DyRA: Dynamic Residual Approximation for Efficient Matrix Multiplication in DNNs

Daewon Chae[1] **Hyunwon Chung**[1] **Changwoo Lee**[2*] **Hun-Seok Kim**[1]
[1]Department of EECS, University of Michigan [2]Qualcomm AI Research
{daewon, hyunwon, hunseok}@umich.edu changle@qti.qualcomm.com

Abstract

Large-scale foundation models achieve strong performance across diverse tasks, but their size makes inference costly, largely due to dense matrix multiplications. Prior work reduces this cost by replacing dense weight matrices with efficient structured forms such as low-rank factorizations. However, these methods approximate weights rather than the output activations that determine inference accuracy. Consequently, small weight-space errors can be amplified by input activations, producing large output errors. In this work, we propose DyRA, an input-adaptive method that improves structured matrix multiplication approximation by correcting residual output errors during inference. We show that matrix multiplication can be approximated more effectively by directly optimizing low-rank factors of the output. DyRA builds on this insight by dynamically approximating and correcting the output error introduced by structured weight approximations. This combines efficient structured computation with input-dependent correction, yielding a more faithful approximation of full matrix multiplication under the same computational budget. Across vision, speech, and language models, DyRA consistently improves the accuracy–efficiency trade-off over structured weight approximations alone. Notably, DyRA achieves a $1.5\times$ end-to-end GPU speedup for DINOv3 while reducing accuracy degradation by more than $3\times$ relative to weight-only baselines. Our code is available at https://github.com/daewon88/DyRA.

1 Introduction

Recent advances in deep learning have achieved state-of-the-art performance across diverse tasks, but at substantial computational cost. Dense matrix multiplications account for much of the computation in modern deep networks[1], motivating work on accelerating them by replacing dense weight matrices with efficient *structured* forms [5, 11, 21, 22, 24, 31]. Low-rank approximation is a representative example, while block-structured variants such as Monarch [11] and BLAST [22] offer more expressive alternatives. Given a pre-trained model, these methods approximate its weight matrices via closed-form decompositions or iterative optimization, enabling faster inference with reduced FLOPs.

Despite their effectiveness, these methods are fundamentally weight-centric: they optimize approximations in weight space without explicitly accounting for the input activations seen at inference time. However, the actual computation is determined by the realized product between activations and weights, not by the weights alone. Thus, even small weight-space residuals can be amplified by the input, causing large output activation errors [40]. We illustrate this limitation on DINOv3 [35], a large-scale vision foundation model. As shown in Figure 1, replacing all dense linear layers with structured matrices such as low-rank, Monarch, and BLAST makes PCA feature visualizations noticeably noisier and less semantically organized than those of the original model. This demonstrates that approximating weights alone may fail to preserve model behavior.

*Contributed to this work while at the University of Michigan.
[1]For example, in ViT-L/16 [14], linear layers account for approximately 96.7% of total FLOPs.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

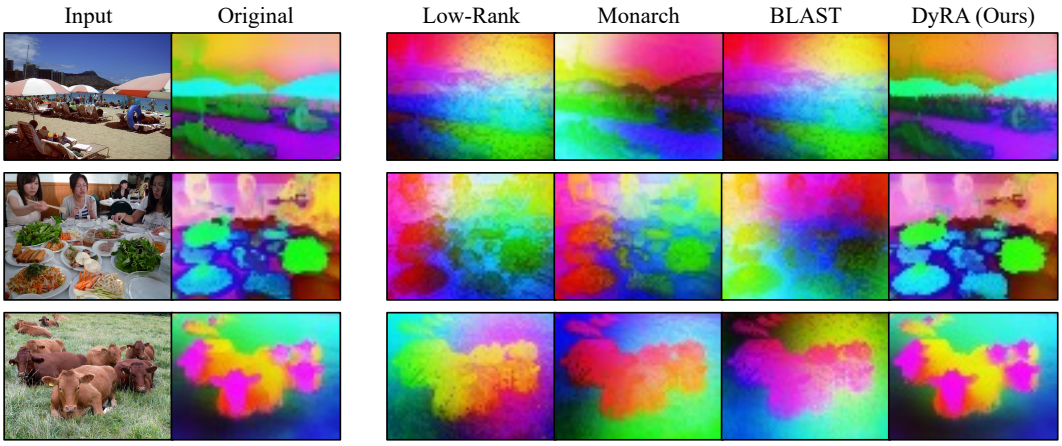


Figure 1: DINOv3 feature visualizations under a 60% FLOPs reduction. After replacing all dense linear layers with structured weights, we visualize the resulting features using the DINO PCA protocol, mapping the first three principal components to RGB. Weight-only approximations produce noisy, semantically degraded features, while DyRA preserves structures close to the original model.

To address this limitation, we propose DyRA (**Dy**namic **R**esidual **A**pproximation), an input-adaptive method that improves structured matrix multiplication by *correcting output errors on the fly*. DyRA augments a static weight approximation with a dynamic correction adapted to the input activations. This correction is computed through nested optimization during inference to minimize residual output errors induced by the approximated weight matrix. Rather than evaluating the exact residual through an expensive dense multiplication, DyRA uses low-rank factors to approximate the residual directly in the output space, not the weight space. These factors are efficiently computed at inference time using alternating least squares, without explicitly forming the dense residual product. Consequently, DyRA provides input-adaptive output correction that more faithfully approximates the original matrix multiplication under a fixed computational budget. Figure 1 shows that this correction substantially preserves the semantic structure of DINOv3 features compared with weight-only approximations.

