DLR: ZERO-INFERENCE-COST LATENT RESIDUALS FOR LOW-RANK PRE-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models have driven recent progress in language and multimodal AI, yet pre-training them at scale is prohibitively expensive. Low-rank pre-training, which factorizes each weight matrix into a rank-r product to reduce both parameters and FLOPs, is a promising response but typically lags full-rank training in quality. We propose Duplicated Latent Residual (DLR), a training-only, parameter-free, foldable plug-in for low-rank pre-training. DLR augments the standard low-rank output Bz with a fixed structured residual $\frac{\alpha}{\sqrt{K}}\,\mathrm{Expand}_K(z)$ that replicates each latent coordinate $K{=}\lceil d_{\mathrm{out}}/r\rceil$ times across the output. With α fixed, DLR adds zero learnable parameters per layer; after training, it is absorbed into the up-projection in closed form, $B^\star = B + \frac{\alpha}{\sqrt{K}}R^\top$, so deployment parameter count, FLOPs and memory match the underlying low-rank backbone exactly. Across LLaMA models from 60M to 7B parameters, DLR strengthens low-rank pre-training on C4 validation perplexity in most settings, with the clearest gains at 130M and above; folded checkpoints transfer cleanly to supervised fine-tuning on standard benchmarks.

1 INTRODUCTION

Large language models (LLMs) have transformed machine learning across language, vision, and multimodal domains, yet this success comes at an unsustainable cost. Pre-training modern LLMs requires vast compute, memory, and energy resources, with models like LLaMA-3 (Grattafiori et al., 2024), Qwen-3 (Yang et al., 2025), DeepSeek-V3 (Liu et al., 2024), and GPT-4 (OpenAI et al., 2024) consuming thousands of GPU-months. This has spurred significant research into efficient pre-training methods that preserve model quality while reducing training cost.

Existing efficient pre-training methods take three forms: extending LoRA-style low-rank updates to pre-training (Lialin et al., 2023; Zhou et al., 2025; Mo et al., 2025), projecting gradients into low-rank subspaces to compress optimizer states (Zhao et al., 2024; Chen et al., 2024), and modifying the network structure with low-rank parameterization possibly augmented by sparse compensation or latent-space mixing (Han et al., 2024; Li et al., 2025; Liu et al., 2025; Zhang et al., 2025). The third line is the most direct route to reducing both trainable parameters and forward FLOPs, but many additional pathways introduced to compensate for the rank-r bottleneck (sparse residuals, channel-sparse compensation, or inter-layer latent gates) *persist at inference*. Deploying these augmented variants therefore requires shipping a modified inference graph rather than the vanilla rank-r decoder, forfeiting one of low-rank pre-training's most attractive practical properties: drop-in compatibility with the existing low-rank inference stack.

A natural question arises: *can we improve the training dynamics of low-rank backbones without sacrificing their parameter and compute efficiency?*

We propose **Duplicated Latent Residual** (DLR), a *training-only*, *parameter-free*, *foldable* plug-in for low-rank pre-training. DLR augments the standard low-rank output Bz with a fixed structured residual that replicates each latent coordinate $K = \lceil d_{\mathrm{out}}/r\rceil$ times across the output dimension and rescales it by $\alpha/\sqrt{K}$, providing an additional gradient pathway that does not depend on the learned decoder (Fig. 1). With α held fixed, no learnable parameters are added per layer at training time. After training, the residual is absorbed into the up-projection in closed form via $B^\star = B + (\alpha/\sqrt{K})\,R^\top$, where $R \in \{0,1\}^{r\times d_{\mathrm{out}}}$ is the structured binary replication matrix induced by the duplication

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

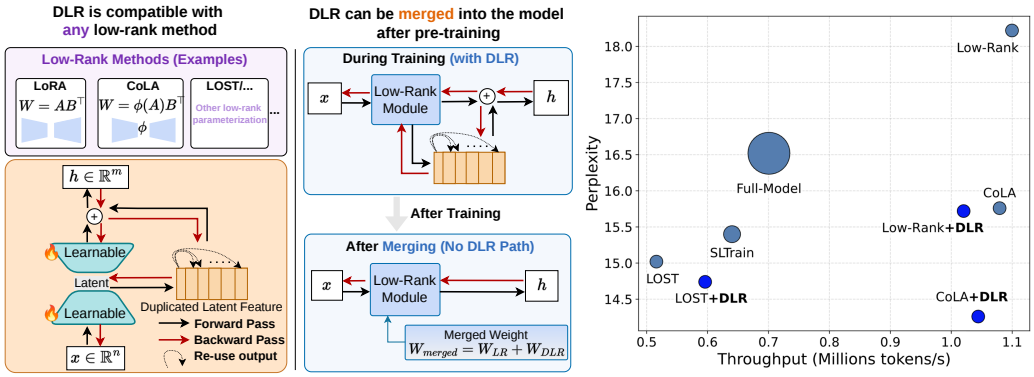


Figure 1: **Overview and perplexity–throughput trade-off.** (a) Duplicated Latent Residual (DLR) is a training-only residual branch for low-rank decoders that is removed by folding before inference. (b) DLR improves the perplexity–throughput trade-off at LLaMA-1B; full results are in Tab. 1.

($\mathrm{Expand}_K(z) = R^\top z$, see Eq. 10). The folded checkpoint runs on the underlying low-rank inference stack with the same parameter count, FLOPs, and memory footprint as the base backbone.

Empirically, DLR strengthens four representative low-rank backbones (linear Low-Rank, CoLA (Liu et al., 2025), LOST (Li et al., 2025), and LaX (Zhang et al., 2025)) in most settings across LLaMA pre-training scales from 60M to 7B using a single untuned configuration, with mixed results only at the smallest 60M scale; the DLR+LOST combination at 1B improves PPL from 15.02 to 14.74, and in our single-seed 7B comparison DLR+CoLA outperforms reported LOST baselines from 40K steps onward while using 2.3× less per-GPU memory (Tab. 1, Tab. 2). Folded checkpoints transfer cleanly to supervised fine-tuning on Alpaca-cleaned, where DLR+CoLA yields a slightly higher average accuracy among the three checkpoints we compare, despite deploying through the same inference graph as vanilla CoLA.

We summarize our contributions as follows:

- **Foldable training residual.** We give a closed-form identity, $B^\star = B + \frac{\alpha}{\sqrt{K}} R^\top$, that absorbs DLR into the up-projection of any low-rank parameterization that exposes a latent-to-output decoder, yielding the same deployed graph shape, parameter count, FLOPs, and memory footprint as the base method.
- **Parameter-free training overhead.** With $\alpha = 1$ fixed, DLR adds zero learnable parameters per layer. Its training-time forward cost is a single `repeat_interleave` plus add, with small overhead under `torch.compile` and zero inference overhead after folding.
- **Backbone-agnostic plug-in.** A single DLR configuration improves four low-rank backbones (linear Low-Rank, CoLA, LOST, LaX) across LLaMA pre-training scales from 130M to 7B on C4 perplexity, with mixed results at 60M (Tab. 1, Tab. 2); folded checkpoints transfer cleanly to Alpaca-cleaned SFT, where DLR + CoLA yields a slightly higher average accuracy among the compared 1B checkpoints (Tab. 4).

2 Low-Rank Efficient Pre-Training

2.1 Setup and a Unified View

Consider a linear map in a transformer block (bias omitted):

$$y = Wx, \quad W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}},\ x \in \mathbb{R}^{d_{\text{in}}}. \tag{1}$$

Many efficient pre-training methods can be expressed as

$$y = \underbrace{\Phi_{\text{LR}}(x;\theta)}_{\text{low-rank backbone}} + \underbrace{\Phi_{\text{EX}}(x;\psi)}_{\text{extra compensation}}, \tag{2}$$

where Φ_{LR} is a low-rank backbone (possibly nonlinear in latent space), and Φ_{EX} is an additional compensation pathway.

We use a shared notation throughout. Let $r \ll \min(d_{\mathrm{in}}, d_{\mathrm{out}})$ be the rank, $A \in \mathbb{R}^{d_{\mathrm{in}} \times r}$ be the down-projection, $B \in \mathbb{R}^{d_{\mathrm{out}} \times r}$ be the up-projection, and define the latent

$$z \;=\; s \cdot \phi(A^\top x) \in \mathbb{R}^r, \tag{3}$$

where $\phi(\cdot)$ is an optional activation and s is a scalar scaling (fixed or learnable). Unless stated otherwise, we focus on *low-rank pre-training* where the linear map is parameterized by (A, B) (rather than an adapter added on top of a separate frozen base weight).

This unified decomposition allows us to compare seemingly different efficient pre-training methods through the lens of where and how they introduce additional learning signals beyond a rank-r backbone. In particular, it makes clear whether improvements stem from modifying the latent representation or from adding new weight-space pathways.

2.2 Baselines in the Unified Notation

Low-rank (linear) (Hu et al., 2021). The linear low-rank parameterization replaces W by $BA^\top$:

$$\Phi_{\mathrm{LR}}(x) = B(A^\top x), \qquad \Phi_{\mathrm{EX}}(x) \equiv 0. \tag{4}$$

This cuts multiplies from $d_{\mathrm{out}} d_{\mathrm{in}}$ to $r(d_{\mathrm{out}}+d_{\mathrm{in}})$ and reduces optimizer states to the size of (A, B), but the fixed rank often underperforms full-rank pre-training (Kamalakara et al., 2022; Khodak et al., 2021).

