\frac{2n^2}{f^4}\boldsymbol{W} + \frac{2\hat{n}^2}{f^4}\boldsymbol{W}\right\|_2 \\
&\le \delta\frac{2n}{f^2} + \frac{2\,|\hat{n}|}{f^2}\delta + \frac{2\delta(2+\delta)n^2}{f^4}\|\boldsymbol{W}\|_2 \le \frac{2\delta(n+|\hat{n}|)}{f^2} + \frac{2\delta(2+\delta)n^2}{f^4}\|\boldsymbol{W}\|_2 .
\end{aligned}$$

Now, to get rid of $\hat{n}$, note that

$$|\hat{n}| \le n + |\hat{n} - n| \le (1+\delta)n.$$

Hence,

$$\begin{aligned}
\left\|\boldsymbol{R} - \widehat{\boldsymbol{R}}\right\|_2 &\le \frac{2\delta(n+(1+\delta)n)}{f^2} + \frac{2\delta(2+\delta)n^2}{f^4}\|\boldsymbol{W}\|_2 \\
&= \frac{2\delta(2+\delta)n}{f^2} + \frac{2\delta(2+\delta)n^2}{f^4}\|\boldsymbol{W}\|_2 \\
&= 2\delta(2+\delta)\left(\frac{n}{f^2} + \frac{n^2}{f^4}\|\boldsymbol{W}\|_2\right).
\end{aligned}$$

In particular, as $T \to \infty$, the error tends to 0, as expected. □

Remark about zero singular values. The Polar Express guarantee we use only concerns the nonzero singular subspace, corresponding to the generalized polar factor (i.e., using the thin SVD) used throughout this paper. Exact zero singular values are therefore excluded from the result. In exact arithmetic, the Polar Express updates do not create components in this zero singular subspace, since the scalar polynomial maps used in their iteration satisfy $p(0) = 0$. Thus, zero singular values remain zero under the idealized iteration, and hence our approximation guarantee concerns only the nonzero singular values.

Remark about normalization and the lower bound parameter. The singular value range $[\ell, 1]$ is inherited from the analysis in Amsel et al. (2026). For a nonzero matrix $\boldsymbol{W}$, we apply Polar Express to the normalized matrix

$$\widetilde{\boldsymbol{W}} = \frac{\boldsymbol{W}}{\|\boldsymbol{W}\|_F + \varepsilon}.$$

This positive rescaling leaves the polar factor unchanged and ensures that the largest singular value of $\widetilde{\boldsymbol{W}}$ is at most one. However, it does not by itself guarantee that every nonzero singular value is at least ℓ. The parameter ℓ should therefore be interpreted as the lower bound parameter used by Polar Express. If the actual smallest nonzero normalized singular value is below ℓ, the displayed worst-case bound does not apply literally to that singular direction. We note that Amsel et al. (2026) mention that inaccurate lower bound guesses are typically not severe in practice.

Remark about practical considerations. The guarantee above is a worst-case bound for the idealized Polar Express iteration and should not be read as a tight estimate of the implemented approximation error. In our experiments, we use a small number of Polar Express iterations together with the practical adaptations of Amsel et al. (2026). These choices slightly deviate from the idealized theorem, and floating point arithmetic can introduce additional error. Empirically, this setting works well in our experiments; see Appendix D.1 for an ablation and implementation details.

D General Experimental and Implementation Details

The code used to produce our experimental results is available at https://anonymous.4open.science/r/SLORR-E4F8. It includes the implementation of our method and baselines, as well as the scripts, hyperparameters, and configurations used for our experiments. Our code for both the vision and LLM experiments is based on the PyTorch framework (Paszke et al., 2019).

D.1 On our Use of Polar Express

Polar Express has several tunable knobs. The first is the lower bound on singular values, ℓ, along with the degree and number of iterations. Following Implementation 1 in Amsel et al. (2026), we set $\ell = 10^{-3}$ and the degree to $d = 5$. They note that choosing ℓ too small can delay convergence for smaller singular values, but that inaccurate guesses are typically not severe in practice. We found these values to work well in our experiments. We use `bfloat16`, as recommended.

An important hyperparameter of Polar Express is the number of iterations. The authors recommend 5 or 6 iterations. Throughout our experiments, we used 6 Polar Express iterations, which add minimal overhead. In early experiments, we found that this quantity worked well. Here, we explicitly ablate the number of iterations. Following our ViT-B/16 experiments, we fix $\lambda = 0.002$ and repeat our uniform compressibility experiments using runs with $\{1, 2, 4, 6, 8, 10\}$ Polar Express iterations. Results are presented in Table 2. In this setting, after 6 iterations, results across compression ratios change little, suggesting diminishing returns from additional Polar Express iterations in this setting.

Table 2: **Effect of Polar Express iterations (ViT-B/16).** We repeat the main ViT-B/16 SLORR-Hoyer experiment with $\lambda = 0.002$ while varying only the number of Polar Express iterations. Columns denote retained parameter ratios, and entries are ImageNet-1K top-1 accuracy.

Steps	21%	31%	41%	51%	60%	70%	80%	90%	Full
1	0.1	0.1	0.6	14.4	49.0	67.9	77.1	81.0	84.9
2	0.1	0.1	0.2	0.3	1.6	32.3	63.7	72.6	83.4
4	0.1	0.1	0.1	0.2	3.8	18.6	82.6	83.3	83.7
6	0.1	0.1	6.6	66.2	79.5	81.1	81.5	81.7	81.7
8	0.1	0.1	5.2	67.7	79.5	80.9	81.3	81.5	81.6
10	0.1	0.1	7.0	68.9	79.4	80.9	81.2	81.4	81.5

D.2 Implementation Details for the Decoupled Variant

Algorithm 1 details how the gradient of the SLORR regularizer can be applied in a decoupled form. This follows the rationale of AdamW (Loshchilov & Hutter, 2019), where weight decay is applied based on the pre-update weights. SLORR-Hoyer-D follows this approach, but note that this approach can also be applied to SLORR-Nuc. Note that, in practice, the regularizer value (not only the gradient) can also be computed (at the same time as the gradient) and, for example, logged.

Algorithm 1 Decoupled SLORR update

Require: Weight set $\mathcal{W}$, regularized weight set $\mathcal{W}_{\text{reg}} \subseteq \mathcal{W}$, learning rate η, regularization strength λ
Require: Task gradients $\nabla_{\mathcal{W}} \mathcal{L}_{\text{task}}$
for all $\boldsymbol{W} \in \mathcal{W} \setminus \mathcal{W}_{\text{reg}}$ **do**
  if $\nabla_{\boldsymbol{W}} \mathcal{L}_{\text{task}}$ exists **then**
    Apply AdamW update to $\boldsymbol{W}$ using $\nabla_{\boldsymbol{W}} \mathcal{L}_{\text{task}}$
  end if
end for
for all $\boldsymbol{W} \in \mathcal{W}_{\text{reg}}$ **do**
  $\widehat{\boldsymbol{G}}^{\text{pre}}_{\text{SLORR}} \leftarrow \widehat{\boldsymbol{G}}_{\text{SLORR}}(\boldsymbol{W})$
  if $\nabla_{\boldsymbol{W}} \mathcal{L}_{\text{task}}$ exists **then**
    Apply AdamW update to $\boldsymbol{W}$ using $\nabla_{\boldsymbol{W}} \mathcal{L}_{\text{task}}$
  end if
  $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta\lambda \widehat{\boldsymbol{G}}^{\text{pre}}_{\text{SLORR}}$ ▷ uses the pre-update SLORR gradient
end for

D.3 Implementation Details for the Memory-Efficient Variant

Algorithm 2 details a more memory-efficient approach for PyTorch environments. When implemented as a loss function (i.e., not in decoupled form, as detailed in Appendix D.2), using an autograd function is easy to implement. However, if one computes each layer-wise loss, sums them, and then backpropagates, PyTorch keeps intermediate values for all layers. This can consume additional memory proportional to the total size of the regularized layers. Again, this can be applied to both SLORR-Nuc and SLORR-Hoyer. Similar to Appendix D.2, the regularizer value can also be computed at the same time, if needed.

Algorithm 2 Iterative memory-efficient SLORR implementation

Require: Regularized weight set $\mathcal{W}_{\text{reg}}$, regularization strength λ
for all $\boldsymbol{W} \in \mathcal{W}_{\text{reg}}$ **do**
  Compute $\widehat{\boldsymbol{G}}_{\text{SLORR}}(\boldsymbol{W})$ from the current value of $\boldsymbol{W}$
  if $\boldsymbol{W}$.grad exists **then**
    $\boldsymbol{W}.\text{grad} \leftarrow \boldsymbol{W}.\text{grad} + \lambda \widehat{\boldsymbol{G}}_{\text{SLORR}}(\boldsymbol{W})$
  else
    $\boldsymbol{W}.\text{grad} \leftarrow \lambda \widehat{\boldsymbol{G}}_{\text{SLORR}}(\boldsymbol{W})$
  end if
end for

E VISION EXPERIMENTAL AND IMPLEMENTATION DETAILS

For the vision experiments, we use standard ImageNet training pipelines built on the model implementations provided by `timm` (Wightman, 2019).

E.1 INTERPRETING CONVOLUTIONAL LAYERS

For factorization and regularization, we interpret each convolutional kernel as a matrix by reshaping

$$\mathbf{W} \in \mathbb{R}^{C_o \times C_i \times H_k \times W_k}$$

as $\mathrm{reshape}(\mathbf{W}; (C_o, -1))$. A low-rank factorization under this interpretation corresponds to replacing the original convolution with two sequential convolutions. This is the standard channel-wise decomposition used in prior low-rank factorization work, including LoRITa.

E.2 IMPLEMENTATION DETAILS FOR BASELINES

E.2.1 LORITA

We implement LoRITa largely as described in Alkhouri et al. (2024), with a few adaptations. We fix $N = 2$, where N denotes the number of factorized layers used to replace each original layer. Since we initialize from pretrained checkpoints, we choose the factors so that the initial network is functionally equivalent to the original model. Specifically, for a weight matrix $\boldsymbol{W}$, we compute its thin SVD and initialize

$$\boldsymbol{W}_0 = \boldsymbol{U}\boldsymbol{\Sigma}^{1/2}, \qquad \boldsymbol{W}_1 = \boldsymbol{\Sigma}^{1/2}\boldsymbol{V}^\top.$$

This corresponds to the "spectral initialization" of Khodak et al. (2021) in the case where all singular values are preserved.

For ResNet-18 pretraining, we first initialize each corresponding dense layer using the same random initializer as in the dense baseline. We then compute its factorization and initialize the LoRITa factors using the spectral initialization above. This matches the initial function of the dense baseline while using LoRITa's factorized parameterization, allowing us to control for initialization effects.

Following Ghosh et al. (2025) and Alkhouri et al. (2024), we implement the LoRITa regularizer through the optimizer's weight decay term. The value reported for LoRITa in the result tables is this coefficient. As we use AdamW, it is applied in a decoupled way. Parameters that are not replaced by LoRITa factors use the default weight decay for the corresponding experiment, while normalization parameters and biases are not decayed. This keeps the LoRITa penalty separate from the ordinary optimizer weight decay used for the rest of the model.

For convolutional layers, we follow the construction described in Alkhouri et al. (2024), which coincides with the general interpretation of convolutional layers discussed in Appendix E.1.

We make one implementation modification to improve memory efficiency, which allows us to run LoRITa on larger models. In the original sequential implementation[9], LoRITa replaces a layer by a sequence of factorized layers. With $N = 2$, this introduces an additional intermediate activation and can substantially increase memory use at realistic batch sizes. Even when memory is sufficient, the sequential implementation can remain computationally expensive; see the LoRITa authors' comments at `https://openreview.net/forum?id=1KCrVMJoJ9`.

We describe the modification for a linear layer without bias; the convolutional case follows the same matrix interpretation as in Appendix E.1. Let the original layer be parameterized by $\boldsymbol{W} \in \mathbb{R}^{I \times O}$, and let $R = \min\{I, O\}$. LoRITa replaces this layer with two trainable matrices,

$$\boldsymbol{W}_0 \in \mathbb{R}^{I \times R}, \qquad \boldsymbol{W}_1 \in \mathbb{R}^{R \times O}.$$

For an input $\boldsymbol{X} \in \mathbb{R}^{B \times I}$, with batch and token dimensions folded into a single effective batch dimension, the original computation is

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W},$$

[9] `https://github.com/XitongSystem/LoRITa`

while the sequential LoRITa implementation computes

$$\boldsymbol{Y} = (\boldsymbol{X}\boldsymbol{W}_0)\boldsymbol{W}_1.$$

Instead, in each forward pass we first merge the factors and then apply the merged matrix:

$$\boldsymbol{Y} = \boldsymbol{X}(\boldsymbol{W}_0\boldsymbol{W}_1).$$

This avoids storing the intermediate activation $\boldsymbol{X}\boldsymbol{W}_0$ and reduces LoRITa's overhead to effectively zero in our experiments. The merged computation defines the same function of $\boldsymbol{W}_0$ and $\boldsymbol{W}_1$ as the sequential implementation, so the forward pass and gradients are equivalent up to floating point differences.