We validate DyRA as a test-time model approximation and acceleration technique across pre-trained vision, language, and speech models. Combined with structured matrices such as low-rank, Monarch, and BLAST, DyRA consistently improves the accuracy–efficiency trade-off over weight-only baselines. For example, on semantic segmentation with DINOv3 at a 60% FLOPs reduction, DyRA limits the mIoU drop to 1.7 points, whereas the weight-only baseline drops by 54.1 points. These gains translate to practical runtime improvements: on DINOv3, it achieves a $1.52\times$ GPU speedup while incurring only 1/3 of the accuracy drop of the weight-only baseline at the same speedup.

2 Proposed Method

2.1 Overview

Starting from a pre-trained weight matrix $\boldsymbol{W}$, matrix approximation or model compression methods approximate the weight with a computationally efficient structured form $\widehat{\boldsymbol{W}}$ (e.g., low-rank or block-structured approximation). The original weight matrix can be written as

$$\boldsymbol{W} = \widehat{\boldsymbol{W}} + \boldsymbol{W}_{\text{res}}, \tag{1}$$

where $\boldsymbol{W}_{\text{res}} = \boldsymbol{W} - \widehat{\boldsymbol{W}}$ denotes the residual. Accordingly, the exact matrix product can be written as

$$\boldsymbol{X}\boldsymbol{W} = \boldsymbol{X}\widehat{\boldsymbol{W}} + \boldsymbol{X}\boldsymbol{W}_{\text{res}} \tag{2}$$

given input activation $\boldsymbol{X}$. Conventional approximation methods use only the first term and approximate the full product as $\boldsymbol{X}\boldsymbol{W} \approx \boldsymbol{X}\widehat{\boldsymbol{W}}$. The approximation error is thus given by $\boldsymbol{X}\boldsymbol{W}_{\text{res}}$, rather than the residual weight matrix $\boldsymbol{W}_{\text{res}}$ alone. This implies that, even when $\widehat{\boldsymbol{W}}$ is chosen to minimize $\|\boldsymbol{W}_{\text{res}}\|$, the resulting error remains input-dependent and may still be large in the output space due to its interaction with the input $\boldsymbol{X}$.

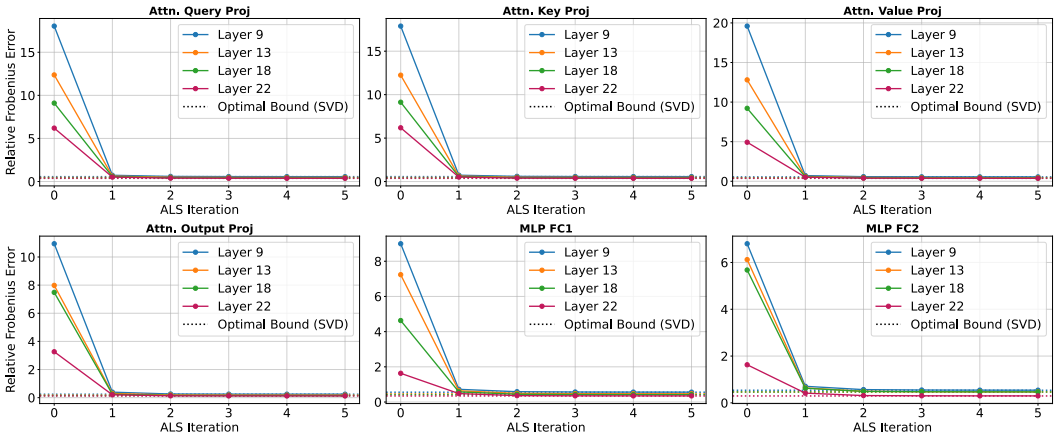


Figure 2: Convergence of ALS-based residual product approximation across ViT layers. Relative squared Frobenius error is measured for attention projections and MLP layers with target rank $r = 16$.

To address this limitation, DyRA explicitly accounts for the input-dependent residual product $\boldsymbol{X}\boldsymbol{W}_{\text{res}}$ by approximating it at inference time. This augments the static approximation $\boldsymbol{X}\widehat{\boldsymbol{W}}$ with a dynamic correction, yielding a more faithful estimate of $\boldsymbol{X}\boldsymbol{W}$ within a fixed computational budget. In the following, we introduce the *dynamic residual approximation* that enables this correction efficiently, and analyze its key properties.