SLTrain (low-rank + element-wise sparse residual) (Han et al., 2024). SLTrain augments the low-rank factorization with a fixed-support sparse residual:

$$W = BA^\top + S, \qquad \mathrm{supp}(S) = \Omega,\ |\Omega| = k, \tag{5}$$

where $\Omega \subseteq [d_{\mathrm{out}}] \times [d_{\mathrm{in}}]$ is sampled once at initialization. Equivalently,

$$\Phi_{\mathrm{LR}}(x) = B(A^\top x), \qquad \Phi_{\mathrm{EX}}(x) = Sx. \tag{6}$$

Thus SLTrain improves expressivity through an explicit sparse weight-space path, unlike DLR's fixed latent-space residual that is folded into B.

LOST (low-rank + channel-sparse compensation) (Li et al., 2025). LOST combines a low-rank branch with an SVD-guided channel-sparse component intended to cover complementary directions.

$$\Phi_{\mathrm{LR}}(x) = Bz, \qquad \Phi_{\mathrm{EX}}(x) = W_s P_I x, \tag{7}$$

where $P_I \in \{0,1\}^{k \times d_{\mathrm{in}}}$ selects a subset I of k input channels, so $P_I x \in \mathbb{R}^k$ contains exactly the channels feeding the learned compensation matrix $W_s \in \mathbb{R}^{d_{\mathrm{out}} \times k}$; equivalently, the columns of W_s correspond to the P_I-selected channels. The latent z is defined in Eq. 3. The compensation path is learned and remains part of the deployed graph, whereas DLR adds a fixed latent residual during training and folds it into B.

CoLA (nonlinear latent low-rank) (Liu et al., 2025). CoLA keeps a pure low-rank decoder but makes the latent representation nonlinear:

$$\Phi_{\mathrm{LR}}(x) = Bz, \qquad z = s \cdot \phi(A^\top x), \qquad \Phi_{\mathrm{EX}}(x) \equiv 0. \tag{8}$$

It enriches trainability through latent gating without adding a separate weight-space compensation path, making it a natural backbone for DLR.

LaX (inter-layer latent residual) (Zhang et al., 2025). LaX improves low-rank training by introducing an *inter-layer* residual in latent space. For a layer latent $z_i = s \cdot \phi(A_i^\top x_i)$, it forms

$$\tilde{z}_i = z_i + G(z_{i-1}), \qquad y_i = \mathrm{LN}(B_i \tilde{z}_i), \tag{9}$$

where G is a lightweight alignment gate and the output LayerNorm follows the original formulation. In the unified view, LaX keeps the same low-rank decoder form but adds information flow from the previous layer's latent, improving trainability without increasing the target rank r.

Taken together, these methods illustrate two dominant strategies for improving low-rank pre-training: (i) adding auxiliary weight-space pathways to compensate for rank deficiency (*e.g.*, SLTrain, LOST), or (ii) enriching the latent representation while keeping a single low-rank decoder (*e.g.*, CoLA, LaX). DLR is most naturally described as a latent-space plug-in, but it can attach to any backbone that exposes a low-rank latent-to-output decoder Bz, including the low-rank branch of hybrid methods such as LOST. It therefore targets training dynamics through a fixed, non-learned expansion without tying the design to one particular low-rank backbone.

3 Duplicated Latent Residual (DLR)

Key idea. Given an existing low-rank layer with latent $z \in \mathbb{R}^r$ and output Bz, DLR attaches an *intra-layer latent-space* residual that expands the same latent to output width with a *fixed structured decoder* as shown in Fig. 1. Let $K = \lceil d_{\text{out}}/r \rceil$ and define an expansion operator

$$\text{Expand}_K : \mathbb{R}^r \to \mathbb{R}^{d_{\text{out}}},\ [\text{Expand}_K(z)]_i = z_{\lfloor i/K \rfloor}, \tag{10}$$

with the last block truncated if $rK > d_{\text{out}}$. Equivalently, $\text{Expand}_K(z) = R^\top z$ for a fixed binary replication matrix $R \in \{0,1\}^{r \times d_{\text{out}}}$ with $R_{j,i} = \mathbb{1}\{j = \lfloor i/K \rfloor\}$.

The plug-in form used during training is:

$$y \;=\; Bz \;+\; \frac{\alpha}{\sqrt{K}} \cdot \text{Expand}_K(z) \;+\; b, \tag{11}$$

where $\alpha \in \mathbb{R}$ controls the residual strength (fixed in all experiments). Duplicated Latent Residual (DLR) therefore augments, rather than replaces, a standard low-rank layer: the backbone still supplies A, B, the activation, and any other latent-space structure, while DLR contributes only the fixed residual term $\frac{\alpha}{\sqrt{K}} \text{Expand}_K(z)$ during training. Intuitively, DLR views the output space as partially redundant and leverages structured replication to expose each latent coordinate to multiple output channels. Unlike sparse or channel-selective compensation, the resulting residual is dense, fixed, and highly structured, leading to uniform coverage of the output space without introducing irregular sparsity. As a result, DLR provides an additional learning signal that bypasses the conditioning of the learned decoder, while retaining the computational characteristics of the underlying low-rank backbone.

Foldability: zero inference overhead. A defining property of DLR is that, after training, the duplicated-residual branch can be *exactly* absorbed into the up-projection B in closed form, leaving the inference graph identical to the underlying low-rank decoder. Recall from Eq. 10 that $R \in \{0,1\}^{r \times d_{\text{out}}}$ with $\text{Expand}_K(z) = R^\top z$, so $R^\top \in \{0,1\}^{d_{\text{out}} \times r}$ has the same shape as B (each column of $R^\top$ is the indicator of one latent group). Substituting into Eq. 11 and collecting linear terms in z yields the algebraic identity

$$y \;=\; Bz \;+\; \tfrac{\alpha}{\sqrt{K}} R^\top z \;+\; b \;=\; \underbrace{\left(B + \tfrac{\alpha}{\sqrt{K}} R^\top\right)}_{B^\star} z \;+\; b \;=\; B^\star z \;+\; b, \tag{12}$$

where $B^\star \in \mathbb{R}^{d_{\text{out}} \times r}$ has *the same shape* as the original up-projection B. Foldability follows by construction: the operation $B \leftarrow B + (\alpha/\sqrt{K})\, R^\top$ is performed once at training termination, and the resulting checkpoint is a drop-in replacement that runs on any code path implementing standard rank-r inference (the Expand_K branch, the buffer storing R, and the residual scale α are all discarded). Hence the deployment artifact has the same graph structure, parameter count, FLOPs, and peak-memory footprint as the corresponding low-rank baseline, while still benefiting from the better-trained $B^\star$. This contrasts with prior residual or auxiliary-pathway approaches over low-rank backbones (e.g., LoR2C (Zhao et al., 2025), ResLoRA (Wang et al., 2025b), sparse-additive variants such as SLTrain (Han et al., 2024), or inter-layer latent residuals such as LaX (Zhang et al., 2025)): they retain auxiliary parameters or non-mergeable nonlinearities at inference time and therefore modify the deployed graph. Appendix G further situates DLR among residual-style methods and clarifies why foldability differs from mergeable adapter residuals (Tab. 11). A reference `fold()` implementation in $\sim$10 lines is provided in Appendix A; the operation is the only DLR-specific code that runs in the deployment pipeline, after which inference is indistinguishable from a vanilla rank-r decoder.

Variance-preserving correction. A naive duplication would amplify the residual energy by a factor of K_j in the j-th block. We therefore use the global scaling $\beta = \alpha/\sqrt{K}$ in Eq. 11. When $d_{\text{out}} = rK$ (all blocks have size K), this choice preserves per-latent energy: $\|\beta \operatorname{Expand}_K(z)\|_2^2 = \alpha^2\|z\|_2^2$. Equivalently, if $\operatorname{Var}(z_j) = \sigma_z^2$, then each duplicated coordinate has variance $\operatorname{Var}(\beta z_j) = \alpha^2\sigma_z^2/K$, so the *total* variance mass assigned to the block remains $\alpha^2\sigma_z^2$. If the last block is truncated ($K_r < K$), the same scaling yields a slightly smaller energy contribution for that block, which is benign and avoids data- or shape-dependent re-scaling. This correction mirrors the variance-preserving "repair" principle used in folding/merging operations (Wang et al., 2025a; Saukh et al., 2026) and is essential for stable pre-training (Jordan et al., 2022).

Duplication map. DLR employs a *deterministic, uniform* duplication structure. The expansion operator Expand_K partitions the d_{out} output coordinates into r consecutive blocks of size K (with the last block possibly truncated to $d_{\text{out}} - (r-1)K$ elements). Each block $\mathcal{G}_j = \{jK, jK{+}1, \ldots, \min((j{+}1)K{-}1, d_{\text{out}}{-}1)\}$ receives a copy of the latent coordinate z_j. Formally, the replication matrix $R \in \{0,1\}^{r \times d_{\text{out}}}$ has row j containing ones at columns $\mathcal{G}_j$ and zeros elsewhere. This fixed, balanced structure ensures that each latent dimension contributes equally to the residual pathway and requires no hyperparameter tuning beyond the rank r. We implement $\operatorname{Expand}_K(z)$ as a single `repeat_interleave` followed by truncation; the *contiguous* form is our hardware-efficient choice, since more complex fixed maps (*e.g.*, permuted masks or non-contiguous/grouped duplication) force indexed gather/scatter operations (e.g., `index_select`) that cannot be fused as cleanly. We quantify this implementation choice via a random-duplication ablation in Sec. 4.