Moreover, the original LoRITa implementation uses $R = I$. We use $R = \min\{I, O\}$, which is the natural full-rank dimension for the matrix factorization above and enables the checkpoint-preserving initialization described here. The main theoretical results supporting LoRITa still apply (see Section 4.3 of Alkhouri et al. (2024)).

We restrict LoRITa to $N = 2$. Extending the initialization to $N > 2$ would require choosing how to split the singular values across more than two factors, introducing an additional experimental choice. Since the LoRITa paper reports only marginal ImageNet ResNet gains from $N = 3$, including for ResNet-18, we keep $N = 2$ throughout and leave larger N as a limitation of our study.

E.2.2 Q3R

Following the authors' recommended setup, we use their AdamQ3R optimizer. AdamQ3R combines Adam-style optimization with the Q3R regularizer, applying the regularization effect in a decoupled form motivated by AdamW (Loshchilov & Hutter, 2019). We copied the optimizer and regularizer implementation from the authors' official codebase[10] and left the linear-layer path unchanged, except that we added support for decoupled weight decay to match our AdamW baselines.

We use a refresh interval of 5 for all main experiments, unless noted otherwise, following the setting used by the Q3R authors in their main experiments. We also found that larger, cheaper refresh intervals can work well in some settings; see Appendix F.

The original implementation does not support convolutional layers. We extend it to convolutions by interpreting convolutional layers as matrices, as described in Appendix E.1. This extension only inserts the necessary reshape operations and leaves other parts of the code untouched.

When we refer to Q3R's "target rank", we mean the value passed to the AdamQ3R optimizer. This value can be a ratio between 0 and 1 rather than an absolute rank.

E.3 Training Details and Hyperparameters

We first describe the general training setup and then list specific hyperparameters for each model.

Regularization and factorization are applied to layers in their native parameterization. In particular, the `timm` ViT implementations use fused QKV layers, and we regularize and factorize these fused matrices without splitting them. This follows LoRITa's ViT implementation. Q3R's experiments split QKV layers, but we note this is an experimental choice rather than a mechanism inherent to the method (we remark that, in any case, this choice must be consistent across all methods). Hence, for simplicity, we use the default fused QKV implementation. We did not explore split QKV layers.

We keep non-regularizer hyperparameters fixed across methods whenever possible. For LoRITa, the factor weight decay is part of the method and is varied as its regularization strength. For Q3R, our AdamQ3R implementation supports the same decoupled weight decay used by the AdamW baselines and SLORR runs. Across methods, ordinary weight decay is applied only to weight matrices; normalization parameters and biases are not decayed.

For continued training experiments, we choose learning rates so that an unregularized baseline largely preserves the accuracy of the pretrained checkpoint. On ViT-B/16, Q3R tended to perform poorly with the smaller learning rate, so we tested a larger one. LoRITa also improved slightly in that setting, so we include both learning rates. We did not explore this higher-learning-rate setting for other

[10] https://github.com/ThatE10/Q3R

models or for ResNet-18 pretraining (for ResNet-18, the learning rate was again chosen based on unregularized performance).

We remark that even though our sweeps are significantly more extensive than those of prior work, differences in training configurations might benefit different techniques. Accordingly, our comparisons should be interpreted as a broad best-effort study under mostly matched training recipes rather than as an exhaustive sweep, as mentioned in Section 5.

All runs use `bfloat16` automatic mixed precision (weights are stored in `float32`).

Checkpoints. For runs starting from checkpoints, we use the default ones on `timm`, except for ResNet-50 for which we use the TorchVision (TorchVision maintainers and contributors, 2016) checkpoint, which is the most common one.

Hyperparameters for ViT-B/16. For ViT-B/16, we use learning rates of 1×10^{-5} and 5×10^{-5}, a batch size of 1024, a weight decay of 0.1, and a cosine schedule with 1 epoch of linear warmup (Goyal et al., 2018; Loshchilov & Hutter, 2017) with an initial factor of 10^{-3}, followed by decay to 0.1 times the original learning rate. We use standard ViT-B/16 augmentation through the `timm` library. We detail the augmentation parameters in Table 3.

Hyperparameters for ViT-L/16. We use the same setup as for ViT-B/16, but with a batch size of 896. With a batch size of 1024, LoRITa produced out-of-memory errors, even with our optimizations. We therefore use a batch size of 896 for all methods in this setting to ensure comparable conditions. We only used a learning rate of 1×10^{-5}.

Hyperparameters for ResNet-50. For ResNet-50, we use a learning rate of 5×10^{-5}. We use the same scheduler as in the ViT experiments, a batch size of 1024, and a weight decay of 0.01. We apply standard data augmentation, consisting of random resizing and cropping, followed by horizontal flipping with probability 0.5, and normalization using the standard ImageNet-1K mean and standard deviation.

Hyperparameters for ResNet-18 (training from scratch). We use a learning rate of 10^{-3} and train for 110 epochs. The rest is the same as for ResNet-50.

All vision experiments use A100, H100, or B200 GPUs depending on availability.

Method hyperparameter selection. Method-specific hyperparameters were selected through exploratory runs and manual coarse-to-fine tuning. SLORR and LoRITa each have a single main regularization coefficient, whereas Q3R has two interacting hyperparameters, excluding the refresh interval, making exhaustive grid search impractical. For Q3R, we found that values larger than the authors' recommended range of $[0.001, 0.01]$ worked substantially better in our settings.

For all methods, stronger regularization generally improved compressibility but could reduce uncompressed accuracy or cause training collapse. Conversely, too little regularization had little effect on compressibility. Runs were discarded only when they collapsed or produced unusable uncompressed accuracy, not based on post-compression performance. All included configurations are listed in Appendix G.

E.4 Factorization Details

Our factorization implementation is adapted from the BALF (González-Martínez, 2025) repository. For every model, we factorize all layers except the first and last, typically the classifier. As mentioned in the main text, we perform experiments with both uniform truncation and energy truncation. We describe both below; our implementation is included in our code.

Uniform truncation. Given a target retention ratio $c \in (0, 1]$, with $c = 1$ denoting no compression, each eligible layer is truncated to approximately retain a fraction c of its original parameters. Under our matrix interpretation, the same rank choice also gives approximately the same retained FLOPs fraction for both linear and convolutional layers. Thus, uniform truncation applies the same compression ratio across eligible layers, although different layers might still retain different absolute numbers of parameters and FLOPs.

Table 3: Training augmentation for the ViT-B runs.

Component	Parameter	Value
create_transform	input_size	(3, 224, 224)
	is_training	True
	color_jitter	0.3
	auto_augment	"rand-m9-mstd0.5-inc1"
	interpolation	"bicubic"
	re_prob	0.25
	re_mode	"pixel"
	re_count	1
	mean	(0.5, 0.5, 0.5)
	std	(0.5, 0.5, 0.5)
Mixup	mixup_alpha	0.8
	cutmix_alpha	1.0
	prob	1.0
	switch_prob	0.5
	mode	"batch"
	label_smoothing	0.1
	num_classes	1000

Energy truncation. Given a single matrix, the retained energy metric when keeping up to rank P is defined as

$$E(P) = \frac{\sum_{i=1}^{P} \boldsymbol{\sigma}_i^2}{\sum_{i=1}^{R} \boldsymbol{\sigma}_i^2} = 1 - \frac{\|\boldsymbol{W} - \widehat{\boldsymbol{W}}\|_F^2}{\|\boldsymbol{W}\|_F^2},$$

where $R = \min\{M, N\}$, M and N are the dimensions of $\boldsymbol{W}$, and $\widehat{\boldsymbol{W}}$ denotes the reconstruction of the compressed matrix. Similar formulations are used in Alvarez & Salzmann (2017); Yang et al. (2020b); Liebenwein et al. (2021).

Given a fixed retained energy threshold ρ, the energy criterion selects a per-layer rank

$$P^* = \min\{P : E(P) \geq \rho\},$$

i.e., the lowest rank that maintains a relative squared error of at most $1 - \rho$. In particular, for each model, we sweep $\{0.7, 0.75, 0.8, 0.85, 0.9, 0.925, 0.95, 0.975, 0.99, 0.995, 0.999, 0.9995, 0.9999, 0.99995, 0.99996, 0.99997, 0.99998, 0.99999\}$.

F Q3R Refresh Period Ablation

Throughout our experiments, we use a refresh period of 5 for Q3R, following their main experiments. Sweeping the period in addition to the target rank and regularization strength would have been impractical, so we use their recommended configuration.

However, for completeness, we also swept the refresh interval over a small number of runs to probe whether one could use a larger period, which is more efficient, and what the corresponding accuracy tradeoff would be. We limit ourselves to ViT-B/16.

Results are presented in Table 4. Q3R appears to be surprisingly robust in the cases we tested. Note, however, that this might differ in other models, scales, or training horizons.

G Additional Vision Model Results

Here, we report the complete uniform compression results for the vision models. Results are shown for ViT-B/16 in Table 7, ResNet-50 in Table 8, ViT-L/16 in Table 5, and ResNet-18 in Table 6.

Table 4: **Q3R refresh interval ablation on ViT-B/16.** ImageNet-1K top-1 accuracy under uniform retained parameter ratios and different configurations.

Setting	Interval	21%	31%	41%	51%	60%	70%	80%	90%	Full
	5	29.6	75.2	76.0	76.1	76.2	76.3	76.3	76.3	76.3
$\lambda = 8.0$	20	34.8	75.2	75.8	76.1	76.3	76.4	76.4	76.4	76.4
target rank 0.05	50	33.3	75.1	75.8	76.1	76.3	76.3	76.3	76.4	76.4
learning rate 5×10^{-5}	100	32.9	75.0	75.8	76.0	76.2	76.2	76.3	76.3	76.3
	300	44.4	74.8	75.5	75.8	75.9	76.0	76.0	76.0	76.0
	5	0.1	0.2	21.2	71.9	80.7	82.5	82.7	82.8	82.8
$\lambda = 3.0$	20	0.1	0.2	19.3	72.0	80.7	82.5	82.7	82.8	82.8
target rank 0.15	50	0.1	0.1	20.1	72.2	80.8	82.5	82.8	82.8	82.9
learning rate 5×10^{-5}	100	0.1	0.2	18.7	72.0	80.8	82.6	82.8	82.8	82.9
	300	0.1	0.2	22.1	72.2	80.6	82.6	82.7	82.8	82.9
	5	0.1	0.1	0.1	9.4	53.5	73.0	77.3	78.7	81.0
$\lambda = 5.0$	20	0.1	0.1	0.1	10.0	53.5	73.0	77.3	78.7	81.1
target rank 0.10	50	0.1	0.1	0.1	8.7	54.1	72.8	77.2	78.7	81.0
learning rate 1×10^{-5}	100	0.1	0.1	0.2	9.1	47.5	72.6	77.4	78.7	81.1
	300	0.1	0.1	0.1	8.6	48.8	70.9	77.2	78.5	81.0

For each table, the different columns indicate the retention ratio (and Full indicates the uncompressed model). Top results are bolded, and second-top results are underlined. For the ViT-B/16 table, where we swept learning rates, $\diamond$ is used for 10^{-5}, and $\dagger$ for 5×10^{-5}.

Notation. For Q3R, the first argument denotes its regularization strength, and the second its target rank. For LoRITa, its only argument denotes the regularization strength (in the form of weight decay). For all SLORR variants, the only argument denotes regularization strength.

Table 5: Complete uniform compression results for ViT-L/16 continued training.