2.2 Dynamic Residual Approximation

Given an input matrix $\boldsymbol{X} \in \mathbb{R}^{n\times c_1}$ and a residual weight matrix $\boldsymbol{W}_{\text{res}} \in \mathbb{R}^{c_1\times c_2}$, our goal is to efficiently approximate the residual product $\boldsymbol{X}\boldsymbol{W}_{\text{res}}$ at inference time *without performing the full matrix multiplication*. To this end, we formulate the residual output computation as an optimization problem over low-rank factors. Specifically, we seek a low-rank approximation $\boldsymbol{X}\boldsymbol{W}_{\text{res}} \approx \boldsymbol{L}\boldsymbol{R}$, where $\boldsymbol{L} \in \mathbb{R}^{n\times r}$ and $\boldsymbol{R} \in \mathbb{R}^{r\times c_2}$ with $r \ll \min(n, c_2)$, by solving

$$\min_{\boldsymbol{L}\in\mathbb{R}^{n\times r},\, \boldsymbol{R}\in\mathbb{R}^{r\times c_2}} \|\boldsymbol{X}\boldsymbol{W}_{\text{res}} - \boldsymbol{L}\boldsymbol{R}\|_F^2 . \tag{3}$$

This objective directly approximates the realized residual output $\boldsymbol{X}\boldsymbol{W}_{\text{res}}$, rather than the residual weight matrix alone. Its optimal rank-r solution is given by the truncated singular value decomposition (SVD) of $\boldsymbol{X}\boldsymbol{W}_{\text{res}}$ [15]. However, computing this solution requires explicitly forming the full product $\boldsymbol{X}\boldsymbol{W}_{\text{res}}$, which defeats the purpose of reducing computational cost. We therefore directly optimize the low-rank factors $\boldsymbol{L}$ and $\boldsymbol{R}$ without explicitly forming $\boldsymbol{X}\boldsymbol{W}_{\text{res}}$. Specifically, we optimize the objective in Eq. 3 using alternating least squares (ALS). Starting from an initial right factor $\boldsymbol{R}^{(0)}$, ALS alternates between least-squares subproblems obtained by fixing one factor and optimizing the other. This yields the following closed-form updates at iteration t: [2]

$$\boldsymbol{L}^{(t+1)} = \boldsymbol{X}\left(\boldsymbol{W}_{\text{res}}\boldsymbol{R}^{(t)\top}\right)\left(\boldsymbol{R}^{(t)}\boldsymbol{R}^{(t)\top}\right)^{-1}, \tag{4}$$

$$\boldsymbol{R}^{(t+1)} = \left(\boldsymbol{L}^{(t+1)\top}\boldsymbol{L}^{(t+1)}\right)^{-1}\left(\boldsymbol{L}^{(t+1)\top}\boldsymbol{X}\right)\boldsymbol{W}_{\text{res}}. \tag{5}$$

A key efficiency advantage of this ALS formulation is that each update is performed through low-rank (r) intermediate computations. In the $\boldsymbol{L}$ update, $\boldsymbol{W}_{\text{res}}$ is first projected onto the rank-r right factor, yielding $\boldsymbol{W}_{\text{res}}\boldsymbol{R}^{(t)\top} \in \mathbb{R}^{c_1\times r}$. This reduces the output dimension from c_2 to r, so that $\boldsymbol{X}$ is multiplied only with a slim $c_1 \times r$ matrix. Similarly, in the $\boldsymbol{R}$ update, the input matrix is first projected onto the rank-r left factor as $\boldsymbol{L}^{(t+1)\top}\boldsymbol{X} \in \mathbb{R}^{r\times c_1}$, reducing the input side from n to r before being multiplied by $\boldsymbol{W}_{\text{res}}$. As a result, the residual output is estimated efficiently as $\boldsymbol{L}\boldsymbol{R} \approx \boldsymbol{X}\boldsymbol{W}_{\text{res}}$ using low-dimensional intermediate products, without explicitly forming the full residual output.

[2] In practice, we use a ridge-regularized version of these updates by adding a small $\lambda \geq 0$ to the Gram matrices, i.e., $\boldsymbol{R}^{(t)}\boldsymbol{R}^{(t)\top} + \lambda\boldsymbol{I}_r$ and $\boldsymbol{L}^{(t+1)\top}\boldsymbol{L}^{(t+1)} + \lambda\boldsymbol{I}_r$, for numerical stability.

Convergence of ALS Residual Approximation We empirically examine the convergence of the ALS-based residual product approximation using a Vision Transformer (ViT) [14] evaluated with ImageNet [13] samples. We apply ALS to approximate the residual product $\boldsymbol{X}\boldsymbol{W}_{\text{res}}$ at multiple linear layers of the model with a small target rank $r = 16$, initializing $\boldsymbol{R}^{(0)}$ from a Gaussian distribution. In Figure 2, we report the relative squared Frobenius error across ALS steps for both attention projections and multi-layer perceptron (MLP) layers. Across layers, the approximation rapidly approaches the optimal rank-r solution within only one or two iterations. These results indicate that only a single update per factor is sufficient to effectively approximate the residual product, demonstrating the efficiency of ALS for the dynamic residual approximation. Further details on the experimental setup, along with additional results across various layers, are provided in Appendix E.3.