Design rationale. DLR is built around two deliberately simple choices: a fixed latent-to-output duplication map and a shape-dependent scale $\beta = \alpha/\sqrt{K}$. The scale prevents the residual branch from growing merely because each latent coordinate is copied multiple times. For the group $\mathcal{G}_j$ that receives copies of z_j, the duplicated residual block satisfies

$$\|\beta(R^\top z)_{\mathcal{G}_j}\|_2^2 = \beta^2 K_j z_j^2 = \alpha^2 \frac{K_j}{K} z_j^2,$$

so when the group is full ($K_j = K$) and $\alpha = 1$, the residual assigns the block the same squared energy as the original latent coordinate. This normalization is purely shape-based: it keeps the forward residual on the same scale as z without introducing data-dependent statistics, per-layer tuning, or additional learned parameters.

The same fixed branch also changes the backward path in a controlled way. Let $g_y = \partial\mathcal{L}/\partial y \in \mathbb{R}^{d_{\text{out}}}$. Differentiating Eq. 11 with respect to the latent variable gives

$$g_z \;=\; \frac{\partial \mathcal{L}}{\partial z} \;=\; B^\top g_y \;+\; \beta R g_y. \tag{13}$$

The first term is the usual gradient route through the learned decoder B, while the second term is a fixed, decoder-independent route that aggregates output gradients according to the same groups used in the forward duplication. This is the mechanism we intend DLR to provide: during training, the encoder A receives an additional calibrated signal that does not have to pass through the current state of B; after training, the entire route is folded into $B^\star$ by Eq. 12. The effectiveness of this design is evaluated empirically in the gradient-path measurements of Sec. 4, where the residual-induced component is strong early in training and remains nearly orthogonal to the learned-decoder component, and in the variance-correction ablation of Tab. 7.

Computational complexity. DLR does not replace the low-rank backbone; it adds a training-time residual branch on top of the backbone's existing latent z and up-projection B. For any backbone that outputs Bz, the only extra forward work during training is the structured expansion-and-add term $\frac{\alpha}{\sqrt{K}} R^\top z$ (equivalently $\frac{\alpha}{\sqrt{K}} \operatorname{Expand}_K(z)$), implemented as `repeat_interleave` followed by truncation (Appendix A, Listing 1). Crucially, the plug-in introduces no extra large GEMMs, no learned matrices, and no sparse/indexed matrix multiplications as in Eq. 5–Eq. 7; under `torch.compile`, our profiler traces do not show the expansion as a separate `aten::repeat_interleave` kernel, suggesting fusion into surrounding computation. After folding, even this residual branch is removed from the inference graph; end-to-end throughput and peak memory are reported in Sec. 4 (Tab. 3).

4 EXPERIMENTS

4.1 EVALUATION SETUP AND PROTOCOL

Pre-training data. We pre-train all models on the Colossal Clean Crawled Corpus (C4) (Raffel et al., 2020), a cleaned and deduplicated snapshot of Common Crawl that is widely used for language modeling.

Model family and token budgets. Following the experimental protocols used by Han et al. (2024), Glentis et al. (2025), and Li et al. (2025), we adopt the same LLaMA-style architecture family and match the token budget for each model scale. Concretely, we pre-train models with 60M, 130M, 350M, 1B, and 7B parameters (Touvron et al., 2023b;a), using the same number of training tokens as prior work at each size (see Tab. 1). Training details are provided in Appendix A.

Baselines. We compare against *Full-Rank* training and representative efficient pre-training methods, including *Low-rank* (Hu et al., 2021) *GaLore* (Zhao et al., 2024), *Fira* (Chen et al., 2024), *LORO* (Mo et al., 2025), *SLTrain* (Han et al., 2024), *LOST* (Li et al., 2025), *CoLA* (Liu et al., 2025), and *LaX* (Zhang et al., 2025),

Unless otherwise stated, all baselines use the authors' recommended hyperparameters and the same optimizer/schedule.

Plugging DLR into LLaMA backbones. For each low-rank LLaMA backbone, we attach the DLR residual to every low-rank linear projection in the Transformer blocks, including self-attention projections (W_Q, W_K, W_V, W_O) and the MLP/FFN projections. The backbone still determines the learned matrices, rank, activation, and any inter-layer latent structure. DLR only adds the fixed duplicated-latent branch during training and folds it into the corresponding up-projection before deployment (Sec. 3).

To ensure a fair comparison, we use the same AdamW optimizer and closely follow the training recipe in prior work, including learning-rate schedule, warmup ratio, and packed-data training, whenever applicable (Zhao et al., 2024; Glentis et al., 2025). Additional implementation details (including rank settings, scaling choices, and compilation/profiling configurations) are provided in Appendix A.

4.2 END-TO-END PERFORMANCE

Main results. We focus on the trade-off between validation perplexity (PPL) and training efficiency under matched token budgets. Improvements are meaningful only if they preserve the throughput and memory profile of the underlying low-rank backbone. Tab. 1 reports validation PPL on C4 across LLaMA scales 60M / 130M / 350M / 1B, together with parameter count and estimated memory, paired by backbone (Low-Rank, CoLA, LOST, Low-Rank+LaX, CoLA+LaX) so that each base method can be read off against its +DLR counterpart. A single untuned DLR configuration (α=1, fixed, uniform Expand_K) consistently improves all four backbones at 130M and above while keeping the same parameter budget as the corresponding backbone. In particular, CoLA+DLR achieves substantial PPL reductions over CoLA at fixed rank (*e.g.*, $15.76 \rightarrow 14.26$ at 1B; $25.61 \rightarrow 23.80$ at 130M), and DLR also improves the strong LOST baseline (*e.g.*, $15.02 \rightarrow 14.74$ at 1B), yielding the best perplexity–efficiency trade-off among the foldable plug-in variants we report.

The 60M scale shows a few sub-cells (LOST+DLR, Low-Rank+LaX+DLR, CoLA+LaX+DLR) where the +DLR row matches or slightly trails the corresponding base baseline. We attribute this to multi-seed noise at the smallest scale: with the lowest token budget in our study (1.4B vs. 13.1B at 1B) and the smallest hidden width, each backbone's run-to-run PPL spread is wider relative to the absolute PPL gap a single-layer plug-in can move. The ±std reported on the +DLR rows (0.05–0.18) is consistent with this regime, and the deviations vanish from 130M onward where the token-to-parameter ratio is larger. We therefore do not draw conclusions from the 60M sub-cells in isolation, and report 60M primarily as the small end of a 4-scale trajectory rather than as a stand-alone benchmark.

Scaling-up results at 7B are reported separately in Tab. 2, and downstream results after instruction fine-tuning are presented in Section 4.3 (Tab. 4). We additionally report pre-fine-tuning zero-shot evaluation for the 1B models in Appendix F (Tab. 10).

Table 1: **Main C4 validation results across LLaMA scales.** We report validation perplexity (PPL), parameter count (M), and estimated optimizer-state memory (GB) for 60M–1B models under matched token budgets. Rows are paired by backbone; **bold** marks the better PPL within each base/+DLR pair. DLR adds no learned parameters and folds into the base decoder, so each +DLR row has the same deployed parameter count, FLOPs, and memory footprint as its base row. Non-DLR baselines are reproduced by us or taken from prior work (Glentis et al., 2025; Li et al., 2025; Mo et al., 2025; Han et al., 2024; Zhang et al., 2025; Liu et al., 2025; Zhao et al., 2024; Chen et al., 2024); memory follows the optimizer-state estimate convention of (Zhao et al., 2024; Han et al., 2024).

	60M			**130M**			**350M**			**1B**		
r / d	128 / 512			256 / 768			256 / 1024			512 / 2048		
Tokens	1.4B			2.6B			7.8B			13.1B		
	PPL	Param	Mem	PPL	Param	Mem	PPL	Param	Mem	PPL	Param	Mem
Full-Model	30.27	58	0.35	23.13	134	0.81	18.76	368	2.21	13.40	1339	8.04
LoRA (Hu et al., 2021)	35.30	43	0.36	25.07	94	0.84	19.13	185	1.85	15.83	609	6.34
GaLore (Zhao et al., 2024)	34.58	58	0.28	25.31	134	0.61	19.37	368	1.59	15.57	1339	4.76
Fira (Chen et al., 2024)	30.34	58	0.28	22.96	134	0.61	16.82	368	1.59	15.10	1339	4.76
SLTrain (Han et al., 2024)	32.58	47	0.30	24.17	104	0.67	18.59	215	1.54	15.40	732	5.33
LORO (Mo et al., 2025)	33.87	43	0.24	24.78	94	0.57	19.66	185	1.11	15.53	609	3.66
Low-Rank	35.13	43	0.24	26.71	94	0.57	21.77	185	1.11	18.22	609	3.66
+**DLR**	**35.01**$_{\pm 0.18}$	43	0.24	**25.00**$_{\pm 0.35}$	94	0.57	**18.75**$_{\pm 0.03}$	185	1.11	**15.72**$_{\pm 0.74}$	609	3.66
CoLA (Liu et al., 2025)	34.10	43	0.24	25.61	94	0.57	19.75	185	1.11	15.76	609	3.66
+**DLR**	**32.96**$_{\pm 0.06}$	43	0.24	**23.80**$_{\pm 0.40}$	94	0.57	**18.38**$_{\pm 0.03}$	185	1.11	**14.26**$_{\pm 0.02}$	609	3.66
LOST (Li et al., 2025)	**32.25**	43	0.24	24.05	94	0.57	18.95	185	1.11	15.02	609	3.66
+**DLR**	33.15$_{\pm 0.12}$	43	0.24	**24.03**$_{\pm 0.03}$	94	0.57	**18.88**$_{\pm 0.02}$	185	1.11	**14.74**$_{\pm 0.01}$	609	3.66
Low-Rank+LaX (Zhang et al., 2025)	**33.54**	44	0.33	24.63	94	0.70	18.90	185	1.38	15.51	609	4.54
+**DLR**	33.71$_{\pm 0.05}$	44	0.33	**24.38**$_{\pm 0.02}$	94	0.70	**18.85**$_{\pm 0.01}$	185	1.38	**15.29**$_{\pm 0.48}$	609	4.54
CoLA+LaX (Zhang et al., 2025)	**33.21**	44	0.33	24.21	99	0.74	18.51	196	1.46	14.78	609	4.54
+**DLR**	34.89$_{\pm 0.07}$	44	0.33	**24.08**$_{\pm 0.07}$	99	0.74	**18.37**$_{\pm 0.02}$	196	1.46	**14.67**$_{\pm 0.00}$	609	4.54