Method	10%	20%	30%	40%	50%	60%	70%	80%	90%	95%	Full
Baseline	**0.2**	0.6	7.5	38.9	63.5	75.2	80.6	83.2	84.4	84.8	85.8
LoRITa(0.0001)	0.1	**0.7**	12.5	45.8	67.2	76.9	81.3	83.5	84.5	84.9	85.5
LoRITa(0.001)	0.1	**0.7**	12.5	45.8	67.2	76.9	81.3	83.5	84.5	84.9	85.5
LoRITa(0.01)	0.1	**0.7**	**12.7**	44.9	66.8	76.9	81.2	83.4	84.5	84.8	85.5
LoRITa(0.05)	0.1	**0.7**	11.5	44.6	66.7	76.8	81.2	83.4	84.4	84.8	85.5
LoRITa(0.3)	**0.2**	0.4	7.5	38.8	63.9	75.8	80.7	83.0	84.3	84.5	85.5
LoRITa(1)	**0.2**	0.2	0.3	3.6	36.4	66.5	77.0	81.1	83.1	83.7	85.1
SLORR-Nuc(0.00002)	**0.2**	0.6	9.1	42.4	65.7	76.5	81.3	83.5	84.6	84.9	85.7
SLORR-Nuc(0.0002)	**0.2**	0.4	8.5	43.8	70.0	79.1	82.5	84.1	84.8	85.0	85.5
SLORR-Nuc(0.001)	0.1	0.6	12.2	54.0	77.0	80.3	81.1	81.5	81.5	81.5	81.5
SLORR-Nuc(0.00005)	**0.2**	0.6	7.8	41.5	66.6	77.1	81.6	83.8	84.7	84.9	85.7
SLORR-Hoyer(0.00005)	0.1	0.4	7.0	38.2	64.1	76.2	81.4	83.8	84.8	85.2	85.8
SLORR-Hoyer(0.0002)	0.1	0.2	7.0	42.8	65.5	77.5	82.8	84.6	**85.3**	**85.5**	**85.9**
SLORR-Hoyer(0.0005)	0.1	0.2	7.9	52.2	73.4	82.0	**84.1**	**84.9**	85.1	85.2	85.4
SLORR-Hoyer(0.001)	0.1	0.2	3.3	54.7	78.0	82.4	83.6	84.1	84.3	84.3	84.4
SLORR-Hoyer(0.0015)	0.1	0.2	5.8	62.1	**80.1**	**82.6**	83.6	83.9	84.1	84.1	84.1
SLORR-Hoyer-D(1)	0.1	0.4	9.2	46.9	69.2	78.5	82.4	84.2	85.1	85.3	85.7
SLORR-Hoyer-D(2)	0.1	0.2	6.8	51.4	72.8	80.4	83.3	84.6	85.2	85.3	85.5
SLORR-Hoyer-D(5)	0.1	0.2	4.0	48.7	73.1	81.7	83.2	83.5	83.6	83.6	83.7
SLORR-Hoyer-D(8)	0.1	0.1	5.9	46.6	77.1	80.1	80.6	80.8	80.8	80.8	80.8
SLORR-Hoyer-D(10)	0.1	0.1	5.0	**65.6**	76.6	77.9	78.2	78.2	78.2	78.2	78.3

Table 6: Complete uniform compression results for ResNet-18 pretraining.

Method	**14%**	**23%**	**33%**	**42%**	**52%**	**62%**	**71%**	**81%**	**90%**	**95%**	**Full**
Baseline	0.1	0.1	0.2	0.2	1.5	4.4	19.1	25.7	45.8	48.7	66.2
Q3R(1, 0.0025)	0.1	0.2	0.9	4.6	12.8	26.9	39.4	48.4	52.2	55.4	60.0
Q3R(1, 0.01)	0.1	0.3	1.6	15.7	32.6	42.4	48.6	54.0	56.0	57.8	62.1
Q3R(5, 0.01)	**0.3**	5.3	18.7	34.3	33.2	47.0	50.2	52.7	53.4	54.2	55.2
Q3R(1, 0.025)	0.2	0.6	30.8	53.2	57.0	60.0	62.8	63.5	63.6	63.9	64.0
Q3R(5, 0.025)	0.1	0.2	26.2	**62.1**	**63.3**	63.3	63.3	63.3	63.3	63.3	63.3
Q3R(1, 0.05)	0.1	0.3	1.2	17.4	42.5	**64.0**	**65.6**	65.9	65.8	65.9	65.9
Q3R(2, 0.05)	0.1	0.2	0.9	18.0	46.3	61.4	64.9	65.6	65.7	65.7	65.7
Q3R(5, 0.05)	0.1	0.2	0.5	3.5	19.4	50.0	63.7	65.4	65.8	65.9	65.9
Q3R(2, 0.1)	0.1	0.1	0.4	1.7	8.4	24.3	41.4	51.2	59.8	62.8	66.3
Q3R(0.01, 0.15)	0.1	0.1	0.3	1.2	3.3	6.0	21.6	35.6	49.4	55.6	66.5
Q3R(2, 0.2)	0.1	0.1	0.1	0.4	0.9	2.0	15.9	28.0	40.2	46.2	59.8
LoRITa(0.0001)	0.1	0.2	1.0	6.6	26.2	45.6	53.9	62.9	66.2	66.5	67.1
LoRITa(0.001)	0.1	0.2	0.6	7.9	27.0	46.9	59.9	64.0	66.0	66.5	67.0
LoRITa(0.01)	0.1	0.1	0.3	2.5	22.0	43.1	56.8	64.0	65.9	66.6	67.3
LoRITa(0.05)	0.1	0.2	0.3	6.9	37.2	58.7	65.1	**67.1**	**67.5**	**67.7**	**67.9**
LoRITa(0.3)	0.1	0.1	1.2	23.9	55.5	63.2	65.0	65.8	66.0	66.0	66.0
LoRITa(0.5)	0.1	0.1	1.4	43.0	55.7	63.1	64.0	64.1	64.2	64.2	64.2
LoRITa(1)	0.1	0.2	3.8	38.1	48.8	59.5	60.7	60.7	60.7	60.7	60.8
SLORR-Nuc(0.00001)	0.1	0.2	0.5	1.5	5.6	16.0	25.8	51.9	59.1	62.1	66.6
SLORR-Nuc(0.00005)	0.1	0.3	0.4	1.6	17.6	34.8	43.5	62.2	65.8	66.6	**67.9**
SLORR-Nuc(0.0001)	0.1	0.1	0.3	1.9	11.1	48.3	58.7	63.9	66.0	66.8	67.8
SLORR-Nuc(0.0005)	0.1	0.1	0.2	0.2	0.7	4.9	13.6	43.9	54.5	55.4	60.3
SLORR-Nuc(0.001)	0.1	0.1	0.2	0.1	0.4	2.2	37.6	52.2	54.4	55.5	56.2
SLORR-Hoyer(0.0002)	0.1	0.2	0.3	0.9	3.5	17.2	37.9	48.8	58.0	61.2	66.5
SLORR-Hoyer(0.0005)	0.1	0.2	0.6	0.6	2.1	13.6	31.1	50.1	57.7	61.3	66.7
SLORR-Hoyer(0.001)	0.1	0.2	0.1	0.6	6.2	13.8	36.4	53.5	57.9	61.4	66.4
SLORR-Hoyer(0.002)	0.1	0.1	0.1	0.4	0.8	11.3	30.0	43.5	54.9	59.0	66.2
SLORR-Hoyer(0.005)	0.1	0.1	0.1	0.3	1.5	7.4	23.6	46.8	58.6	62.2	65.2
SLORR-Hoyer(0.007)	0.1	0.1	0.1	0.2	0.8	5.2	36.1	54.5	58.8	61.3	64.4
SLORR-Hoyer(0.01)	0.1	0.2	0.2	0.1	0.7	8.5	27.4	40.8	57.0	60.9	63.3
SLORR-Hoyer-D(0.05)	0.1	0.2	0.5	1.0	1.6	3.0	8.3	23.8	44.6	53.9	66.3
SLORR-Hoyer-D(0.1)	0.1	0.1	0.3	1.5	3.8	14.6	31.3	49.3	58.2	61.0	66.2
SLORR-Hoyer-D(0.3)	0.1	0.5	1.3	4.2	10.6	32.9	49.8	58.2	62.5	63.7	65.7
SLORR-Hoyer-D(0.5)	0.1	0.2	1.2	3.3	19.8	46.9	55.6	60.8	63.6	64.3	65.9
SLORR-Hoyer-D(1)	0.1	0.2	3.9	26.7	45.5	54.9	60.1	62.4	63.9	64.5	65.1
SLORR-Hoyer-D(5)	0.1	0.6	28.3	53.0	58.9	61.1	61.6	61.9	61.9	61.9	61.9
SLORR-Hoyer-D(10)	0.1	**8.8**	**38.8**	54.9	58.7	59.3	59.4	59.5	59.6	59.6	59.6

Table 7: Complete uniform compression results for ViT-B/16 continued training.

Method	**11%**	**21%**	**31%**	**41%**	**51%**	**60%**	**70%**	**80%**	**90%**	**95%**	**Full**
Baseline$^{\diamond}$	0.1	0.1	0.1	0.7	12.1	46.5	67.0	76.8	80.9	81.9	**84.9**
Baseline†	0.1	0.1	0.2	1.1	17.8	52.4	69.4	77.3	80.7	81.6	84.1
Q3R(1, 0.05)$^{\diamond}$	0.1	0.1	0.1	0.1	0.2	5.1	36.0	65.6	75.5	78.0	84.4
Q3R(2, 0.05)$^{\diamond}$	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.1	0.1	0.2	83.4
Q3R(3, 0.05)$^{\diamond}$	0.1	0.1	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	81.8
Q3R(5, 0.05)$^{\diamond}$	0.1	0.1	0.1	0.1	0.7	3.8	17.0	45.1	53.8	58.9	76.6
Q3R(1, 0.1)$^{\diamond}$	0.1	0.1	0.1	0.1	0.9	25.9	57.4	73.4	79.0	80.5	84.5
Q3R(2, 0.1)$^{\diamond}$	0.1	0.1	0.1	0.2	0.4	9.2	34.2	50.8	66.1	70.9	83.9
Q3R(3, 0.1)$^{\diamond}$	0.1	0.1	0.1	0.2	0.4	2.2	1.5	10.2	37.1	58.2	83.1
Q3R(5, 0.1)$^{\diamond}$	0.1	0.1	0.1	0.1	9.3	53.4	73.3	77.3	78.7	79.1	81.1
Q3R(0.01, 0.15)$^{\diamond}$	0.1	0.1	0.1	0.6	12.3	47.0	67.3	76.7	80.8	81.9	**84.9**
Q3R(1, 0.15)$^{\diamond}$	0.1	0.1	0.1	0.1	3.6	41.1	65.9	76.5	80.7	81.8	84.7
Q3R(3, 0.15)$^{\diamond}$	0.1	0.1	0.1	0.2	1.7	41.8	51.5	70.6	77.2	79.8	83.8
Q3R(2, 0.2)$^{\diamond}$	0.1	0.1	0.1	0.1	3.4	44.8	69.0	78.4	81.7	82.5	84.5
Q3R(3, 0.2)$^{\diamond}$	0.1	0.1	0.1	0.2	3.1	45.4	70.7	78.8	81.9	82.6	84.2
Q3R(3, 0.05)†	0.1	0.1	70.8	76.0	77.4	77.9	78.2	78.4	78.5	78.5	78.6
Q3R(5, 0.05)†	0.1	2.3	75.1	**76.3**	76.9	77.2	77.3	77.3	77.3	77.4	77.4
Q3R(8, 0.05)†	0.1	29.6	**75.2**	76.0	76.1	76.2	76.3	76.3	76.3	76.3	76.3
Q3R(10, 0.05)†	0.1	**44.5**	75.0	75.5	75.7	75.8	75.9	75.9	75.9	75.9	75.9
Q3R(3, 0.1)†	0.1	0.1	0.3	65.4	80.9	**81.3**	81.5	81.6	81.7	81.8	81.8
Q3R(5, 0.1)†	0.1	0.1	3.3	71.9	**81.0**	81.2	81.3	81.4	81.4	81.4	81.4
Q3R(3, 0.15)†	0.1	0.1	0.2	21.4	71.9	80.6	**82.5**	82.7	82.7	82.8	82.8
Q3R(5, 0.15)†	0.1	0.1	0.3	34.5	74.6	81.1	**82.5**	82.7	82.7	82.8	82.8
Q3R(3, 0.2)†	0.1	0.1	0.2	4.9	57.8	75.6	80.7	82.3	83.1	83.2	83.3
Q3R(5, 0.2)†	0.1	0.1	0.3	8.7	61.7	76.7	81.1	82.6	83.2	83.3	83.3
LoRITa(0.0001)$^{\diamond}$	0.1	0.1	0.2	1.0	17.8	52.2	70.0	77.7	81.2	82.2	84.7
LoRITa(0.001)$^{\diamond}$	0.1	0.1	0.2	1.0	17.8	52.2	70.0	77.7	81.2	82.2	84.7
LoRITa(0.01)$^{\diamond}$	0.1	0.1	0.1	1.0	17.2	51.9	70.1	77.9	81.3	82.1	84.8
LoRITa(0.05)$^{\diamond}$	0.1	0.1	0.1	1.1	17.7	52.6	70.3	77.8	81.2	82.2	84.8
LoRITa(0.3)$^{\diamond}$	0.1	0.1	0.1	0.6	12.3	48.4	68.3	77.0	80.8	81.9	84.7
LoRITa(0.5)$^{\diamond}$	0.1	0.1	0.2	0.4	5.9	40.6	65.1	75.9	80.2	81.4	84.6
LoRITa(1)$^{\diamond}$	0.1	0.1	0.1	0.1	0.2	8.3	42.7	68.1	76.2	78.7	84.4
LoRITa(0.0001)†	0.1	0.2	1.0	15.4	47.1	67.7	76.4	80.2	81.9	82.6	84.0
LoRITa(0.001)†	0.1	0.1	0.9	13.2	47.2	67.6	76.2	80.1	81.9	82.6	84.0
LoRITa(0.01)†	0.1	0.1	1.1	14.0	47.4	68.0	76.5	80.0	81.8	82.5	84.1
LoRITa(0.05)†	0.1	0.1	0.7	15.4	48.3	68.5	76.6	80.2	82.0	82.6	84.0
LoRITa(0.3)†	0.1	0.1	0.2	3.3	35.8	64.9	75.0	79.5	81.5	82.2	83.7
LoRITa(0.5)†	0.1	0.1	0.2	0.6	10.7	47.7	70.0	77.2	80.2	81.4	83.5
LoRITa(1)†	0.1	0.1	0.1	0.1	0.2	0.2	1.4	4.5	11.6	15.4	81.8
SLORR-Nuc(0.001)$^{\diamond}$	0.1	0.1	0.2	1.2	30.6	64.9	75.5	79.9	81.7	82.1	82.8
SLORR-Nuc(0.002)$^{\diamond}$	0.1	0.1	0.2	1.1	36.9	68.8	75.9	78.0	78.9	79.1	79.2
SLORR-Nuc(0.003)$^{\diamond}$	0.1	0.1	0.1	6.7	47.8	69.2	73.4	74.6	75.0	75.1	75.1
SLORR-Nuc(0.001)†	0.1	0.5	66.4	73.7	74.9	75.3	75.3	75.3	75.4	75.3	75.3
SLORR-Nuc(0.0002)†	0.1	0.2	0.3	16.7	55.4	74.8	79.4	81.8	82.5	82.8	83.2
SLORR-Nuc(0.0001)†	0.1	0.1	0.2	7.6	52.1	72.1	78.5	81.3	82.6	83.0	83.7
SLORR-Nuc(0.0005)†	0.1	0.2	4.0	63.7	77.2	79.7	80.4	80.6	80.7	80.7	80.7
SLORR-Hoyer(0.0002)$^{\diamond}$	0.1	0.1	0.1	1.1	21.3	58.2	74.1	80.4	82.4	83.2	84.8
SLORR-Hoyer(0.0005)$^{\diamond}$	**0.2**	0.1	0.1	1.4	24.9	67.1	78.9	82.0	**83.4**	**83.8**	84.4
SLORR-Hoyer(0.001)$^{\diamond}$	0.1	0.1	0.1	1.3	32.3	75.3	81.3	**82.8**	**83.4**	83.5	83.5
SLORR-Hoyer(0.002)$^{\diamond}$	0.1	0.1	0.1	6.6	66.2	79.5	81.1	81.5	81.7	81.6	81.7
SLORR-Hoyer(0.003)$^{\diamond}$	0.1	0.1	0.1	34.7	74.7	78.3	79.2	79.6	79.6	79.6	79.8
SLORR-Hoyer(0.005)$^{\diamond}$	0.1	0.2	1.8	61.5	73.0	75.4	75.9	75.9	76.0	76.0	76.1
SLORR-Hoyer(0.0002)†	0.1	0.1	0.1	0.7	10.1	67.9	80.8	**82.8**	83.3	83.5	83.8
SLORR-Hoyer(0.0005)†	0.1	0.1	0.1	6.7	70.6	80.6	82.1	82.7	82.8	82.9	83.0
SLORR-Hoyer(0.0007)†	0.1	0.1	0.2	15.9	75.7	81.0	82.0	82.3	82.6	82.6	82.6
SLORR-Hoyer(0.001)†	0.1	0.1	0.1	62.3	78.7	80.8	81.5	81.7	81.9	81.9	82.0