2.3 Analysis of DyRA with One-Step ALS Residual Approximation

Given the rapid convergence observed above, we use only one ALS iteration as the default configuration of DyRA throughout this work.[3] The resulting approximation is

$$\boldsymbol{X}\boldsymbol{W} \approx \boldsymbol{X}\widehat{\boldsymbol{W}} + \mathrm{ALS}_1(\boldsymbol{X}\boldsymbol{W}_{\text{res}}) = \boldsymbol{X}\widehat{\boldsymbol{W}} + \boldsymbol{L}^{(1)}\boldsymbol{R}^{(1)}, \tag{6}$$

where $\mathrm{ALS}_1(\cdot)$ denotes one ALS iteration, which updates $\boldsymbol{L}$ once and $\boldsymbol{R}$ once. Notably, the left-factor update admits an efficient implementation:

$$\boldsymbol{L}^{(1)} = \boldsymbol{X} \underbrace{\left(\boldsymbol{W}_{\text{res}}\boldsymbol{R}^{(0)\top}\right)\left(\boldsymbol{R}^{(0)}\boldsymbol{R}^{(0)\top}\right)^{-1}}_{\boldsymbol{P}_{\boldsymbol{L}}\ :\ \text{precomputable}} \tag{7}$$

where $\boldsymbol{P}_{\boldsymbol{L}}$ is *precomputed offline* as it depends only on $\boldsymbol{W}_{\text{res}}$ and the fixed initialization $\boldsymbol{R}^{(0)}$. Thus, computing $\boldsymbol{L}^{(1)}$ at inference time requires only multiplying $\boldsymbol{X}$ by a slim $c_1 \times r$ matrix $\boldsymbol{P}_{\boldsymbol{L}}$, reducing the online cost of ALS_1 largely to the $\boldsymbol{R}^{(1)}$ update.

Subspace Comparison under the Same Rank Increase We now analyze this one-step variant to clarify what DyRA contributes beyond static approximation. While DyRA can be combined with any structured weight $\widehat{\boldsymbol{W}}$ such as Monarch [11] and BLAST [22], we focus on the low-rank case for analytical clarity. We first compare DyRA with simply increasing the rank of a static low-rank approximation. We then quantify the additional cost of DyRA and use it to construct a FLOPs-matched static baseline.

Specifically, we compare DyRA at correction rank r with a static (weight-only) low-rank approximation whose rank is increased by the same amount, from k to $k + r$. For DyRA, we take the rank-k truncated SVD of $\boldsymbol{W}$ as the static approximation and apply rank-r DyRA correction. The resulting DyRA is compared with the static rank-$(k + r)$ truncated SVD approximation of the weight. The following proposition characterizes this difference.

Proposition 1. *Let $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ be the SVD of $\boldsymbol{W}$, and let $\widehat{\boldsymbol{W}} = \boldsymbol{U}_{1:k}\boldsymbol{\Sigma}_{1:k}\boldsymbol{V}_{1:k}^\top$ be its rank-k truncated SVD approximation. Assume DyRA approximates $\boldsymbol{X}\boldsymbol{W}_{\text{res}}$ with rank r by one ALS iteration, initialized with $\boldsymbol{R}^{(0)} = \boldsymbol{V}_{k+1:k+r}^\top$. Then the resulting approximation can be written as*

$$\boldsymbol{X}\boldsymbol{W} \approx \boldsymbol{X}\begin{bmatrix}\boldsymbol{U}_{1:k}\boldsymbol{\Sigma}_{1:k} & \boldsymbol{U}_{k+1:k+r}\boldsymbol{\Sigma}_{k+1:k+r}\end{bmatrix}\begin{bmatrix}\boldsymbol{V}_{1:k}^\top \\ \boldsymbol{R}^{(1)}\end{bmatrix}, \tag{8}$$

where $\boldsymbol{R}^{(1)} = (\boldsymbol{A}_r^{\text{res}\top}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{A}_r^{\text{res}})^{-1}\boldsymbol{A}_r^{\text{res}\top}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{W}_{\text{res}}$ and $\boldsymbol{A}_r^{\text{res}} = \boldsymbol{U}_{k+1:k+r}\boldsymbol{\Sigma}_{k+1:k+r}$.

The proposition shows that DyRA uses the same additional left directions as the static rank-$(k + r)$ SVD approximation, $\boldsymbol{U}_{k+1:k+r}\boldsymbol{\Sigma}_{k+1:k+r}$. The difference lies in how these directions are paired on the right side. A static rank-$(k + r)$ approximation uses the fixed weight-only basis $\boldsymbol{V}_{k+1:k+r}^\top$. In contrast, DyRA replaces this basis with $\boldsymbol{R}^{(1)}$, which is optimized for the current input through $\boldsymbol{X}^\top\boldsymbol{X}$. Thus, DyRA is not merely increasing the static rank; it constructs an input-adaptive residual subspace tailored to the realized product $\boldsymbol{X}\boldsymbol{W}_{\text{res}}$.

[3] While more ALS iterations can further refine the approximation, one iteration captures most of the gain. Thus, under a fixed computation budget, we use a single iteration and allocate the saved computation to using a larger rank r.