Variance/scale correction ($1/\sqrt{K}$). The duplicated residual copies each latent coordinate into K output coordinates, so we scale it by $1/\sqrt{K}$ to keep the residual energy comparable across ranks and output widths. Ablating this correction consistently worsens validation perplexity for DLR+CoLA across scales, indicating that the normalization is important in our pre-training setting. Full results are reported in Appendix B (Tab. 7).

Effective-rank / target-rank sensitivity. We also test whether DLR remains useful under tighter rank budgets by reducing the default rank to $0.75r_0$ and $0.5r_0$. Perplexity degrades smoothly rather than catastrophically, indicating a predictable quality–efficiency trade-off rather than dependence on a single tuned rank. The full rank-sensitivity table is in Appendix B (Tab. 9).

Scaling-law token budget (1B). To check whether the 1B gains persist beyond the standard 13.1B-token budget, we additionally train the 1B setting for 26B tokens. The low-rank backbone equipped with DLR continues to closely track the full-rank trajectory throughout training, with a small but consistent convergence gap, matching the final ordering in Tab. 1. The full trajectory is shown in Appendix B (Fig. 2).

Gradient-path analysis (LLaMA-350M). To understand why DLR helps low-rank pre-training, we decompose the latent gradient into the learned-decoder path $B^\top g_y$ and the duplicated-residual path $\beta R g_y$, where $\beta = \alpha/\sqrt{K}$ (cf. Eq. 13). During LLaMA-350M pre-training, we track their relative magnitude $\rho(t) = \|\beta R g_y\|/\|B^\top g_y\|$ and directional alignment $\cos_t$ across representative projections (details in Appendix C). The residual-induced component is strongest early in training and then decays as the learned decoder takes over, while its cosine with the decoder path remains close to zero. This supports the view that DLR improves optimization by adding a calibrated, complementary gradient route rather than merely rescaling the existing low-rank update. A complementary initialization-offset ablation in Appendix B.2 separates this active training-time effect from simply initializing B with the folded block-constant offset.

Choosing the residual scale α. Our formulation allows at most one scalar α per layer to control the strength of the duplicated-latent residual in Eq. 11. In practice, however, we find that α is not a sensitive knob in our pre-training setting: fixing α=1.0 yields essentially identical training

Table 2: **LLaMA-7B pre-training on C4.** Validation perplexity at 10K–150K steps and steady-state per-GPU peak memory. Baselines are reported from (Han et al., 2024; Li et al., 2025; Glentis et al., 2025); DLR+CoLA uses the matched protocol described in this paper. "OOM" marks runs that cannot advance at the listed batch size; **bold** indicates the best PPL at each milestone.

Method	Batch	Mem (GB)	10K	40K	80K	120K	150K
Full-Rank Adam	4	49.53	24.95	20.05	—	OOM	—
8-bit SLTrain	8	60.91	27.59	—	OOM	—	—
LOST	8	62.15	**24.41**	16.48	14.01	12.93	12.80
DLR +CoLA (ours)	8	**27.03**	24.98	**15.82**	**13.71**	**12.69**	**12.08**

dynamics and final perplexity compared to making α learnable, and a coarse hyperparameter sweep consistently selects values near α=1.0 as optimal. We therefore use a fixed, non-trainable α=1.0 in all experiments, which keeps the plug-in parameter-free in practice.

The role of B and why we use a fixed contiguous duplication. We ablate two design choices on LLaMA-60M: whether the learnable decoder B remains necessary, and whether the fixed duplication must be implemented as a contiguous Expand_K. Removing B substantially degrades convergence, showing that DLR is a complement to the learned low-rank decoder rather than a replacement for it. Replacing contiguous duplication with a fixed random per-output mapping reaches similar loss but is much slower under `torch.compile`, because indexing-based gathers do not fuse as cleanly as `repeat_interleave`. Thus the default design keeps B, uses a fixed contiguous map, and remains both foldable at inference and cheap during training; details are in Appendix D.

Scaling to LLaMA-7B: DLR+CoLA enables low-memory pre-training at scale. To probe scaling beyond 1B, we attach DLR to a CoLA LLaMA-7B backbone and compare against Full-Rank Adam, 8-bit SLTrain, and LOST on the same C4 schedule (Tab. 2). Baselines are taken from prior work (Han et al., 2024; Li et al., 2025; Glentis et al., 2025). Our DLR+CoLA run uses the matched protocol described here (rank r=1024, batch size 8, BF16, Adam states, single seed). From 40K steps onward in this comparison, DLR+CoLA achieves the lowest perplexity at every milestone (12.08 at 150K vs. LOST 12.80) while using 2.3× less per-GPU memory than LOST (27.03GB vs. 62.15GB). Full-Rank Adam and 8-bit SLTrain encounter OOM at this batch size. This shows that the foldable plug-in extends to a memory-bound 7B setting while achieving the best perplexity among the compared runs.

System measurements (LLaMA 1B). To corroborate that DLR improves perplexity without introducing a practical efficiency bottleneck, we report end-to-end system measurements on 1B models with `torch.compile` enabled in Tab. 3. Each entry reports validation PPL together with peak GPU memory after initialization and training throughput after warmup, under the same multi-node H100 setup described in the table caption. On the nonlinear low-rank backbone, adding DLR improves CoLA from PPL 15.76 to 14.26 while keeping memory and throughput nearly unchanged (13.10→13.09GB; 1,079,350→1,044,264 tok/s). On the linear low-rank backbone, DLR improves PPL 18.22→15.72 with only a small memory increase (12.58→13.09GB) and a modest throughput decrease (1,099,699→1,020,316 tok/s). In contrast, sparse-compensation baselines show substantially lower throughput (SLTrain 640,132 tok/s; LOST 515,971 tok/s) and/or higher peak memory (14.55GB and 19.33GB), highlighting that DLR retains the efficiency profile of pure low-rank training while recovering much of the quality gap.

4.3 Folded checkpoints transfer cleanly to supervised fine-tuning

A core appeal of foldability is that the training-only residual leaves no trace at deployment. We test this end-to-end by taking the 1B Full-Rank, CoLA, and DLR+CoLA pre-trained checkpoints, folding the DLR branch into the up-projection (Sec. 3), running full-parameter SFT on Alpaca-cleaned, and evaluating six standard benchmarks with `lm-evaluation-harness` (Gao et al., 2024) (details in Appendix E).

Tab. 4 reports mean ± std accuracy across three SFT seeds applied to the same fixed pre-trained checkpoint per method. DLR +CoLA attains a slightly higher average (39.81%, vs. 39.55% Full-Rank and 39.39% CoLA), narrows the HellaSwag gap to Full-Rank, and achieves the highest PIQA accuracy. This supports the *train with* DLR → *fold* → *deploy* workflow: the downstream gain

Table 3: System measurements on 1B models with `torch.compile` enabled. All runs use DDP on 8 nodes $\times$ 4 NVIDIA H100 SXM5 GPUs (94GB per GPU), a local 7.68TB NVMe drive, and 4$\times$ Infiniband NDR200. We report validation PPL, total parameter count, max GPU memory after initialization (`max_memory`), and throughput (global tokens/s) after warmup.

Method	PPL $\downarrow$	Params (M)	Memory (GB)	Throughput (tok/s)
Full-Rank	13.40	1339	17.45	700697
Low-Rank	18.22	609	12.58	1099699
Low-Rank + DLR	15.72	609	13.09	1020316
Low-Rank + LaX + DLR	15.29	610	14.96	878076
CoLA	15.76	609	13.10	1079350
CoLA + DLR	14.26	609	13.09	1044264
CoLA + LaX	14.78	610	15.58	846626
CoLA + LaX + DLR	14.67	610	15.59	841800
SLTrain	15.40	609	14.55	640132
LOST	15.02	609	19.33	515971
LOST + DLR	14.74	609	19.33	595978

appears after folding, while deployment keeps the same graph, parameter count, FLOPs, and memory footprint as vanilla CoLA.

Table 4: **Downstream evaluation after Alpaca-cleaned SFT (1B models).** Each cell reports zero-shot accuracy as mean $\pm$ standard deviation across three SFT seeds applied to the same fixed pre-trained checkpoint. DLR +CoLA achieves a slightly higher average while folding to the same deployed graph as vanilla CoLA; **bold** indicates the best per-task mean.