Table 7: Complete uniform compression results for ViT-B/16 continued training (continued).

Method	**11%**	**21%**	**31%**	**41%**	**51%**	**60%**	**70%**	**80%**	**90%**	**95%**	**Full**
SLORR-Hoyer(0.00125)†	0.1	0.1	0.3	65.8	78.8	80.4	81.1	81.3	81.3	81.4	81.4
SLORR-Hoyer-D(0.25)◇	0.1	0.1	0.1	0.7	14.3	50.3	69.6	77.8	81.5	82.5	84.8
SLORR-Hoyer-D(0.5)◇	0.1	0.1	0.1	0.7	16.6	52.6	71.1	78.6	81.9	82.8	84.7
SLORR-Hoyer-D(1)◇	0.1	0.1	0.1	1.0	22.1	58.2	74.0	80.0	82.5	83.2	84.5
SLORR-Hoyer-D(2)◇	0.1	0.1	0.1	1.8	33.9	67.2	77.6	81.1	82.6	83.0	83.7
SLORR-Hoyer-D(0.1)†	0.1	0.1	0.2	2.2	27.4	59.0	73.4	78.9	81.8	82.5	84.0
SLORR-Hoyer-D(0.25)†	0.1	0.1	0.2	4.3	40.0	66.3	76.5	80.9	82.6	83.1	83.9
SLORR-Hoyer-D(0.5)†	0.1	0.2	0.2	8.5	50.7	70.8	79.8	82.1	83.0	83.2	83.6
SLORR-Hoyer-D(1)†	0.1	0.1	0.4	26.9	62.7	79.2	81.7	82.3	82.5	82.5	82.6
SLORR-Hoyer-D(3)†	0.1	0.2	56.4	73.2	75.5	75.7	75.7	75.7	75.7	75.7	75.7

Table 8: Complete uniform compression results for ResNet-50 continued training.

Method	**17%**	**26%**	**35%**	**45%**	**54%**	**63%**	**72%**	**81%**	**90%**	**95%**	**Full**
Baseline	0.1	0.2	0.4	9.4	40.3	62.5	67.9	71.9	74.0	74.5	76.1
Q3R(8, 0.0025)	0.1	0.3	1.4	14.3	53.8	64.6	68.6	69.5	70.2	70.5	71.4
Q3R(20, 0.0025)	**0.2**	0.6	11.3	46.7	60.2	63.1	64.0	64.6	64.8	65.0	65.6
Q3R(8, 0.01)	0.1	1.0	3.2	36.3	60.7	68.0	69.4	70.1	70.4	70.5	71.1
Q3R(20, 0.01)	0.1	**5.3**	41.9	59.1	63.2	64.6	65.1	65.4	65.6	65.6	65.9
Q3R(5, 0.025)	0.1	0.4	7.5	49.7	65.6	70.2	71.3	71.9	72.2	72.5	73.1
Q3R(8, 0.025)	**0.2**	0.8	29.0	63.1	67.4	70.0	70.9	71.2	71.4	71.5	71.9
Q3R(1, 0.05)	0.1	0.3	0.7	25.8	58.0	69.4	72.5	73.7	74.4	74.6	75.3
Q3R(2, 0.05)	**0.2**	0.3	4.4	38.4	67.2	71.8	73.3	73.5	74.1	74.3	74.6
Q3R(3, 0.05)	**0.2**	0.4	8.1	49.9	69.9	73.0	73.5	73.7	74.2	74.2	74.5
Q3R(5, 0.05)	0.1	0.3	11.7	60.8	69.5	73.0	73.4	73.6	73.6	73.7	74.1
Q3R(8, 0.05)	**0.2**	0.5	8.5	63.7	72.5	73.2	73.4	73.5	73.5	73.6	73.7
Q3R(12, 0.05)	0.1	0.4	11.3	65.6	72.7	72.9	73.1	73.2	73.3	73.3	73.4
Q3R(20, 0.05)	0.1	0.4	9.1	63.7	72.7	72.8	72.8	72.9	72.9	72.9	73.0
Q3R(1, 0.1)	0.1	0.2	1.0	32.5	61.9	70.8	73.4	74.6	75.0	75.3	75.5
Q3R(2, 0.1)	0.1	0.2	3.1	40.5	65.1	72.4	74.0	**74.9**	**75.2**	**75.4**	75.4
Q3R(0.01, 0.15)	**0.2**	0.2	0.3	10.4	43.2	62.2	67.7	71.8	73.9	74.5	76.0
Q3R(1, 0.15)	0.1	0.2	1.2	18.6	54.2	68.5	72.4	74.5	75.0	75.2	75.6
Q3R(2, 0.2)	0.1	0.1	0.5	19.2	50.6	68.0	71.8	74.1	75.1	75.3	75.6
LoRITa(0.00001)	**0.2**	0.2	0.4	7.0	37.6	61.9	67.5	71.7	73.8	74.3	76.0
LoRITa(0.0001)	**0.2**	0.2	0.4	7.0	37.6	61.9	67.5	71.7	73.8	74.3	76.0
LoRITa(0.001)	0.1	0.2	0.4	6.6	39.8	61.5	67.5	71.8	73.9	74.5	76.1
LoRITa(0.01)	**0.2**	0.2	0.4	5.9	39.5	61.8	67.3	71.5	73.8	74.5	76.0
LoRITa(0.05)	**0.2**	0.2	0.3	5.9	40.3	61.7	67.5	71.7	74.0	74.7	**76.2**
LoRITa(0.3)	0.1	0.2	0.4	6.0	38.8	61.5	67.5	71.5	73.8	74.4	76.1
LoRITa(0.5)	**0.2**	0.1	0.4	6.6	38.5	61.2	67.8	71.7	73.9	74.3	76.1
SLORR-Nuc(0.0002)	0.1	0.2	0.5	13.0	52.8	67.0	71.5	74.0	75.0	75.2	75.7
SLORR-Nuc(0.0005)	**0.2**	0.1	0.2	20.5	64.0	72.1	74.3	**74.9**	75.1	75.2	75.3
SLORR-Nuc(0.001)	0.1	0.1	0.2	37.1	71.6	**74.0**	**74.5**	74.6	74.6	74.6	74.7
SLORR-Nuc(0.002)	0.1	0.1	0.1	58.1	**72.9**	73.2	73.3	73.3	73.3	73.3	73.3
SLORR-Nuc(0.003)	0.1	0.2	1.0	62.0	71.8	72.0	72.0	72.0	72.0	72.0	72.0
SLORR-Nuc(0.005)	0.1	0.2	1.9	66.2	69.4	69.6	69.6	69.6	69.6	69.6	69.6
SLORR-Hoyer(0.0002)	0.1	0.2	0.4	23.4	56.8	69.2	73.4	74.7	**75.2**	**75.4**	75.5
SLORR-Hoyer(0.0005)	0.1	0.2	0.6	44.1	68.2	72.5	74.1	74.5	74.6	74.6	74.6
SLORR-Hoyer(0.001)	0.1	0.2	1.6	46.8	69.9	72.8	73.5	73.6	73.7	73.7	73.6
SLORR-Hoyer(0.002)	0.1	0.1	0.9	55.5	70.7	72.0	72.2	72.4	72.3	72.3	72.4
SLORR-Hoyer(0.003)	0.1	0.1	1.5	61.1	70.6	71.2	71.2	71.3	71.3	71.3	71.3
SLORR-Hoyer(0.005)	0.1	0.1	8.2	68.1	69.7	69.8	69.9	69.9	69.9	69.9	69.9
SLORR-Hoyer-D(0.1)	**0.2**	0.2	0.6	17.5	49.8	67.7	71.9	74.1	75.1	**75.4**	75.9
SLORR-Hoyer-D(0.25)	0.1	0.3	0.6	29.0	59.3	71.7	74.1	**74.9**	**75.2**	**75.4**	75.5
SLORR-Hoyer-D(0.5)	0.1	0.4	0.7	45.0	69.5	73.7	74.3	74.4	74.5	74.4	74.5
SLORR-Hoyer-D(1)	**0.2**	0.6	16.6	68.1	72.4	73.0	73.0	73.0	73.0	73.0	73.0
SLORR-Hoyer-D(1.5)	0.1	0.7	32.4	**69.2**	70.9	70.9	70.9	70.9	70.9	70.9	70.9
SLORR-Hoyer-D(2)	0.1	1.1	**46.5**	68.2	68.5	68.5	68.5	68.6	68.6	68.5	68.6

G.1 Sensitivity to Training Seed

We repeat ResNet-50 continued training with SLORR-Hoyer using training seeds 42, 43, and 44, fixing the regularization strength to $\lambda = 0.002$, the learning rate to 5×10^{-5}, and the remaining training settings. Each model is compressed using the same uniform retention targets. We report top-1 accuracy for each training seed and the mean and sample standard deviation across seeds in Table 9. Variability appears to be small.