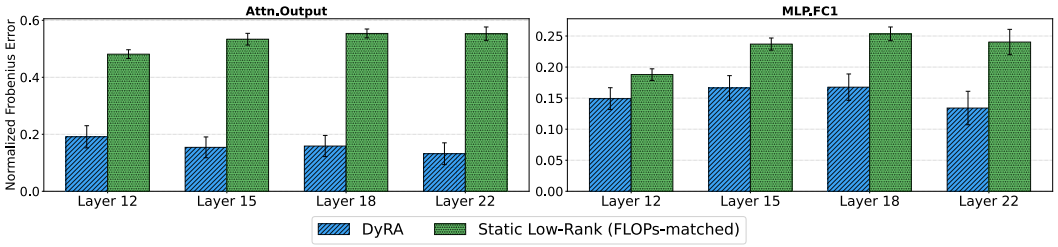


Figure 3: Remaining normalized Frobenius error of DyRA and a static low-rank baseline under a matched FLOPs budget. The error is measured between the original dense output and the approximated output across ViT layers.

This distinction becomes clearer in the isotropic case $\boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{I}$. In this case, $\boldsymbol{R}^{(1)} = \boldsymbol{V}_{k+1:k+r}^\top$, so DyRA reduces to the static rank-$(k+r)$ approximation. When $\boldsymbol{X}^\top \boldsymbol{X} \neq \boldsymbol{I}$, however, the current input reweights feature directions, and $\boldsymbol{R}^{(1)}$ generally differs from the weight-only basis. This is precisely where DyRA gains input adaptivity. The proof is provided in Appendix C.

Computational Cost We now quantify the computational overhead of constructing the input-adaptive subspace. Let $\boldsymbol{X} \in \mathbb{R}^{n \times c_1}$, $\boldsymbol{W}_{\text{res}} \in \mathbb{R}^{c_1 \times c_2}$, and let r denote the rank of the dynamic residual correction. As shown in Eq. 7, the left-factor update multiplies $\boldsymbol{X}$ by a slim $c_1 \times r$ matrix, costing $O(nc_1 r)$. The right-factor update projects the residual product onto the r-dimensional subspace defined by $\boldsymbol{L}^{(1)}$, with dominant cost $O(nc_1 r + c_1 c_2 r)$. Including the final multiplication $\boldsymbol{L}^{(1)}\boldsymbol{R}^{(1)}$, the dynamic residual correction costs $O(nc_1 r + c_1 c_2 r + nc_2 r)$. Combined with the $O(nk(c_1 + c_2))$ cost of the rank-k static approximation, DyRA totals $O\big(nk(c_1 + c_2) + r\big(n(c_1 + c_2) + c_1 c_2\big)\big)$. Since r is typically much smaller than c_1 and c_2, this remains substantially cheaper than the dense product $\boldsymbol{XW}$, which costs $O(nc_1 c_2)$. Compared with simply increasing the static rank to $k + r$, DyRA incurs an extra $O(c_1 c_2 r)$ term, which is the cost of constructing the input-adaptive subspace.

FLOPs-Matched Comparison Given this overhead, we examine whether DyRA's gains persist under a matched computational budget. To this end, we compare DyRA against a FLOPs-matched static low-rank weight approximation baseline on ViT. Starting from a rank-k approximation of $\boldsymbol{W}$, DyRA applies a rank-r dynamic correction to reduce the remaining output error. For the static baseline, we instead increase the rank from k to $k + r + r'$, where the additional rank r' accounts for the $O(c_1 c_2 r)$ overhead of the dynamic correction, so that both methods use the same FLOPs budget [4]. Figure 3 shows the remaining normalized Frobenius error across different ViT layers. Under this FLOPs-matched setting, DyRA consistently achieves lower error than the static low-rank baseline. These results show that, under the same computational budget, input-adaptive residual subspaces approximate the output more faithfully than higher-rank static approximations. Additional results for the remaining attention and MLP layers are provided in Appendix E.4.

2.4 Implementation of DyRA

To minimize GPU memory traffic, we implement DyRA with custom Triton [36] kernels. Rather than materializing the reconstructed residual $\boldsymbol{LR}$ in HBM, we fuse the low-rank reconstruction with the output merge: given $\boldsymbol{L} \in \mathbb{R}^{B \times T \times r}$ and $\boldsymbol{R} \in \mathbb{R}^{B \times r \times N}$, the kernel accumulates $\alpha \boldsymbol{LR}$ on-chip and directly writes $\boldsymbol{Y}_{\text{base}} + \alpha \boldsymbol{LR}$ to the output tensor. For linear layers, we further fuse the base projection, bias addition, and residual correction, computing $\boldsymbol{Y} = \boldsymbol{X}_{\text{tmp}} \boldsymbol{A}^\top + \boldsymbol{b} + \alpha \boldsymbol{LR}$ within a single Triton kernel. This eliminates intermediate tensors and reduces HBM accesses.

The ALS update requires solving an $r \times r$ symmetric positive definite system for each batch instance. For small r, we use a Triton Cholesky solver, keeping the factorization and triangular solves in on-chip cache/registers. For a larger r, Cholesky becomes too sequential to fully utilize GPU parallelism, so we use a Newton–Schulz iterative solver. This replaces dependency-heavy factorization with matrix-multiplication-based updates, improving the GPU's parallelism and tensor-core utilization.

[4] The matching is done at the level of exact FLOPs counts rather than asymptotic order, so r' also absorbs the constant and lower-order terms omitted in the $O(\cdot)$ notation.