Task	DLR +CoLA	CoLA	Full-Rank
ARC-Challenge	0.2327 ± 0.0027	$\mathbf{0.2361 \pm 0.0010}$	0.2190 ± 0.0013
BoolQ	0.3783 ± 0.0000	0.3783 ± 0.0000	0.3783 ± 0.0000
HellaSwag	0.3519 ± 0.0011	0.3296 ± 0.0009	$\mathbf{0.3564 \pm 0.0001}$
MMLU	0.2295 ± 0.0000	0.2295 ± 0.0000	0.2295 ± 0.0000
PIQA	$\mathbf{0.6750 \pm 0.0022}$	0.6649 ± 0.0005	0.6674 ± 0.0027
WinoGrande	0.5212 ± 0.0012	$\mathbf{0.5254 \pm 0.0061}$	0.5222 ± 0.0020
Average	$\mathbf{0.3981 \pm 0.0004}$	0.3939 ± 0.0011	0.3955 ± 0.0006

5 Conclusion, Limitations and Outlook

We introduced DLR, a training-only, parameter-free residual plug-in for low-rank pre-training that is folded into the up-projection after training, leaving the deployed low-rank graph unchanged. Across LLaMA scales from 60M to 7B, DLR improves representative low-rank backbones in most matched-token-budget settings, retains the efficiency profile of the base decoder, and transfers cleanly after supervised fine-tuning. Together, these results support foldable latent residuals as a practical way to improve low-rank pre-training dynamics without adding inference-time complexity.

Limitations. DLR improves training dynamics but does not remove the approximation ceiling imposed by a fixed rank-r bottleneck. Its scale is shape-based and fixed at $\alpha = 1$ in our experiments. We do not yet characterize how to retune it for architectures or rank regimes with substantially different latent statistics. Our evaluation focuses on LLaMA-style language models trained on C4 and instruction-tuned on Alpaca-cleaned, leaving other modalities, architectures, and tasks for future work.

Outlook. More broadly, any training-time latent intervention that remains linear in z may admit a closed-form fold, suggesting a design space of inference-free training plug-ins beyond the particular contiguous duplication map studied here.

AI USE STATEMENT

We used `Codex GPT-6 Astra` and `Claude Fable 5.1` to correct grammatical errors in the manuscript, fix minor compilation issues in Overleaf, and debug programming errors encountered during implementation. Apart from these auxiliary uses, the research ideas, theoretical contributions, and the writing of this paper were entirely carried out by the authors. The code implementation is adapted from the open-source code of (Glentis et al., 2025).

REPRODUCIBILITY STATEMENT

Our implementation is available in an anonymous repository.[1] Appendix A describes the model architectures, training hyperparameters, random seeds, and tokenization settings, and provides reference implementations of DLR and its folding operation. Appendices E and F describe the supervised fine-tuning and zero-shot evaluation protocols.

[1] `https://anonymous.4open.science/r/Efficient-Pretraining-preview-34C4/`

REFERENCES

Xi Chen, Kaituo Feng, Changsheng Li, Xunhao Lai, Xiangyu Yue, Ye Yuan, and Guoren Wang. Fira: Can we achieve full-rank training of llms under low-rank constraint?, 2024. URL https://arxiv.org/abs/2410.01623.

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.

Athanasios Glentis, Jiaxiang Li, Qiulin Shang, Andi Han, Ioannis Tsaknakis, Quan Wei, and Mingyi Hong. Scalable parameter and memory efficient pretraining for llm: Recent algorithmic advances and benchmarking, 2025. URL https://arxiv.org/abs/2505.22922.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury,

Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

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, 2024. URL https://arxiv.org/abs/2406.02214.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

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

Yuezhou Hu, Kang Zhao, Weiyu Huang, Jianfei Chen, and Jun Zhu. Accelerating transformer pre-training with 2:4 sparsity, 2024. URL https://arxiv.org/abs/2404.01847.

Keller Jordan, Hanie Sedghi, Olga Saukh, Rahim Entezari, and Behnam Neyshabur. Repair: Renormalizing permuted activations for interpolation repair. *arXiv preprint arXiv:2211.08403*, 2022.

Siddhartha Rao Kamalakara, Acyr Locatelli, Bharat Venkitesh, Jimmy Ba, Yarin Gal, and Aidan N Gomez. Exploring low rank training of deep neural networks. *arXiv preprint arXiv:2209.13569*, 2022.

Mikhail Khodak, Neil Tenenholtz, Lester Mackey, and Nicolo Fusi. Initialization and regularization of factorized neural layers. *arXiv preprint arXiv:2105.01029*, 2021.

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, 2025. URL https://arxiv.org/abs/2508.02668.

Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. Relora: High-rank training through low-rank updates. *arXiv preprint arXiv:2307.05695*, 2023.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Ziyue Liu, Ruijie Zhang, Zhengyang Wang, Zi Yang, Paul Hovland, Bogdan Nicolae, Franck Cappello, and Zheng Zhang. Cola: Compute-efficient pre-training of llms via low-rank activation, 2025. URL https://arxiv.org/abs/2502.10940.

Roy Miles, Pradyumna Reddy, Ismail Elezi, and Jiankang Deng. Velora: Memory efficient training using rank-1 sub-token projections, 2024. URL https://arxiv.org/abs/2405.17991.

Zhanfeng Mo, Long-Kai Huang, and Sinno Jialin Pan. Parameter and memory efficient pretraining via low-rank riemannian optimization. In *The Thirteenth International Conference on Learning Representations*, 2025.

Mohammad Mozaffari, Amir Yazdanbakhsh, Zhao Zhang, and Maryam Mehri Dehnavi. Slope: Double-pruned sparse plus lazy low-rank adapter pretraining of llms, 2025. URL https://arxiv.org/abs/2405.16325.

Le-Trung Nguyen, Aël Quélennec, Enzo Tartaglione, Samuel Tardieu, and Van-Tam Nguyen. Activation map compression through tensor decomposition for deep learning, 2024. URL https://arxiv.org/abs/2411.06346.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt

Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

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.

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, 2023. URL https://arxiv.org/abs/1910.10683.

Olga Saukh, Dong Wang, Haris Šikić, Yun Cheng, and Lothar Thiele. Cut less, fold more: Model compression through the lens of projection geometry. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=JV9CEtKLQF.

Yara Shamshoum, Nitzan Hodos, Yuval Sieradzki, and Assaf Schuster. Compact: Compressed activations for memory-efficient llm training, 2025. URL https://arxiv.org/abs/2410.15352.

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, 2023a. URL https://arxiv.org/abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023b. URL https://arxiv.org/abs/2307.09288.

Dong Wang, Haris Šikić, Lothar Thiele, and Olga Saukh. Forget the data and fine-tuning! just fold the network to compress. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025a. URL https://openreview.net/forum?id=W2Wkp9MQsF.

Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. Lora-pro: Are low-rank adapters properly optimized?, 2025b. URL https://arxiv.org/abs/2407.18242.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Ruijie Zhang, Ziyue Liu, Zhengyang Wang, and Zheng Zhang. Lax: Boosting low-rank training of foundation models via latent crossing, 2025. URL https://arxiv.org/abs/2505.21732.

Zhenyu Zhang, Ajay Jaiswal, Lu Yin, Shiwei Liu, Jiawei Zhao, Yuandong Tian, and Zhangyang Wang. Q-galore: Quantized galore with int4 projection and layer-adaptive low-rank gradients. *arXiv preprint arXiv:2407.08296*, 2024.

Jiancheng Zhao, Xingda Yu, Yuxiang Zhang, and Zhen Yang. Lor2c : Low-rank residual connection adaptation for parameter-efficient fine-tuning, 2025. URL https://arxiv.org/abs/2503.00572.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection, 2024. URL https://arxiv.org/abs/2403.03507.

Kaiye Zhou, Shucheng Wang, and Jun Xu. Switchlora: Switched low-rank adaptation can learn full-rank information, 2025. URL https://arxiv.org/abs/2406.06564.

APPENDIX

The following sections provide supplementary information omitted from the main text:

- Appendix H: Impact statement and existing assets.
- Appendix A: Implementation details.
- Appendix B: Additional ablations and long-horizon results.
- Appendix C: Gradient-path analysis results.
- Appendix D: The impact of DLR on training dynamics.
- Appendix E: Supervised fine-tuning details.
- Appendix F: Zero-shot evaluation.
- Appendix G: Related work.

A IMPLEMENTATION DETAILS

We trained over 100 models to evaluate the performance of DLR presented in this work. Experiments were run on an NVIDIA H100 SLURM cluster where each node has 4 NVIDIA H100 GPUs (each with 94GB memory), a local 7.68TB NVMe drive, and 4× Infiniband NDR200 adapters. Training time varies from 20 minutes (LLaMA 60M) to 5 hours (LLaMA 1B) depending on the model size. `Huggingface Hub`[2] is used to load the datasets. Weights & Biases (W&B)[3] is used to log training history, training result, and evaluation metrics. The source code is available at an anonymized repository: `https://anonymous.4open.science/r/Efficient-Pretraining-preview-34C4/`.

This section outlines the LLaMA architectures and the hyperparameters used during pre-training. To ensure fair comparison, we follow the same experimental settings as Zhao et al. (2024); Glentis et al. (2025). Tab. 6 summarizes the hyperparameters for different model scales. Across all architectures, we adopt a maximum sequence length of 256 and a batch size of 131,072 tokens. The learning rate is linearly warmed up during the first 10% of training steps, followed by a cosine annealing schedule that decays to 10% of the initial value. We use the T5-base tokenizer (Raffel et al., 2023), consistent with prior work (Han et al., 2024; Glentis et al., 2025).