Table 9: **Training-seed sensitivity for ResNet-50 with SLORR-Hoyer ($\lambda = 0.002$).** Entries are ImageNet-1K validation top-1 accuracy (%); SD denotes sample standard deviation across three training seeds. Retention targets specify the requested uniform compression setting, not the exact achieved whole-model parameter ratio. A target of 1.0 denotes the uncompressed model.

Retention target	Seed 42	Seed 43	Seed 44	Mean $\pm$ SD
1.00	72.35	72.37	72.29	72.34 ± 0.04
0.95	72.34	72.37	72.30	72.34 ± 0.04
0.90	72.33	72.34	72.29	72.32 ± 0.03
0.80	72.37	72.38	72.30	72.35 ± 0.05
0.70	72.24	72.32	72.25	72.27 ± 0.04
0.60	71.99	72.11	72.05	72.05 ± 0.06
0.50	70.68	70.65	70.46	70.60 ± 0.12
0.40	55.47	53.42	55.44	54.77 ± 1.18
0.30	0.93	0.41	0.35	0.56 ± 0.32
0.20	0.14	0.12	0.11	0.12 ± 0.02
0.10	0.14	0.12	0.15	0.13 ± 0.01

H Overhead on Vision Models

For our main experiments on ResNet-50 and ViT-B/16, we used NVIDIA A100-80GB GPUs. For the ViT-L/16 experiments, we used B200 GPUs. For the ResNet-18 experiments, we used NVIDIA A100-40GB GPUs. To obtain controlled overhead measurements, we do not rely on end-to-end times from full training runs. Instead, our procedure is as follows. For each model, we conduct 10 short benchmark runs on the same hardware as in the main experiments. For each run, we first perform 10 warmup training steps and then measure time and peak memory over 300 training steps under the same setup as in our training runs. The measurement takes every part of training into account: data loading and augmentation, transfer to the device, training steps, regularization, etc. In the case of Q3R, this includes periodic SVDs. This does not include, for example, evaluation or checkpointing. Each benchmark run repeats this for all methods, including the unregularized baseline. We also tried using autotuning during the Polar Express compilation step, but this always yielded worse performance, so we do not use it.

For SLORR, we also vary the number of Polar Express steps; for Q3R, we vary the refresh interval and the target rank.

For each run, we compute the normalized time as the time taken with a given method or regularizer divided by the time taken with no regularizer in the same run. More precisely, if $T_{m,i}$ is the time taken by method m in run i, measured over the 300 timed training steps, and $T_{\text{base},i}$ is the corresponding time for the unregularized baseline, then

$$\text{NormTime}_{m,i} = \frac{T_{m,i}}{T_{\text{base},i}}.$$

Thus, a normalized time of $\times 1.037$ means that the method takes 3.7% more time than unregularized training in that setting. Normalized memory is measured similarly, using peak memory instead of time. We show the means and standard deviations of both the normalized and raw measurements in Table 10.

H.1 Scaling Up with Vision Transformers

To support our scalability claims, we also measure overhead at different ViT scales. In addition to standard configurations (ViT-T/B/L/H), we include interpolated variants to obtain a smooth overhead curve across model sizes. We use the same settings as in our ViT overhead measurements, but vary the batch size so that all regularizers fit into a B200 GPU, with a maximum of 1024. These models span from the ViT-T scale (on the order of 6M parameters) to ViT-H (over 0.6B parameters). We detail the configurations used in Table 11. We then measure overhead following the protocol described in Appendix H: normalized time is computed by dividing the measured time of each method by the measured time of the corresponding unregularized run under the same model and benchmark setup, and normalized memory is computed analogously using peak memory. Results are shown in Figure 2. For Q3R, the target rank is fixed to 0.1, and different refresh intervals are shown for completeness.

Table 11: Vision Transformer (ViT) model configurations and parameter counts used in our overhead measurements.

Model	Patch Size	Embed Dim	Depth	# Heads	MLP Ratio	Batch Size	Params (M)
`vit_ti`	16	192	12	3	4.0	1024	5.72
`vit_256`	16	256	12	4	4.0	1024	9.98
`vit_s`	16	384	12	6	4.0	1024	22.05
`vit_512`	16	512	12	8	4.0	1024	38.84
`vit_b`	16	768	12	12	4.0	1024	86.57
`vit_b24`	16	768	24	12	4.0	1024	171.62
`vit_960_24`	16	960	24	12	4.0	1024	267.61
`vit_l`	16	1024	24	16	4.0	896	304.33
`vit_1152_28`	16	1152	28	16	4.0	720	448.60
`vit_h`	16	1280	32	16	4.0	512	632.20

Table 10: **Training overhead.** Average wall-clock time and peak memory per 300 optimization steps. Means and standard deviations over 10 jobs for each method are reported.

Model	Method	Time/300 steps (s)	Peak Mem (GB)	Norm. Time	Norm. Mem
ViT-B/16	Baseline	258.975 ± 6.500	71.584 ± 0.001	×1.000 ± 0.000	×1.000 ± 0.000
	SLORR-Hoyer (steps=6)	268.447 ± 6.470	71.923 ± 0.000	×1.037 ± 0.003	×1.005 ± 0.000
	SLORR-Hoyer (steps=8)	271.520 ± 6.112	71.926 ± 0.000	×1.049 ± 0.004	×1.005 ± 0.000
	SLORR-Hoyer (steps=10)	274.345 ± 6.506	71.924 ± 0.001	×1.059 ± 0.003	×1.005 ± 0.000
	SLORR-Hoyer-D (steps=6)	274.687 ± 6.255	71.582 ± 0.002	×1.061 ± 0.003	×1.000 ± 0.000
	SLORR-Hoyer-D (steps=8)	277.429 ± 6.678	71.582 ± 0.002	×1.071 ± 0.003	×1.000 ± 0.000
	SLORR-Hoyer-D (steps=10)	280.014 ± 6.140	71.582 ± 0.002	×1.081 ± 0.005	×1.000 ± 0.000
	LoRITa	260.670 ± 6.372	72.153 ± 0.001	×1.007 ± 0.002	×1.008 ± 0.000
	Q3R (interval=5, rank=0.01)	426.005 ± 6.311	72.729 ± 0.001	×1.645 ± 0.016	×1.016 ± 0.000
	Q3R (interval=5, rank=0.05)	426.984 ± 6.009	72.728 ± 0.002	×1.649 ± 0.019	×1.016 ± 0.000
	Q3R (interval=5, rank=0.1)	428.857 ± 6.109	72.729 ± 0.001	×1.656 ± 0.018	×1.016 ± 0.000
	Q3R (interval=5, rank=0.15)	431.551 ± 6.355	72.727 ± 0.002	×1.667 ± 0.018	×1.016 ± 0.000
	Q3R (interval=20, rank=0.1)	310.177 ± 6.220	72.728 ± 0.002	×1.198 ± 0.006	×1.016 ± 0.000
	Q3R (interval=50, rank=0.1)	286.478 ± 6.248	72.726 ± 0.001	×1.106 ± 0.004	×1.016 ± 0.000
	Q3R (interval=100, rank=0.1)	278.859 ± 6.291	72.725 ± 0.001	×1.077 ± 0.004	×1.016 ± 0.000
	Q3R (interval=300, rank=0.1)	273.238 ± 6.271	72.724 ± 0.001	×1.055 ± 0.003	×1.016 ± 0.000
ResNet-50	Baseline	238.173 ± 5.457	46.221 ± 0.001	×1.000 ± 0.000	×1.000 ± 0.000
	SLORR-Hoyer (steps=6)	249.511 ± 5.716	46.316 ± 0.002	×1.048 ± 0.003	×1.002 ± 0.000
	SLORR-Hoyer (steps=8)	252.558 ± 5.616	46.315 ± 0.001	×1.060 ± 0.002	×1.002 ± 0.000
	SLORR-Hoyer (steps=10)	256.021 ± 5.647	46.315 ± 0.001	×1.075 ± 0.004	×1.002 ± 0.000
	SLORR-Hoyer-D (steps=6)	256.658 ± 5.717	46.221 ± 0.001	×1.078 ± 0.004	×1.000 ± 0.000
	SLORR-Hoyer-D (steps=8)	260.724 ± 5.618	46.221 ± 0.001	×1.095 ± 0.006	×1.000 ± 0.000
	SLORR-Hoyer-D (steps=10)	264.580 ± 6.477	46.222 ± 0.001	×1.111 ± 0.016	×1.000 ± 0.000
	LoRITa	239.302 ± 5.506	46.350 ± 0.001	×1.005 ± 0.006	×1.003 ± 0.000
	Q3R (interval=5, rank=0.01)	291.274 ± 5.665	46.532 ± 0.003	×1.223 ± 0.007	×1.007 ± 0.000
	Q3R (interval=5, rank=0.05)	291.162 ± 5.735	46.533 ± 0.002	×1.223 ± 0.008	×1.007 ± 0.000
	Q3R (interval=5, rank=0.1)	290.995 ± 5.594	46.531 ± 0.002	×1.222 ± 0.006	×1.007 ± 0.000
	Q3R (interval=5, rank=0.15)	290.084 ± 5.973	46.531 ± 0.001	×1.218 ± 0.012	×1.007 ± 0.000
	Q3R (interval=20, rank=0.1)	258.970 ± 5.592	46.530 ± 0.002	×1.087 ± 0.002	×1.007 ± 0.000
	Q3R (interval=50, rank=0.1)	253.010 ± 5.494	46.530 ± 0.002	×1.062 ± 0.002	×1.007 ± 0.000
	Q3R (interval=100, rank=0.1)	251.038 ± 5.510	46.529 ± 0.001	×1.054 ± 0.006	×1.007 ± 0.000
	Q3R (interval=300, rank=0.1)	249.735 ± 5.463	46.527 ± 0.001	×1.049 ± 0.005	×1.007 ± 0.000
ViT-L/16	Baseline	192.797 ± 1.644	164.376 ± 0.000	×1.000 ± 0.000	×1.000 ± 0.000
	SLORR-Hoyer (steps=6)	203.345 ± 1.227	165.582 ± 0.001	×1.055 ± 0.003	×1.007 ± 0.000
	SLORR-Hoyer (steps=8)	205.978 ± 1.196	165.583 ± 0.001	×1.068 ± 0.004	×1.007 ± 0.000
	SLORR-Hoyer (steps=10)	208.406 ± 1.064	165.582 ± 0.001	×1.081 ± 0.005	×1.007 ± 0.000
	SLORR-Hoyer-D (steps=6)	207.288 ± 1.125	164.375 ± 0.001	×1.075 ± 0.004	×1.000 ± 0.000
	SLORR-Hoyer-D (steps=8)	210.050 ± 1.210	164.375 ± 0.001	×1.090 ± 0.004	×1.000 ± 0.000
	SLORR-Hoyer-D (steps=10)	212.514 ± 1.081	164.375 ± 0.001	×1.102 ± 0.004	×1.000 ± 0.000
	LoRITa	194.632 ± 1.502	166.391 ± 0.000	×1.010 ± 0.001	×1.012 ± 0.000
	Q3R (interval=5, rank=0.01)	580.708 ± 1.323	168.420 ± 0.001	×3.012 ± 0.023	×1.025 ± 0.000
	Q3R (interval=5, rank=0.05)	585.406 ± 1.353	168.422 ± 0.001	×3.037 ± 0.022	×1.025 ± 0.000
	Q3R (interval=5, rank=0.1)	589.815 ± 1.282	168.421 ± 0.002	×3.059 ± 0.023	×1.025 ± 0.000
	Q3R (interval=5, rank=0.15)	594.638 ± 1.470	168.421 ± 0.001	×3.084 ± 0.022	×1.025 ± 0.000
	Q3R (interval=20, rank=0.1)	307.126 ± 1.312	168.419 ± 0.001	×1.593 ± 0.007	×1.025 ± 0.000
	Q3R (interval=50, rank=0.1)	250.827 ± 1.454	168.418 ± 0.002	×1.301 ± 0.004	×1.025 ± 0.000
	Q3R (interval=100, rank=0.1)	232.288 ± 1.213	168.416 ± 0.000	×1.205 ± 0.006	×1.025 ± 0.000
	Q3R (interval=300, rank=0.1)	219.497 ± 1.362	168.415 ± 0.002	×1.139 ± 0.003	×1.025 ± 0.000
ResNet-18	Baseline	169.288 ± 7.488	13.146 ± 0.000	×1.000 ± 0.000	×1.000 ± 0.000
	SLORR-Hoyer (steps=6)	169.822 ± 6.864	13.192 ± 0.000	×1.003 ± 0.022	×1.003 ± 0.000
	SLORR-Hoyer (steps=8)	171.127 ± 8.464	13.191 ± 0.000	×1.011 ± 0.025	×1.003 ± 0.000
	SLORR-Hoyer (steps=10)	169.576 ± 6.405	13.191 ± 0.000	×1.002 ± 0.027	×1.003 ± 0.000
	SLORR-Hoyer-D (steps=6)	168.660 ± 5.460	13.144 ± 0.000	×0.997 ± 0.028	×1.000 ± 0.000
	SLORR-Hoyer-D (steps=8)	170.404 ± 9.644	13.144 ± 0.000	×1.006 ± 0.022	×1.000 ± 0.000
	SLORR-Hoyer-D (steps=10)	170.196 ± 8.941	13.144 ± 0.000	×1.005 ± 0.031	×1.000 ± 0.000
	LoRITa	167.907 ± 4.230	13.191 ± 0.000	×0.993 ± 0.032	×1.003 ± 0.000
	Q3R (interval=5, rank=0.01)	174.902 ± 16.476	13.288 ± 0.001	×1.031 ± 0.051	×1.011 ± 0.000
	Q3R (interval=5, rank=0.05)	170.511 ± 9.189	13.289 ± 0.001	×1.008 ± 0.047	×1.011 ± 0.000
	Q3R (interval=5, rank=0.1)	169.531 ± 5.638	13.289 ± 0.000	×1.002 ± 0.022	×1.011 ± 0.000
	Q3R (interval=5, rank=0.15)	170.471 ± 6.208	13.288 ± 0.000	×1.008 ± 0.032	×1.011 ± 0.000
	Q3R (interval=20, rank=0.1)	171.572 ± 9.018	13.289 ± 0.000	×1.014 ± 0.038	×1.011 ± 0.000
	Q3R (interval=50, rank=0.1)	170.862 ± 7.709	13.288 ± 0.001	×1.010 ± 0.026	×1.011 ± 0.000
	Q3R (interval=100, rank=0.1)	171.271 ± 8.197	13.288 ± 0.000	×1.012 ± 0.034	×1.011 ± 0.000
	Q3R (interval=300, rank=0.1)	170.911 ± 7.846	13.288 ± 0.000	×1.010 ± 0.038	×1.011 ± 0.000