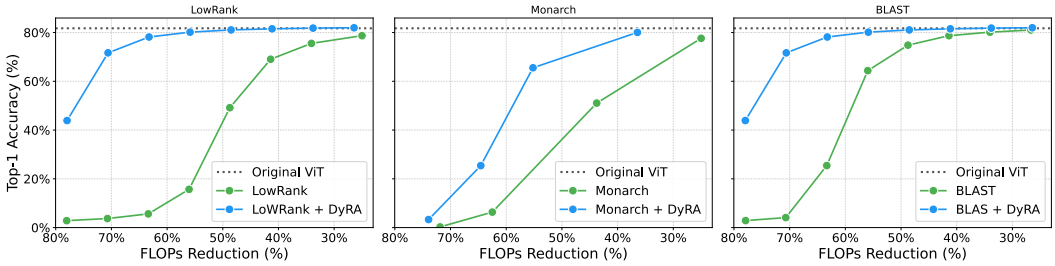


Figure 4: ImageNet classification accuracy at different FLOPs reduction levels for ViT-L. DyRA consistently improves the accuracy–efficiency trade-off across low-rank, Monarch, and BLAST.

3 Experimental Results

We validate DyRA by integrating it with several structured weight approximations, including low-rank, Monarch [11], and BLAST [22]. We first evaluate DyRA as a test-time model approximation/acceleration technique applied to pre-trained models (without any fine-tuning) across vision, language, and speech tasks, demonstrating consistent improvements in downstream performance under matched computational budgets (Section 3.1). We then measure end-to-end latency to assess whether these computational savings translate into practical speedups on a GPU (Section 3.2). Finally, we study settings where additional adaptation is available, showing that DyRA can be combined with post-training calibration or fine-tuning to further improve performance (Section 3.3). All experimental details are provided in Appendix D.

3.1 Test-Time Model Approximation and Acceleration Across Domains

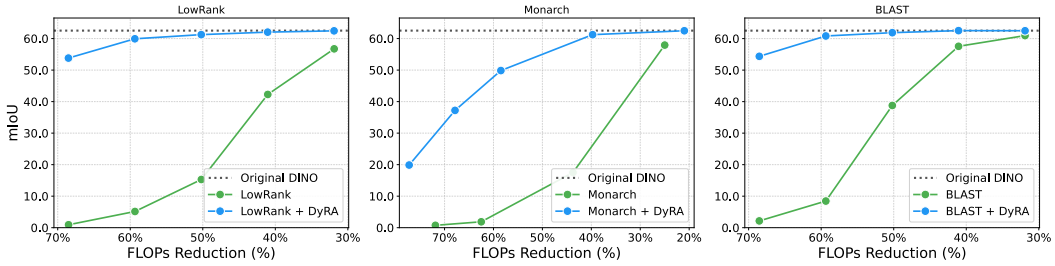
Image Classification with ViT We evaluate DyRA on ImageNet [13] classification using a pre-trained ViT-L model [14]. We apply structured weight approximations to all dense linear layers in both the attention and MLP blocks, and measure top-1 accuracy under different FLOPs budgets. For DyRA, we fix the dynamic residual rank to $r = 8$ and vary the budget of the static weight approximation to control the overall FLOPs. As shown in Figure 4, DyRA consistently improves the accuracy–efficiency trade-off across all structured baselines. In particular, at around 55% FLOPs reduction, the low-rank and BLAST baselines incur accuracy drops of 66.1% and 17.4%, respectively, whereas combining them with DyRA reduces the drops to 2.4% and 1.6%.

Dense Prediction Tasks with DINO Beyond image-level classification, we further show that DyRA preserves fine-grained representations required for dense prediction. To this end, we evaluate semantic segmentation on ADE20K [42] and depth estimation on NYUv2 [34] using DINOv3 [35]. We apply DyRA to all linear layers and compare it against the corresponding static weight approximation baselines. As shown in Figure 5, DyRA achieves a substantially better accuracy–efficiency trade-off across both tasks. In particular, for semantic segmentation at 60% FLOPs reduction, DyRA reduces mIoU by only 1.7 points relative to the original model, preserving most of its dense prediction performance. In contrast, the static weight approximation baseline suffers a 54.1 point drop in mIoU, indicating a severe degradation in fine-grained representations.

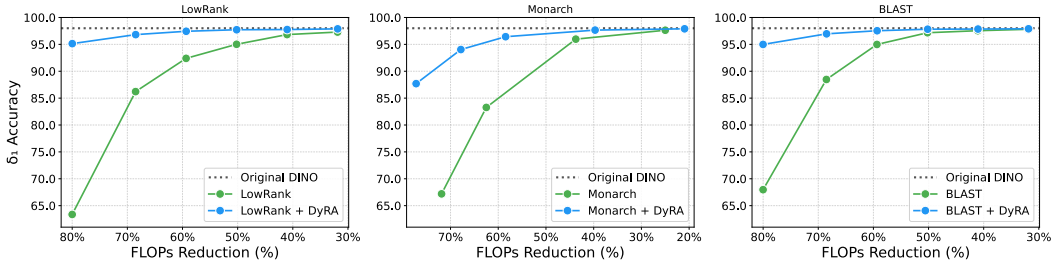
Speech Recognition We evaluate DyRA on automatic speech recognition using a pre-trained Wav2Vec 2.0 Large model [2]. We apply DyRA to all MLP layers and the query/key projections, and measure word error rate (WER) under different FLOPs reduction ratios. As shown in Table 1 on the LibriSpeech Clean test set [28], DyRA consistently achieves lower WER than static weight approximation baselines under matched computational budgets. These results demonstrate that, similar to the vision domains, combining DyRA with structured weight approximations is more effective than using weight approximation alone, and generalizes well to the audio domain.