Table 5: Pre-training hyperparameters for LLaMA architectures.

Params	Hidden	Intermediate	Heads	Layers	Steps	Tokens (B)
60M	512	1376	8	8	11K	1.4
130M	768	2048	12	12	22K	2.6
350M	1024	2736	16	24	65K	7.8
1B	2048	5461	32	24	140K	13.1

REFERENCE IMPLEMENTATION OF DLR

To make the definition of the expansion operator $\mathrm{Expand}_K(\cdot)$ concrete, we provide a short Python-style reference implementation of a DLR layer below. The paper writes the up-projection as $B \in \mathbb{R}^{d_{\text{out}} \times r}$ and Bz; the code below uses the common PyTorch row-major convention $B_{\text{code}} \in \mathbb{R}^{r \times d_{\text{out}}}$ and computes `z @ B`, which is the transpose convention of the mathematical B. The key operation is to replicate each latent coordinate $K = \lceil d_{\text{out}}/r \rceil$ times along the last dimension and truncate to d_{out}, matching Eq. 10–Eq. 11.

Listing 1: Python-style reference code for the DLR layer (Expand via repeat+truncate).

```
import math
```

[2] https://huggingface.co/docs/hub/index

[3] https://wandb.ai

Table 6: Training hyperparameters used in our DLR experiments (C4 pre-training). Common settings across these runs: C4 dataset; sequence length 256; token batch size $512 \times 256 = 131{,}072$ tokens; tokenizer T5-base; optimizer AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$, weight decay 0.1, and gradient clipping 0.5; cosine learning-rate schedule with minimum LR ratio 0.1; evaluation every 1,000 update steps; dtype bfloat16; seeds $\{41, 42, 43, 44, 45\}$.

Model	World Size	Batch	Total Batch	r	LR	Steps	Warmup	AdamW ϵ
60M	8	64	512	128	0.01	11,000	1,100	10^{-8}
130M	16	32	512	256	0.005	22,000	2,200	10^{-6}
350M	16	32	512	256	0.003	65,000	6,500	10^{-6}
1B	32	16	512	512	0.002	140,000	10,000	10^{-6}

```python
def expand_k(z, d_out):
    # z: [..., r]
    r = z.shape[-1]
    K = math.ceil(d_out / r)
    # Repeat each latent coordinate K times (e.g., PyTorch: z.repeat_interleave(K, dim=-1))
    u = repeat_interleave(z, K, axis=-1) # length r*K
    return u[..., :d_out]              # truncate to d_out

def dlr_layer(x, A, B, alpha, bias=None, act=None):
    # x: [..., d_in], A: [d_in, r], B: [r, d_out]
    # B is B_code = B_math^T, so z @ B corresponds to B_math z.
    z = x @ A
    if act is not None:
        z = act(z)

    y_lr = z @ B
    d_out = B.shape[-1]
    u = expand_k(z, d_out)
    K = math.ceil(d_out / z.shape[-1])
    y = y_lr + (alpha / math.sqrt(K)) * u
    if bias is not None:
        y = y + bias
    return y
```

Reference implementation of the fold operation. Folding a trained DLR layer reduces to a single in-place update of B_{code} that absorbs the structured residual; equivalently, in the paper notation, it applies $B \leftarrow B + (\alpha/\sqrt{K})R^\top$ in Eq. 12. The resulting checkpoint is a drop-in replacement for the underlying rank-r decoder. The operation is exact (no learning, no approximation) and runs once at training termination.

Listing 2: Python-style reference code for `fold()`: absorb the DLR residual into B in closed form, see Eq. 12.

```python
import math

def fold_(B, alpha):
    r, d_out = B.shape
    K = math.ceil(d_out / r)
    scale = alpha / math.sqrt(K)
    for j in range(r):
        lo = j * K
        hi = min((j + 1) * K, d_out)
        B[j, lo:hi].add_(scale)
    return B
```

The loop is the transpose-layout version of the paper update $B_{\text{math}} \mathrel{+}= (\alpha/\sqrt{K})R^\top$ in Eq. 12. Equivalently, for the code layout $B_{\text{code}} = B_{\text{math}}^\top$, row j of B receives the constant $\alpha/\sqrt{K}$ on the output coordinates in group $\mathcal{G}_j$. It runs in $\mathcal{O}(d_{\text{out}})$ time and $\mathcal{O}(1)$ extra memory beyond the in-place buffer of $B^\star$. After folding, the original α, K, the $\texttt{Expand}_\texttt{K}$ branch, and any buffer storing R are discarded; what remains is a vanilla rank-r low-rank layer parameterized by $(A, B^\star)$.

Fold consistency on a trained checkpoint. We verify the closed-form fold operation on one trained 1B DLR +CoLA checkpoint used for this fold-consistency diagnostic. The checkpoint loads without missing or unexpected keys, and folding updates 168 DLR layers while leaving the parameter count unchanged (609.31M before and after folding). Under BF16 evaluation on C4 validation with the same 10M-token budget and sequence length 256 as our training-time evaluation, perplexity changes from 14.6295 before folding to 14.6301 after folding ($\Delta\text{PPL} = +0.0006$). The folded and unfolded logits match within mean absolute error 1.91×10^{-2} and safe mean relative error 1.44×10^{-2}, passing our BF16 tolerance. Folding also removes the training-only residual branch in our single-GPU diagnostic, reducing median forward latency from 29.59 ms to 21.18 ms with no change in peak memory. These results confirm that the fold identity in Eq. 12 holds up to floating-point roundoff in trained checkpoints.

B Additional ablations and long-horizon results

B.1 Variance-preserving correction ablation

We ablate the variance/scale correction of the duplicated residual by training DLR +CoLA with the default scaling $\alpha/\sqrt{K}$ versus an uncorrected variant (no $1/\sqrt{K}$), across model scales. Tab. 7 reports validation perplexity and shows that the correction consistently improves performance, supporting the variance-preserving design used in Eq. 11.

Table 7: **Variance-preserving correction ablation (CoLA backbone).** Validation perplexity across model scales for DLR +CoLA with vs. without the $1/\sqrt{K}$ correction; mean±std across seeds $\{41, 42, 43, 44, 45\}$.

DLR + CoLA	60M	130M	350M	1B
w/ correction	$32.96_{\pm 0.06}$	$23.80_{\pm 0.40}$	$18.38_{\pm 0.03}$	$14.26_{\pm 0.02}$
w/o correction	$33.80_{\pm 0.14}$	$24.46_{\pm 0.05}$	$19.02_{\pm 0.03}$	$14.60_{\pm 0.02}$

B.2 Initialization-offset ablation

From the folded perspective, DLR adds a structured block-constant offset to the learned decoder B. To separate this static initialization effect from the active residual path during training, we compare full DLR +CoLA with an *Init-offset CoLA* variant: at initialization, we fold the same $(\alpha/\sqrt{K})R^\top$ offset into B, then disable the DLR branch and train the resulting CoLA model normally. Tab. 8 shows that the initialization offset is a useful structured prior, improving over CoLA, but it does not match full DLR training. The remaining gap indicates that the active training-time residual path provides additional optimization benefit beyond a static folded initialization of B.

Table 8: **Initialization-offset ablation at LLaMA-1B.** Validation perplexity on C4. Init-offset CoLA folds the DLR offset into B at initialization and then trains without the DLR branch.

Method	PPL ↓
CoLA	15.76
Init-offset CoLA	$14.79_{\pm 0.04}$
DLR +CoLA	$14.26_{\pm 0.02}$

Table 9: Rank sensitivity for DLR +CoLA. Target rank r is varied as a fraction of the default rank r_0 at each scale. We report mean±std validation perplexity over seeds $\{41, 42, 43, 44, 45\}$ on C4.

Model params	Rank ratio	r	PPL ↓
60M	0.50	64	$37.26_{\pm 0.12}$
60M	0.75	96	$34.98_{\pm 0.08}$
60M	1.00	128	$32.96_{\pm 0.06}$
130M	0.50	128	$26.18_{\pm 0.11}$
130M	0.75	192	$24.57_{\pm 0.22}$
130M	1.00	256	$23.80_{\pm 0.40}$
350M	0.50	128	$20.59_{\pm 0.01}$
350M	0.75	192	$19.18_{\pm 0.03}$
350M	1.00	256	$18.38_{\pm 0.03}$
1B	0.50	256	$15.67_{\pm 0.02}$
1B	0.75	384	$14.77_{\pm 0.03}$
1B	1.00	512	$14.26_{\pm 0.02}$