I LLM PRETRAINING EXPERIMENTAL AND IMPLEMENTATION DETAILS

Implementation. For our LLM pretraining experiments, we use the OLMo (OLMo et al., 2024) codebase[11]. For all training experiments, which use multiple GPUs, we use distributed data parallelism (DDP). Aside from adding the SLORR regularizer and benchmarking utilities, the main codebase remains mostly unchanged. SLORR is implemented as an additional loss term, similar to Listing 1. To distribute its computational cost, we partition the regularized layers across DDP ranks, so that each rank computes SLORR only for its assigned subset.

This does not introduce any extra communication. Instead, SLORR reuses the gradient synchronization already performed by DDP. Since DDP averages gradients across ranks during its standard all-reduce, each rank scales its locally computed SLORR gradients by the world size before synchronization. After the DDP all-reduce, every regularized parameter therefore receives exactly the gradient it would have obtained if the full SLORR regularizer had been computed redundantly on every rank. In this way, the computational overhead of SLORR is divided across GPUs without any additional communication cost.

Experimental details. We use a Llama-like architecture (Touvron et al., 2023) and the T5 tokenizer (Raffel et al., 2020), available at `https://huggingface.co/google-t5/t5-base`. We use the AdamW optimizer with learning rate 10^{-3} (5×10^{-4} for 1B), $\beta_1 = 0.9$, $\beta_2 = 0.95$, and weight decay set to 0.1. Gradients are clipped using a maximum global norm of 1.0. We use a cosine scheduler with 10% of the training steps dedicated to a linear warmup from 0 to the base learning rate, which is then decayed to 10% of the base learning rate. We chose these settings based on common practice for standard training and did not try other settings.

Training is performed on a subsample of the FineWeb-Edu (Penedo et al., 2024) 100BT sample[12]; evaluation and calibration sets (for SVD-LLM) are also selected from this sample but are not included in the training set.

We summarize the size configurations used in Table 12. All models use a global batch size of 512 and a sequence length of 1024. The 135M, 560M and 1B rows correspond to compute-optimal training budgets. The multipliers indicate the amount of data on which the model was trained relative to compute optimality (Hoffmann et al., 2022). The 135M models and their overtrained versions were trained on 4 H100 GPUs, while the 560M models were trained on 8 H100 GPUs.

We regularize (and factorize after training) every linear layer inside transformer blocks (we exclude the embedding matrix and the final projection layer).

Table 12: LLM model configurations and training budgets. All models use a global batch size of 512 and sequence length 1024.

Model	Hidden	Intermediate	Heads	Layers	Training steps
135M	768	2048	12	12	5,200
135M×4	768	2048	12	12	20,600
135M×8	768	2048	12	12	41,200
560M	1280	3456	20	24	21,400
1B	1536	4096	24	32	38,400

As in our vision experiments, all runs use a default seed. For transparency, we note that the 135M×8 Llama run with regularization strength 10^{-5} encountered a numerical explosion and terminated before completion. We monitored different metrics and did not observe preceding indicators of instability. We believe the crash may have been due to the particular trajectory taken by the model under this seed and hyperparameter setting, rather than due to the regularizer itself. We therefore reran this configuration with another seed and report the completed rerun.

[11] `https://github.com/allenai/olmo`

[12] `https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu/viewer/sample-100BT`

Method hyperparameters. The range of hyperparameters was not exhaustively explored and was mainly based on early experimentation. Similar to our vision experiments, reasonable ranges were determined first, where reasonable means not so low as to have no effect and not so high as to significantly degrade the model. With the chosen hyperparameters, training was mostly stable. Among the completed runs, the $\times 8$ run with the highest regularization strength reported (see Appendix J) had a loss spike early in training. This can also destabilize compression, as can be seen in Figure 5.

I.1 Compression

For compression, we adapt the SVD-LLM (Wang et al., 2025b) codebase[13] to support OLMo checkpoints and the FineWeb dataset. We substantially simplified the codebase to specialize it for our setting, and also added support for plain SVD compression. For SVD-LLM, we use 256 samples of sequence length 1024 (same as in the training procedure) of held-out FineWeb-Edu data.

For activation-aware (whitened) truncation as used in SVD-LLM, we observe that regularized models often yield rank-deficient activations, meaning that their second-moment matrices are singular. SVD-LLM uses Cholesky-based whitening, which does not support rank-deficient matrices; in practice, this is addressed by adding diagonal shift. We instead modify the codebase to use BALF-like whitening (González-Martínez, 2025), which admits rank-deficient activations in closed form. We did not observe notable end-to-end gains from this modification, though.

I.2 Evaluation

After training and compressing each model, we evaluate them according to standard practice. For perplexity, we report results on a held-out set of approximately 2M tokens. For downstream evaluations, we use the `lm-evaluation-harness` library (Gao et al., 2024) throughout and report zero-shot metrics. When available, we use `acc_norm`; otherwise, we use the `acc` field.

Tasks. We evaluate on ARC-Easy and ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), LAMBADA (Paperno et al., 2016), OpenBookQA (Mihaylov et al., 2018), and PIQA (Bisk et al., 2019). We also initially explored WinoGrande (Sakaguchi et al., 2019), but accuracy appeared to be mostly random in all cases, so we did not end up using it.

J Beyond Compute-Optimal LLM Pretraining

Here, we report results for 135M models trained for $4\times$ and $8\times$ the compute-optimal token budget (Hoffmann et al., 2022). As noted in the main text, we find that the compressibility of all models, including regularized and unregularized ones, appears to be reduced as they ingest more data. However, in all cases, regularized models are still more compressible.

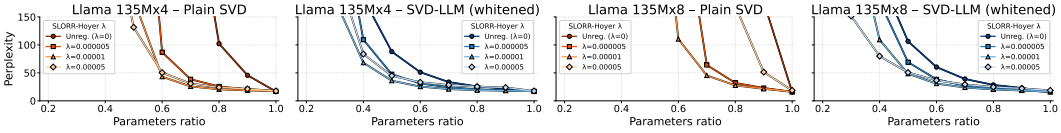

Figure 5: Llama models trained for $4\times$ and $8\times$ the compute-optimal token budget. Perplexity values are clipped to 150, as in our main text figures.

K Beyond AdamW LLM Pretraining

To evaluate whether the gains of SLORR transfer beyond the typical AdamW choice of optimizer, we additionally perform experiments on the 135M sizes with Muon (Jordan et al., 2024), SOAP (Vyas et al., 2025) and Kron.

Settings. To test whether SLORR's effect is specific to AdamW, we repeat the 135M language-model experiments with three alternative optimizers: Muon(Jordan et al., 2024), SOAP(Vyas et al.,

[13] https://github.com/AIoT-MLSys-Lab/SVD-LLM

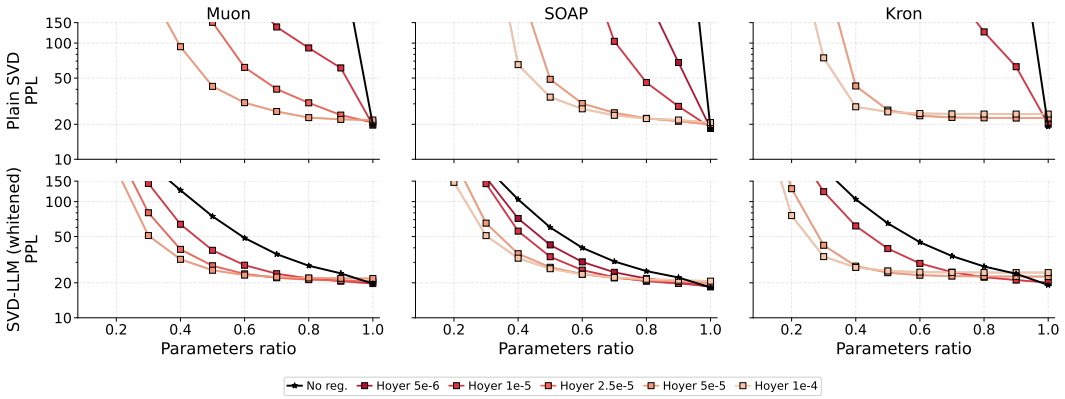

Figure 6: SLORR-Hoyer with different optimizers on Llama 135M. Columns correspond to Muon, SOAP, and Kron; rows show Plain SVD and SVD-LLM (whitened). Red curves indicate regularization strengths listed in the legend; black curves show matched unregularized baselines. Perplexity is shown on a logarithmic axis capped at 150.

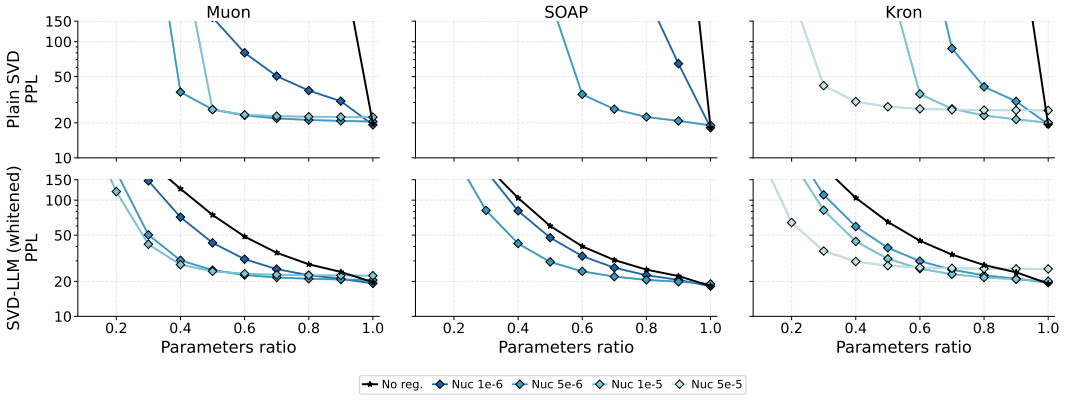

Figure 7: SLORR-Nuc with different optimizers on Llama 135M. Layout and axes follow Figure 6. Blue curves indicate regularization strengths listed in the legend; black curves show matched unregularized baselines.

2025), and Kron. For each optimizer, we keep settings identical, and compare SLORR-Hoyer and SLORR-Nuc against a no-regularization run trained with the same optimizer.