Table 1: WER under different FLOPs reduction ratios (FRR).

FRR	Method	WER(↓)
0%	Original	1.92
25%	Low-rank	2.21
	Low-rank + DyRA	**2.08**
	BLAST	2.07
	BLAST + DyRA	**2.03**
50%	Low-rank	24.54
	Low-rank + DyRA	**4.89**
	BLAST	6.48
	BLAST + DyRA	**4.27**



(a) Semantic Segmentation on ADE20K.



(b) Depth Estimation on NYUv2.

Figure 5: DyRA vs. static weight approximations on dense prediction with DINOv3. DyRA achieves a superior accuracy–efficiency trade-off for semantic segmentation and depth estimation.

Language Modeling We evaluate DyRA on language modeling using LLaDA-8B [27], a large language diffusion model. We apply DyRA to all MLP layers and the query/key projections, and evaluate performance on HumanEval [6] for code generation and GSM8K [10] for mathematical reasoning. As shown in Figure 7a, the low-rank baseline performs poorly on both tasks, with accuracy below 4%. In contrast, DyRA improves performance by more than 20 percentage points in both cases under the same computational budget, achieving results substantially closer to the original model. These results demonstrate that DyRA makes structured weight approximation usable for large language diffusion models, even when the FLOPs-matched low-rank baseline largely fails.

3.2 Runtime Analysis

We evaluate whether the computational savings of DyRA translate into actual end-to-end speedups on a GPU. To this end, we apply DyRA to DINOv3 and compare it against static weight approximation baselines, including low-rank and BLAST. Following Section 2.4, we use the fused Triton kernels for DyRA. For the weight-only baselines, we use the faster option between the optimized structured kernels in [1] and `torch.compile()` optimization of PyTorch [29]. We measure the end-to-end latency of depth estimation on the NVIDIA A40 GPU.

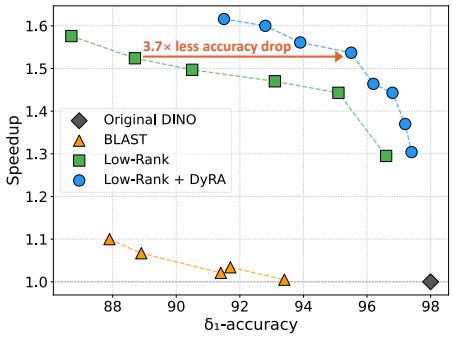
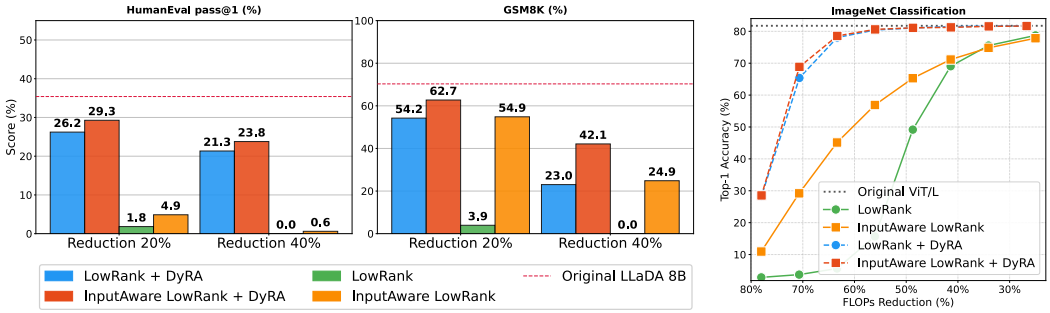


Figure 6: End-to-end latency vs. accuracy on DINOv3 depth estimation.

Figure 6 shows the relative speedup across different depth-estimation accuracy levels. DyRA achieves a $1.3\times$ end-to-end speedup with only a small accuracy degradation of about 0.6%, demonstrating that its FLOPs reduction translates into practical runtime improvement. Compared with the static low-rank baseline, DyRA provides a substantially better accuracy–latency trade-off. It achieves larger speedups at similar accuracy levels, while the low-rank baseline incurs a $3.7\times$ larger accuracy drop to reach a $1.52\times$ speedup. Overall, these results show that DyRA provides practical speedups on the NVIDIA A40 GPU while maintaining a more favorable balance between speed and accuracy than static weight approximations.



(a) LLaDA 8B on HumanEval and GSM8K

(b) ViT-L on ImageNet-1K

Figure 7: Comparison between DyRA and input-aware decomposition on LLaDA-8B and ViT-L. DyRA generally achieves better performance, and can further improve results by using input-aware decomposition as the static approximation.