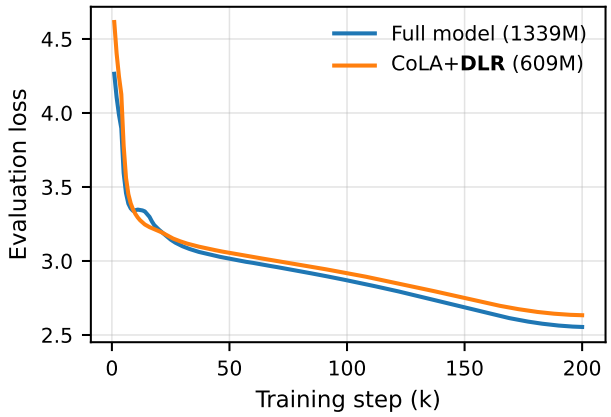


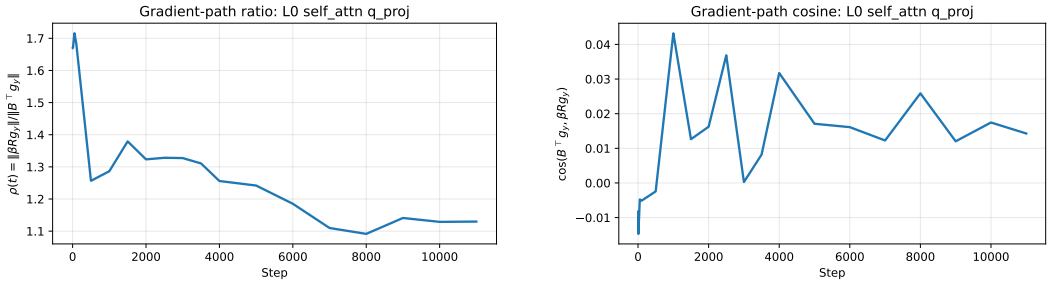
Figure 2: **Scaling-law token budget at LLaMA-1B with 26B tokens.** Validation loss trajectories on C4 for LLaMA-1B trained for 26B tokens, comparing full-rank pre-training to a low-rank backbone equipped with DLR (r=512). The full-rank model converges to a slightly lower loss, consistent with the 1B perplexity ordering reported in Tab. 1.

B.3 Target-rank sensitivity

To probe sensitivity to the target rank, we train DLR +CoLA models with reduced ranks $r \in \{0.5r_0, 0.75r_0\}$, where r_0 is the default rank used throughout the paper (Tab. 6). All settings follow the main recipe and use five random seeds $\{41, 42, 43, 44, 45\}$. Tab. 9 shows a smooth, monotonic trade-off: decreasing r degrades perplexity gradually rather than catastrophically. For instance, at 1B scale, reducing the rank from $r_0 = 512$ to 384 ($0.75r_0$) increases PPL from 14.26 to 14.77, and further to 15.67 at 256 ($0.5r_0$). Similar trends hold at 350M (18.38 → 19.18 → 20.59) and 130M (23.80 → 24.57 → 26.18), indicating that DLR remains effective under tighter rank budgets.

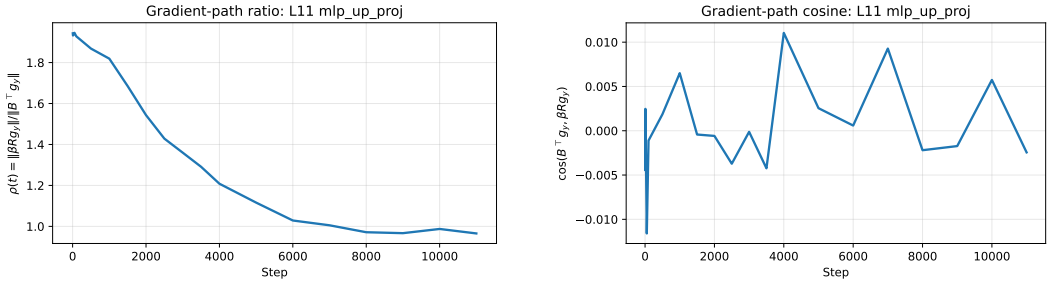
B.4 Scaling-law token budget at 1B

To complement the final validation perplexities in Tab. 1, we plot the full training trajectory for the 1B setting at a scaling-law token budget (26B tokens) in Fig. 2. The low-rank backbone equipped with DLR closely tracks the full-rank baseline throughout training and exhibits a small but consistent gap at convergence, matching the relative performance reported in Tab. 1.



(a) Ratio $\rho(t) = \|\beta R g_y\| / \|B^\top g_y\|$ (b) Cosine $\cos_t = \cos(B^\top g_y, \beta R g_y)$

Figure 3: **Gradient-path diagnostics (LLaMA-350M): `L0 self_attn q_proj`.** The DLR-induced component dominates early updates ($\rho(t) > 1$) and decays below 1 as training progresses, while remaining nearly orthogonal to the baseline low-rank gradient ($\cos_t \approx 0$).



(a) Ratio $\rho(t) = \|\beta R g_y\| / \|B^\top g_y\|$ (b) Cosine $\cos_t = \cos(B^\top g_y, \beta R g_y)$

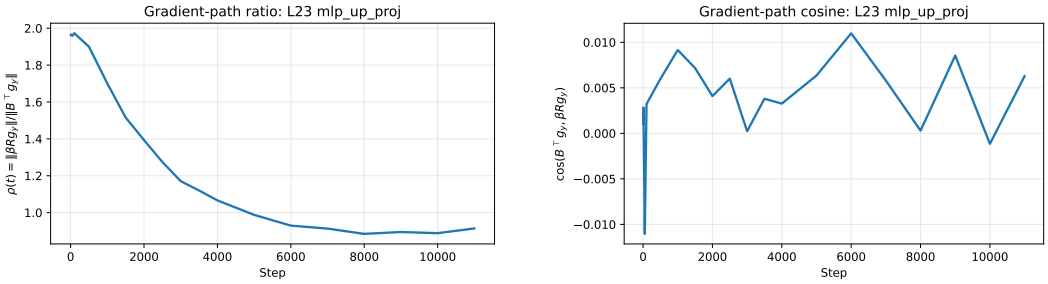
Figure 4: **Gradient-path diagnostics (LLaMA-350M): `L11 mlp_up_proj`.** The ratio $\rho(t)$ decreases from above 1 to below 1 over training, while $\cos_t$ stays close to 0, indicating a strong but complementary DLR gradient contribution.

C Gradient-path analysis results

We instrument the training loop to probe the gradient paths induced by the low-rank decoder and the DLR expansion operator at selected update steps. For each probed module output $y \in \mathbb{R}^{d_{\text{out}}}$, we capture y via a forward hook and compute $g_y = \partial \mathcal{L} / \partial y$ using `torch.autograd.grad`, avoiding any modification of parameter gradients. We then form the two latent-gradient components $B^\top g_y$ and $\beta R g_y$ (with $\beta = \alpha/\sqrt{K}$), where R corresponds to the DLR "expand-then-sum" operator (implemented as a scatter-add over duplicated groups). Following our implementation, we reshape g_y to $[N, d_{\text{out}}]$ by flattening batch and sequence dimensions, compute token-wise ℓ_2 norms, and report the token-average norms $\|B^\top g_y\|$ and $\|\beta R g_y\|$. We log $\rho(t) = \|\beta R g_y\| / \|B^\top g_y\|$ and the token-average cosine similarity $\cos_t = \langle B^\top g_y, \beta R g_y \rangle / (\|B^\top g_y\| \cdot \|\beta R g_y\|)$. Note that $\cos_t$ measures the *directional alignment of the two gradient components* (not the cosine between the parameter matrices B and R).

D The impact of DLR on training dynamics

We provide additional training-dynamics evidence by plotting evaluation loss over pre-training for three variants: (i) DLR+B (default), (ii) DLR only (no learnable decoder branch B), and (iii) Random duplication + B (fixed random duplication mapping with the same variance correction $1/\sqrt{K}$). All three runs enable `torch.compile` and are executed on the same DGX machine with $8\times$A100 GPUs connected via NVLink. Figure 6 shows that removing B significantly worsens optimization and yields consistently higher evaluation loss. Meanwhile, DLR+B and Random duplication + B converge along nearly identical trajectories and reach comparable final losses, supporting the interpretation of DLR as an optimization/conditioning aid that complements (rather than replaces) the learnable



(a) Ratio $\rho(t) = \|\beta R g_y\| / \|B^\top g_y\|$ (b) Cosine $\cos_t = \cos(B^\top g_y, \beta R g_y)$

Figure 5: **Gradient-path diagnostics (LLaMA-350M): `L23 mlp_up_proj`.** We observe the same trend as in other layers: the DLR term dominates early ($\rho(t) > 1$) and then decays, while remaining nearly orthogonal to the baseline gradient ($\cos_t \approx 0$).

low-rank decoder. Despite similar convergence, Random duplication is substantially less efficient: it achieves only 507,186 tok/s versus 1,445,134 tok/s for DLR+B, because it relies on indexing-based gathers instead of contiguous duplication. These results validate the design constraints used in the main method. First, keeping the learnable decoder B is necessary because the duplicated residual is not intended to replace the low-rank up-projection. Second, the residual map should be fixed and linear in z: any such map, including the fixed random mapping tested here, can in principle be folded into a same-shape decoder, whereas data-dependent or nonlinear maps generally cannot. Finally, using contiguous `repeat_interleave` rather than indexed gathers is what lets `torch.compile` fuse the duplication into surrounding computation and recover the throughput profile of vanilla low-rank training.

Definition of random duplication. For each module, we sample a *fixed* index vector $\texttt{idx} \in \{0, \dots, r-1\}^{d_{\text{out}}}$ once and reuse it throughout training, where $\texttt{idx} \sim \texttt{torch.randint}(0, r, (d_{\text{out}},))$ under a deterministic per-module seed (base seed XOR $\text{MD5}(\texttt{module_name})$). We form the expanded residual by indexing $z_{\text{exp}}[j] = z[\texttt{idx}[j]]$ (i.e., `gather`/`index_select`), which incurs irregular memory access and poorer fusion than deterministic `repeat_interleave`.