For Muon, we apply Muon to two-dimensional weight matrices and use its auxiliary AdamW update for the remaining parameters, with Muon learning rate 10^{-2}, auxiliary AdamW learning rate 10^{-3}, weight decay 0.1, auxiliary AdamW betas $(0.9, 0.95)$, epsilon 10^{-8}, momentum 0.95, Nesterov momentum, and five Newton–Schulz iterations. For SOAP, we use learning rate 3×10^{-3}, weight decay 0.1, betas $(0.95, 0.95)$, epsilon 10^{-8}, and update the preconditioner every 40 steps. For Kron, we use learning rate 3×10^{-4}, weight decay 0.3, first-moment coefficient 0.9, preconditioner learning rate 0.1, merged dimensions, and maximum triangular preconditioner size 8192. For all methods, given the unfamiliarity with these optimizers, we performed small preliminary hyperparameter sweeps for each optimizer (not for SLORR hyperparameters) on unregularized models, in order to obtain reasonable hyperparameters, particularly learning rates. Results are shown in Figures 6 and 7. We find that SLORR produces more compressible models when compared against a matched unregularized run on the same optimizer. This suggests that even when a "low-rank inducing gradient $\rightarrow$ low-rank inducing parameter update" implication is not obvious in theory, SLORR appears to still work well in practice.

L Additional LLM Results

We report numerical perplexity results for the 135M models, followed by per-task downstream figures and uncapped perplexity figures for the 560M and 1B models. In the tables, rows identify training configurations and columns denote the retained parameter ratio in transformer blocks. For Q3R, the three arguments denote regularization strength, target-rank fraction, and refresh interval.

L.1 Llama 135M Results

L.1.1 Compression Results

Table 13: Llama 135M – Plain SVD – FineWeb-Edu perplexity

Method / setting	0.100	0.200	0.300	0.400	0.500	0.600	0.700	0.800	0.900	1.000
No reg.	148924.22	96542.02	94756.84	48167.34	6204.80	745.65	120.20	43.56	30.31	20.03
Hoyer (0.000005)	82613.87	89934.46	121806.95	10169.25	620.10	62.31	30.94	24.71	22.56	20.29
Hoyer (0.00001)	244959.88	126294.20	17418.12	2114.17	101.18	31.82	24.55	22.72	21.73	20.79
Hoyer (0.00005)	19597.55	4195.52	144.97	30.38	25.98	24.73	24.17	23.85	23.68	23.51
Nuc (0.000001)	193206.62	44012.09	56673.54	54392.27	6163.55	847.33	92.90	37.36	27.96	19.86
Nuc (0.00001)	44872.24	11712.02	2134.13	188.28	52.48	30.65	24.61	22.26	21.15	20.04
Nuc (0.00005)	1298.88	506.69	137.23	45.22	28.83	26.08	25.19	24.83	24.71	24.65
LoRITa(0.05)	107994.66	70145.65	40062.16	4174.68	80.74	34.04	25.77	22.92	21.55	20.10
LoRITa(0.3)	175884.08	14755.20	11336.23	331.24	53.26	27.43	23.28	21.54	**20.76**	20.03
LoRITa(1.0)	2221.62	393.78	76.30	35.53	25.34	**22.24**	**21.30**	**20.93**	20.77	20.68
LoRITa(2.0)	**383.73**	**59.77**	**31.09**	**25.11**	**23.68**	23.21	23.07	23.03	23.02	23.01
Q3R(0.5, 0.1, 300)	117083.30	13169.51	1639.50	234.04	71.83	38.84	28.54	24.49	22.46	**19.52**
Q3R(3.0, 0.05, 5)	1439.79	221.05	41.26	28.97	25.30	23.71	22.82	22.25	21.90	21.28
Q3R(3.0, 0.1, 300)	3964.43	994.16	178.96	40.28	25.02	22.59	21.84	21.45	21.21	20.85
Q3R(3.0, 0.2, 5)	12383.82	4452.66	900.59	67.71	32.54	25.69	23.14	22.18	21.73	21.19

Table 14: Llama 135M – SVD-LLM (whitened) – FineWeb-Edu perplexity

Method / setting	0.100	0.200	0.300	0.400	0.500	0.600	0.700	0.800	0.900	1.000
No reg.	1860.53	508.60	185.68	90.66	52.75	36.50	29.00	25.01	22.86	20.03
Hoyer (0.000005)	4931.33	659.68	158.93	66.69	38.35	28.79	24.71	22.65	21.52	20.29
Hoyer (0.00001)	4820.77	586.07	125.66	51.09	31.66	25.63	23.13	21.93	21.33	20.79
Hoyer (0.00005)	1855.93	123.38	38.59	27.87	25.40	24.45	24.00	23.77	23.64	23.51
Nuc (0.000001)	2787.25	543.44	189.16	81.81	47.34	33.54	27.41	24.11	22.25	19.86
Nuc (0.00001)	3585.01	541.55	138.51	54.63	33.75	26.40	23.24	21.68	20.87	20.04
Nuc (0.00005)	949.18	135.59	53.13	32.35	27.31	25.72	25.11	24.83	24.72	24.65
LoRITa(0.05)	4492.30	520.58	125.99	53.08	32.24	25.76	23.00	21.58	20.84	20.10
LoRITa(0.3)	1988.42	559.27	134.18	46.89	29.17	24.28	22.12	21.05	**20.52**	20.03
LoRITa(1.0)	267.73	84.63	40.83	28.19	23.66	**21.92**	**21.20**	**20.90**	20.77	20.68
LoRITa(2.0)	**130.50**	**46.14**	**28.92**	**24.71**	23.55	23.18	23.06	23.02	23.01	23.01
Q3R(0.5, 0.1, 300)	9151.60	542.18	140.16	59.41	35.94	27.77	24.10	22.18	21.08	**19.52**
Q3R(3.0, 0.05, 5)	324.44	67.89	33.93	26.88	24.51	23.27	22.54	22.07	21.77	21.28
Q3R(3.0, 0.1, 300)	833.56	128.92	42.33	27.32	**23.53**	22.17	21.64	21.34	21.15	20.85
Q3R(3.0, 0.2, 5)	709.47	127.72	54.27	34.24	27.25	24.24	22.76	22.02	21.65	21.19

L.2 Per-Task Downstream Results

Figures 8 to 11 report the individual task accuracies used for Figure 4, using the same runs. Evaluation details are in Appendix I. Parameter ratios refer to transformer blocks.

L.3 Full-Range Perplexity Results

Figure 12 shows the perplexities from the main text comparison without the 150 limit, with ratios from 0.1 to 1.0. We use a logarithmic y axis.

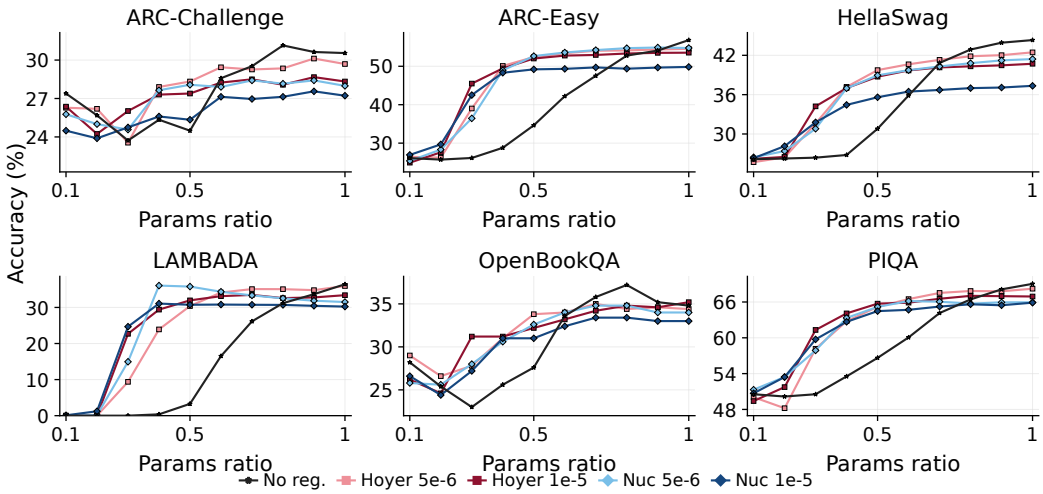

Figure 8: Llama 560M: zero-shot evaluation after Plain SVD compression.

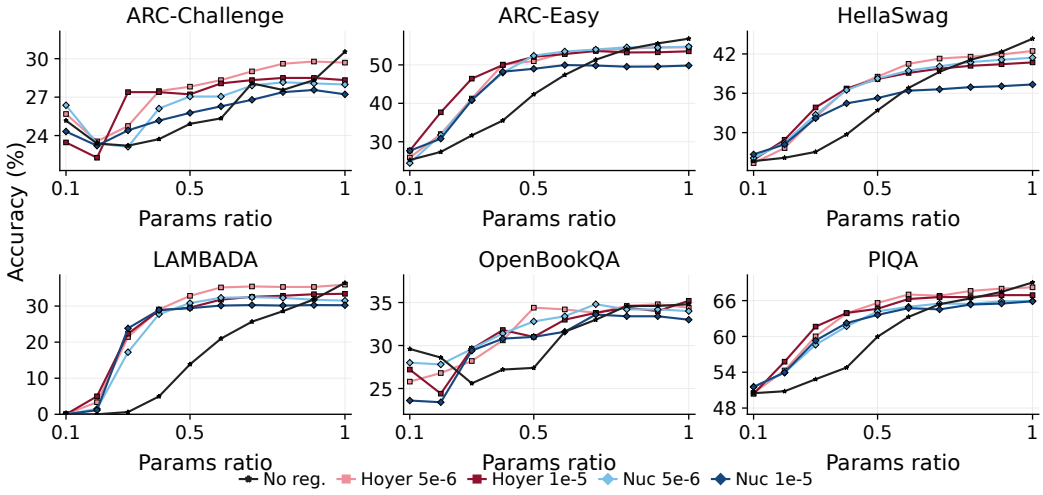

Figure 9: Llama 560M: zero-shot evaluation after SVD-LLM (whitened) compression.

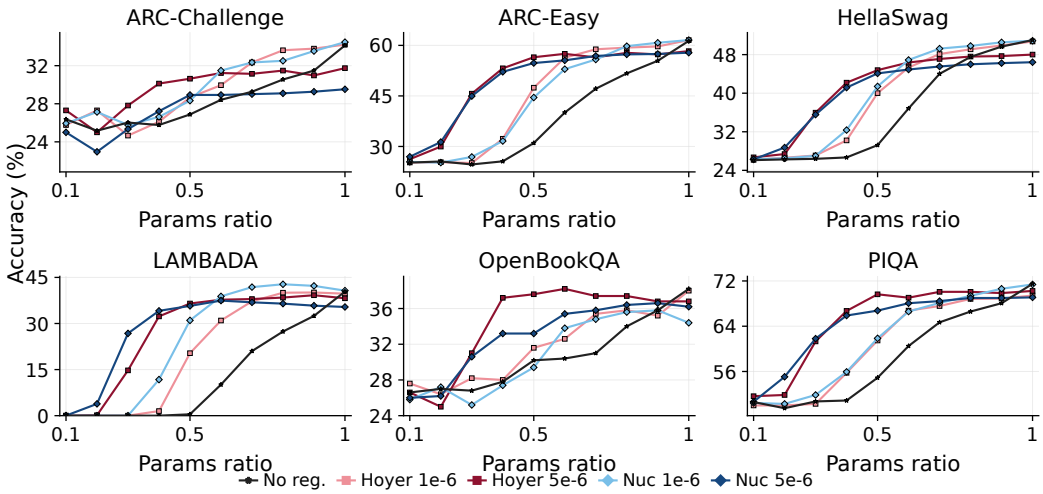

Figure 10: Llama 1B: zero-shot evaluation after Plain SVD compression.

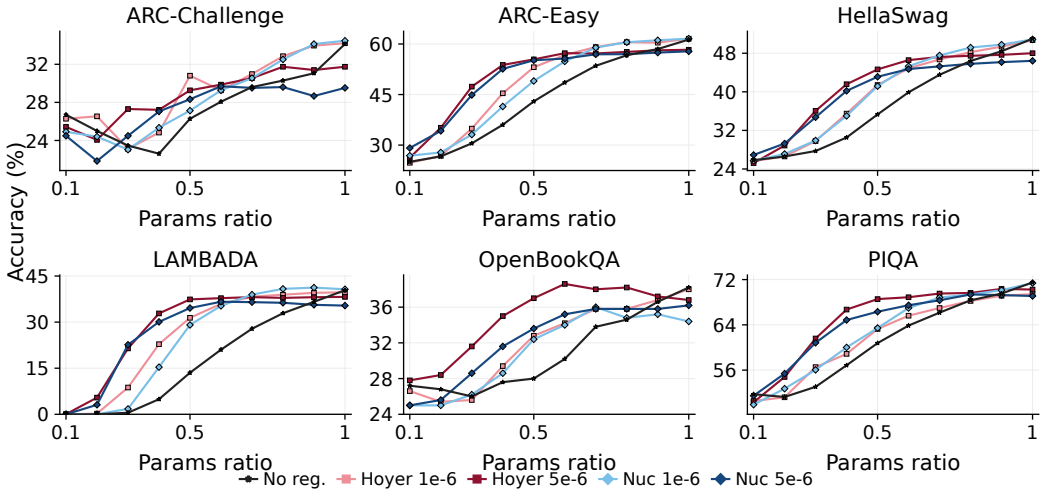

Figure 11: Llama 1B: zero-shot evaluation after SVD-LLM (whitened) compression.