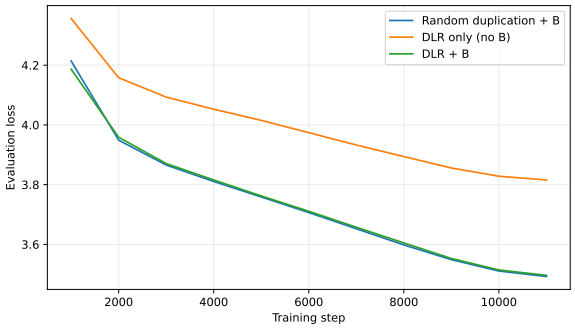


Figure 6: **Training dynamics on LLaMA-60M.** Evaluation loss curves for (i) DLR+B (default), (ii) DLR only (removing the learnable decoder branch B), and (iii) Random duplication + B (replacing deterministic duplication with a fixed random mapping while keeping the same variance correction $1/\sqrt{K}$). Removing B substantially degrades convergence, while DLR+B and Random duplication + B exhibit very similar trajectories.

E SUPERVISED FINE-TUNING DETAILS

For the downstream SFT evaluation in Tab. 4, we fine-tune the 1B Full-Rank, CoLA, and folded DLR +CoLA checkpoints on Alpaca-cleaned (52K instructions) for 3 epochs using full-parameter SFT. All SFT runs use 16×H100 DDP, effective batch size 64, learning rate 2×10^{-5} with cosine

schedule, BF16, and maximum sequence length 512. We then evaluate ARC-Challenge, BoolQ, HellaSwag, MMLU, PIQA, and WinoGrande with `lm-evaluation-harness` in the zero-shot setting. For each method, the three SFT seeds are applied to the same fixed pre-trained checkpoint, so the reported standard deviations capture SFT-side data-ordering variability rather than pre-training initialization variance. BoolQ and MMLU collapse to constant chance-level accuracy with zero across-seed variance for all three checkpoints, consistent with the behavior of pre-training-only 1B models on these formats (Han et al., 2024; Glentis et al., 2025); we therefore interpret the average mainly through the tasks where the checkpoints differentiate.

F ZERO-SHOT EVALUATION

In addition to validation perplexity on C4, we perform zero-shot evaluations of 1B-scale models on ARC-Challenge (ARC-C), BoolQ, HellaSwag, MMLU, PIQA, and WinoGrande, using standard zero-shot prompts (no instruction tuning). All low-rank variants use rank $r = 512$ and are trained with five random seeds $\{41, 42, 43, 44, 45\}$; we report mean$\pm$std across seeds. As shown in Tab. 10, DLR improves the average score over the pure low-rank baseline and largely closes the gap to full-rank; we treat these results as supplementary since several tasks remain near-chance at this pre-training-only stage.

Table 10: **Zero-shot evaluation on 1B models.** Each cell reports zero-shot accuracy as mean$\pm$std across five pre-training seeds $\{41, 42, 43, 44, 45\}$; the Avg. column is the across-task mean of the per-task means. BoolQ and MMLU saturate at constant chance-level accuracy (std$= 0$) for all four checkpoints, consistent with the well-documented behavior of pre-training-only 1B models on these formats.

Method (1B, r=512)	ARC-C	BoolQ	HellaSwag	MMLU	PIQA	WinoGrande	Avg.
Full-Rank	22.06±0.57	37.83±0.00	34.94±0.18	22.95±0.00	67.57±0.63	51.10±1.13	39.41
Low-Rank	19.78±0.76	37.83±0.00	27.47±0.16	22.95±0.00	60.28±0.76	51.16±1.21	36.58
DLR +Low-Rank	21.59±0.65	37.83±0.00	32.52±1.03	22.95±0.00	65.59±0.68	50.69±0.24	38.53
DLR +CoLA	22.29±0.57	37.83±0.00	34.20±0.24	22.95±0.00	66.81±0.58	50.61±1.57	39.12

G RELATED WORK

Related work on efficient pre-training spans four complementary directions. First, low-rank adapters popularized by LoRA learn update-level low-rank corrections (Hu et al., 2021); pre-training variants such as ReLoRA and SwitchLoRA recover higher effective ranks via periodic merges or frequent subspace switching (Lialin et al., 2023; Zhou et al., 2025), while LORO directly optimizes a fixed-rank factorization on the low-rank manifold (Mo et al., 2025). Second, memory-efficient gradient projection methods (GaLore and variants) project gradients into low-rank subspaces to reduce optimizer state, with Fira stabilizing training via norm-based scaling and a norm-growth limiter (Zhao et al., 2024; Zhang et al., 2024; Chen et al., 2024). Third, sparse-plus-low-rank designs increase expressivity by adding sparse components: SLTrain employs a fixed unstructured support with low memory overhead (Han et al., 2024; Glentis et al., 2025), LOST co-designs complementary low-rank and channel-wise structured sparse components guided by an SVD initialization (Li et al., 2025). A closely related latent-shaping line improves low-rank trainability without introducing an explicit sparse pathway: CoLA inserts a nonlinearity between factors to enforce low-rank structure in activations (Liu et al., 2025). LaX (Zhang et al., 2025) designs inter-layer latent crossing to enhance the capacity of low-rank models by establishing information flow across low-rank spaces of different layers. LoR2C (Zhao et al., 2025) introduces low-rank residual connections within the model layers to solve gradient vanishing during LoRA fine-tuning. ResLoRA (Wang et al., 2025b) integrates residual paths into the LoRA fine-tuning framework to accelerate training convergence and improve performance, utilizing merging strategies to ensure no additional computational cost during inference. Similarly, FST (Hu et al., 2024) and SLoPe (Mozaffari et al., 2025) optimize training for semi-structured (e.g., 2:4) or block-sparse patterns to leverage specialized kernels. Finally, post-training model folding clusters redundant channels/heads and merges them with variance-preserving corrections to compress models without data or fine-tuning (Wang et al., 2025a; Saukh et al., 2026).

Table 11: **Where DLR sits among residual-style methods.** We anchor against ResNet to remind readers that residual primitives predate efficient pre-training, then compare DLR with three pre-training / fine-tuning residual baselines. DLR is the only entry that combines (i) pre-training applicability, (ii) a parameter-free residual path, and (iii) an exact closed-form merge into the deployment graph (Eq. 12).

Method	Setting	Residual space	Learnable?	Main role	Foldable?
ResNet	Full	Activation	No	Stabilize optimization	N/A (identity)
ResLoRA	FT	Adapter	Yes	Improve adaptation	Sometimes
LoR2C	FT	Correction	Yes	Improve capacity	Sometimes
LaX	Pretrain	Inter-layer latent	Partial	Cross-layer flow	No
DLR	Pretrain	Intra-layer latent	No	Decoder-independent gradient path	**Yes**

DLR brings the folding insight *into training*: it introduces a *foldable* latent-space compensation within each low-rank (or CoLA-style) layer. Concretely, with a rank-r latent $z = s\,\phi(A^\top x)$, DLR augments the decoder output by a structured duplicated residual, which is a fixed replication matrix that maps each latent coordinate to a small group of output channels ; compared to the above, this design reduces FLOPs and optimizer state like low-rank methods, avoids sparse-kernel overheads, and empirically delivers favorable perplexity–efficiency trade-offs across LLaMA scales under matched token budgets. Orthogonal to these four directions, a complementary line of work targets the activation-memory bottleneck by compressing activations stored for backpropagation: VeLoRA compresses intermediate activations via rank-1 sub-token projections and reconstructs them approximately during the backward pass (Miles et al., 2024), CompAct stores low-rank, randomly projected activations and only decompresses gradients for the optimizer update, achieving 25–30% peak-memory savings for LLaMA pre-training and up to 50% for RoBERTa fine-tuning (Shamshoum et al., 2025), and tensor-decomposition methods compress activation maps using (high-order) SVD with theoretical guarantees on gradient approximation error (Nguyen et al., 2024).

Tab. 11 is not intended as a complete taxonomy of efficient pre-training. Instead, it isolates residual-style interventions that are easy to conflate with DLR. ResNet (He et al., 2015) is included only as a conceptual anchor for identity residuals, not as an efficient pre-training baseline. Within this narrower comparison, DLR differs by combining a pre-training setting, a parameter-free intra-layer latent residual, and a closed-form fold into the deployed low-rank graph.

H Impact statement and existing assets

Impact statement. This work studies an efficiency technique for low-rank language-model pre-training. Its intended positive impact is to reduce the compute and memory required to train and deploy low-rank models, which may make pre-training experiments more accessible and reduce resource use. The method does not introduce a new application, dataset, or user-facing system, but improvements in pre-training efficiency could also lower the cost of training models that may be misused if deployed without appropriate safety evaluation. We therefore view DLR as a general-purpose training method whose downstream risks are primarily those of the models and deployment settings to which it is applied.

Existing assets and licenses. We use publicly available research assets and cite their original sources where they are introduced in the paper. Pre-training uses C4 (Raffel et al., 2020) accessed through the Hugging Face Datasets interface; tokenization uses the T5-base tokenizer (Raffel et al., 2023); supervised fine-tuning uses Alpaca-cleaned; and downstream evaluation uses `lm-evaluation-harness`. We do not redistribute the raw datasets or third-party model checkpoints as part of this source package. Users reproducing the experiments should obtain these assets from their original providers and comply with the licenses and terms listed on the corresponding dataset, model, and software package pages. Our code release is adapted from the open-source implementation of Glentis et al. (2025); the release will include attribution and license information for the reused code and experiment scripts.