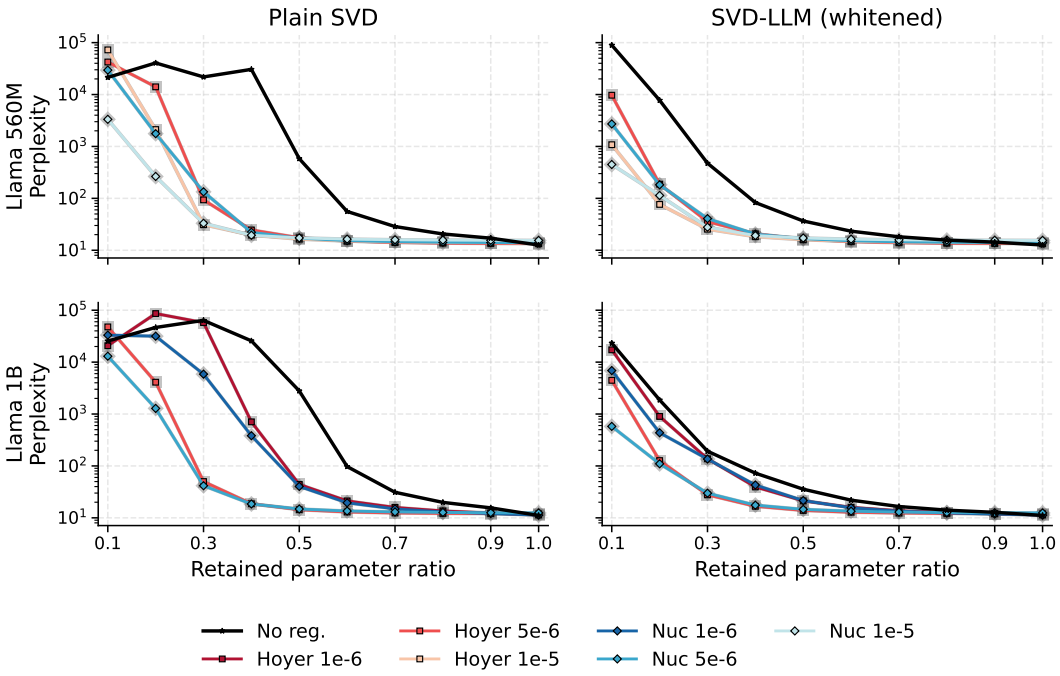

Figure 12: **Uncapped FineWeb-Edu perplexity for Llama 560M and 1B.** All panels use the same logarithmic perplexity scale.

M Overhead on LLM training

To provide a more realistic benchmark, we also measure overhead under several implementation choices. Specifically, we benchmark `OLMoLlamaBlock`, which uses separate Q, K, and V projections, with both manual attention and fused attention; we add the fused attention variant by modifying the block. We also benchmark `OLMoSequentialBlock`, which uses a single shared QKV projection and only uses fused attention. Here, fused attention refers to `torch.nn.functional.scaled_dot_product_attention`.

For each model/configuration, we run four matched pairs of jobs. Each pair consists of two independent training runs with identical model size, attention implementation, block type, microbatch size, data, and benchmark window. One baseline is run without regularization and one run with regularization. Within a pair, the two runs are performed on the exact same node, as we observed slight overall performance variations per node. For these, we use autotuning in the Polar Express compilation step, which we observed to yield slight gains in terms of overhead (unlike in the vision benchmarks).

Each individual run measures the wall-clock time for steps 101–700. The 135M experiments use 4 H100 GPUs, and the 560M experiments use 8 H100 GPUs, following our main experiments.

We aggregate results by first computing the overhead for each matched pair,

$$100 \times \left(\frac{T_{\mathrm{reg},i}}{T_{\mathrm{base},i}} - 1 \right),$$

where $T_{\mathrm{reg},i}$ and $T_{\mathrm{base},i}$ are the 600-step wall times for pair i. We report the mean and standard deviation of these paired overheads. Seconds and tokens/sec are reported as arithmetic means over the corresponding baseline or regularized runs. Peak memory is computed similarly, and was stable across runs. The reported quantities are per device, and the peak memory is taken as the maximum over all GPUs.

Results are shown in Table 15. MBS denotes the microbatch size per GPU. Note that, for the sake of fairness, it is adjusted for fused attention runs because each sequence consumes less memory.

On average, overhead remains below 1% across the different settings, although individual runs can slightly exceed 1%. Peak memory overhead is negligible in these benchmarks.

Table 15: LLM overhead benchmark summary.

Size	Attention	Block	MBS	Reg.	Seconds	Tokens/sec	Overhead	Peak mem. (GB)
135M	Manual	Llama	32	no	735.031	106,992.99	0.000%	58.890
135M	Manual	Llama	32	yes	738.614	106,474.08	0.487% ± 0.042%	59.010
135M	SDPA	Llama	64	no	355.179	221,418.77	0.000%	56.810
135M	SDPA	Llama	64	yes	358.113	219,606.97	0.826% ± 0.275%	56.930
135M	SDPA	Sequential	64	no	345.173	227,837.61	0.000%	56.780
135M	SDPA	Sequential	64	yes	347.225	226,491.57	0.594% ± 0.122%	56.930
560M	Manual	Llama	12	no	1,267.318	31,027.41	0.000%	72.880
560M	Manual	Llama	12	yes	1,276.090	30,814.13	0.692% ± 0.003%	72.890
560M	SDPA	Llama	24	no	616.515	63,780.88	0.000%	61.010
560M	SDPA	Llama	24	yes	622.627	63,154.74	0.991% ± 0.014%	61.020
560M	SDPA	Sequential	24	no	596.743	65,894.00	0.000%	61.030
560M	SDPA	Sequential	24	yes	602.191	65,297.77	0.913% ± 0.079%	61.040

M.1 Overhead Against Other Baselines at Larger Scales

We additionally benchmark memory and runtime overhead at the 1B and 3B scales on eight NVIDIA A100-80GB GPUs, where overhead differences are more visible. All methods are otherwise reasonably efficient at the smaller scales. For Q3R, this is partially because the regularization computational load is distributed across the GPUs for fairness. Table 16 reports peak memory and average throughput. MBS denotes the MBS of the method itself. The baseline uses the largest power of two MBS that fits. This is to account for the overhead coming from having to lower the MBS. These are measured on equivalent GPUs.

Table 16: LLM training overhead at larger scales. "Base" denotes the unregularized baseline at the same MBS. At 1B, SLORR supports MBS 16 with small runtime overhead, while Q3R and LoRITa run out of memory at the same MBS. At 3B, SLORR runs at MBS 1 and 2, while Q3R and LoRITa run out of memory at both settings.

Scale	Method / Setting	MBS	Peak Mem.	Base Mem.	Base tok/s	Method tok/s	Overhead
1B	SLORR-Hoyer	16	71.23 GB	70.55 GB	18,871	18,618	+1.4%
1B	Q3R (refresh=300)	8	54.67 GB	46.28 GB	18,871	17,321	+8.9%
1B	Q3R (refresh=50)	8	54.67 GB	46.28 GB	18,871	16,770	+12.5%
1B	Q3R (refresh=5)	8	54.67 GB	46.28 GB	18,871	9,872	+91.2%
1B	LoRITa	8	58.09 GB	46.28 GB	18,871	15,966	+18.2%
1B	Q3R / LoRITa	16	OOM	70.55 GB	18,871	OOM	OOM
3B	SLORR-Hoyer	1	74.80 GB	73.25 GB	5,369	5,272	+1.8%
3B	SLORR-Hoyer	2	79.98 GB	77.78 GB	6,324	6,154	+2.8%
3B	Q3R / LoRITa	1–2	OOM	73.25–77.78 GB	5,369–6,324	OOM	OOM

N ALL LLAMA 135M SINGULAR VALUES

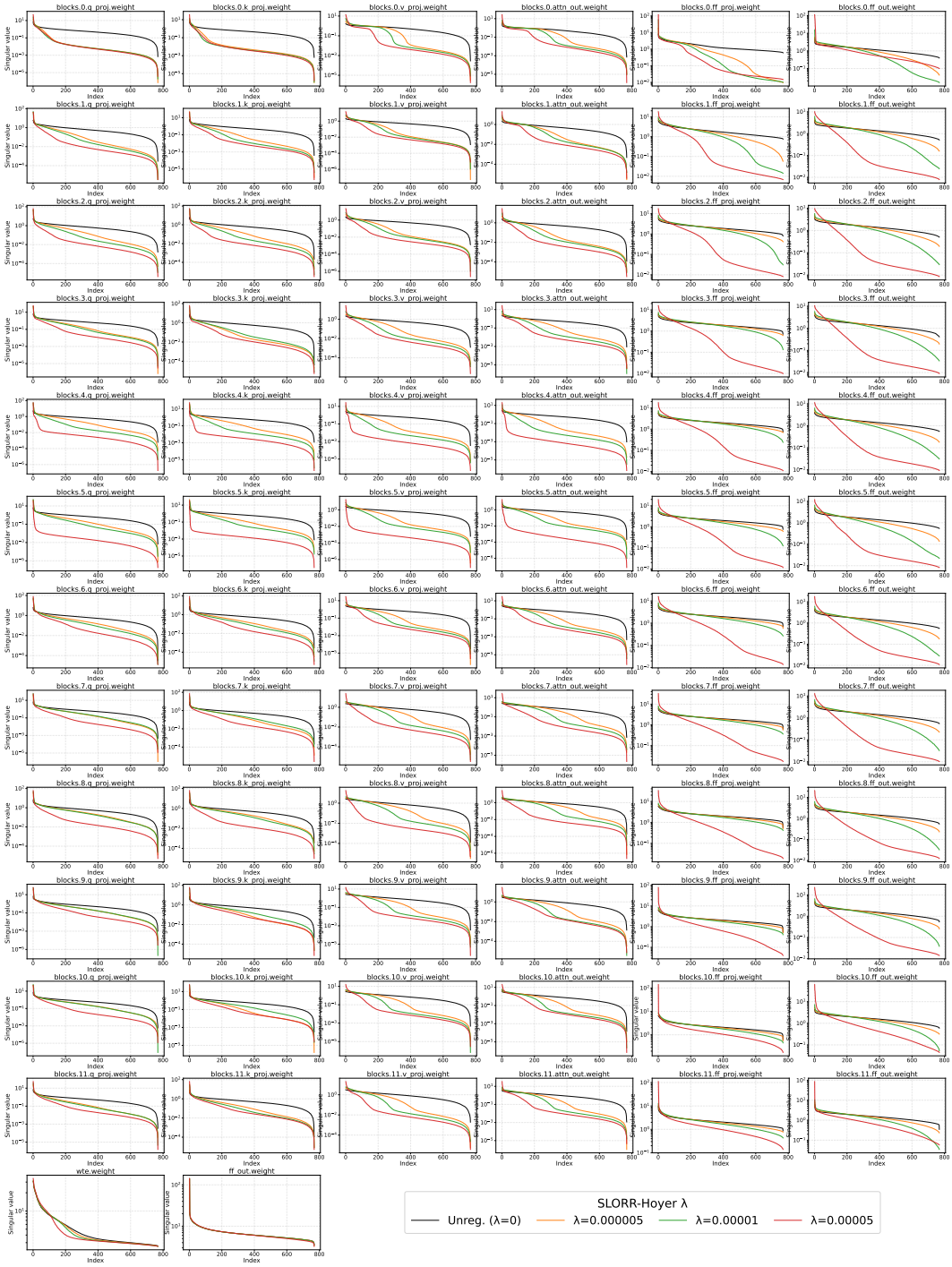

Figure 13: Singular value spectra for all layers of Llama 135M. The different curves correspond to different values of λ. As noted in the main text, only layers inside transformer blocks are regularized, but we also include weight embeddings and the final projection layer for completeness.