3.3 Further Evaluations with Augmented Methods

Comparison with Input-aware Decomposition We compare DyRA with input-aware decomposition [38, 40], a post-training method that leverages calibration data to compute low-rank weight factors better aligned with output activations. Figure 7 reports results on LLaDA for HumanEval and GSM8K, and on ViT for ImageNet classification. Compared with vanilla low-rank decomposition, input-aware decomposition recovers part of the performance loss across these settings, confirming that activation information is useful for structured approximation.

However, since the decomposed weights are fixed after calibration, input-aware decomposition cannot adapt to the activations of each test sample. DyRA provides this missing input-dependent correction at inference time, achieving a better accuracy–efficiency trade-off than input-aware decomposition alone. Moreover, because DyRA operates as a correction on top of the structured computation, it can be combined with input-aware decomposition. In LLaDA, combining DyRA with input-aware decomposition further improves performance on both HumanEval and GSM8K. Similarly, on ImageNet classification at around 70% FLOPs reduction, combining DyRA with input-aware decomposition further improves accuracy from 65.4% to 68.8% over DyRA with vanilla low-rank decomposition, showing that DyRA can further benefit from stronger calibrated static approximations.

Fine-tuning Results Beyond combining DyRA with input-aware decomposition, we further evaluate DyRA under direct fine-tuning on ImageNet classification. In this experiment, we replace all linear layers of a ViT-L model with DyRA-augmented low-rank approximations and fine-tune the resulting model on ImageNet for 15 epochs. For comparison, we fine-tune a low-rank weight-only baseline under the same setting. As shown in Figure 8, consistent with the test-time approximation evaluation in prior subsections, DyRA with fine-tuning outperforms the static weight approximation baseline across different FLOPs budgets. Notably, at around 60% FLOPs reduction, DyRA even slightly surpasses the original dense model while using substantially fewer FLOPs. These results suggest that DyRA remains effective after fine-tuning.

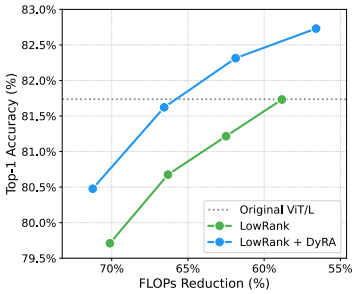


Figure 8: Fine-tuning results on ImageNet classification with ViT-L.

Ablation Study We analyze key components of DyRA: the residual rank r and the number of ALS iterations t for an ablation study. We conduct experiments on ViT image classification by combining DyRA with a low-rank weight approximation and varying each hyperparameter. Figure 9 (left) reports the effect of the residual rank, evaluated with $r \in \{4, 8, 16, 24\}$. DyRA consistently outperforms the corresponding low-rank static weight approximation baseline across all choices of r, showing that allocating even a small budget to the dynamic residual is beneficial and that performance is relatively robust to the choice of r. We then evaluate the effect of the ALS iteration count with $t \in \{1, 1.5, 2\}$ in Figure 9 (right), where $t = 1.5$ denotes the update sequence L, R, L. It shows that all configurations substantially improve over the static approximation baseline. Notably, $t = 1$ already achieves strong

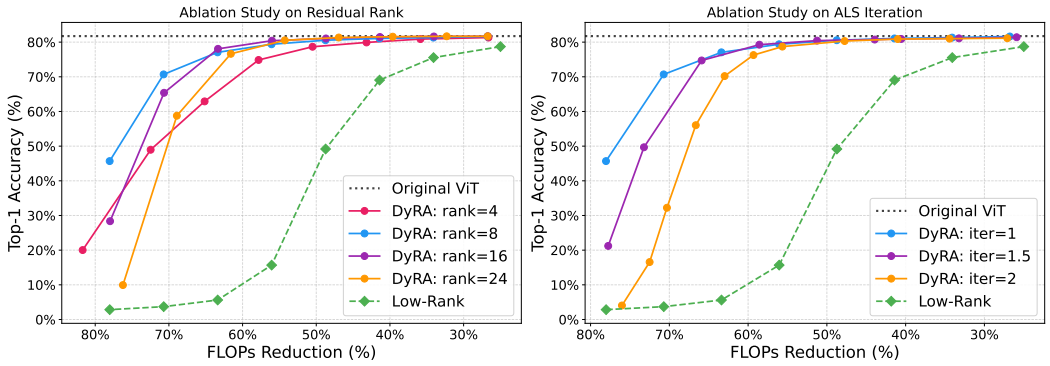


Figure 9: Ablation study on ImageNet classification with ViT-L. Left: DyRA with different residual ranks r. Right: DyRA with different ALS iteration counts t.

performance, suggesting that most of the gain comes from a single iteration. This supports our default choice of $t = 1$, which provides a favorable trade-off between accuracy and computational cost.

4 Related Works

Structured Matrices for Efficient DNN Inference Structured matrices have been widely used to reduce the cost of dense linear layers in deep neural networks. Prior studies [3, 16, 18, 19] have observed that pretrained weight matrices often exhibit low-rank structure, motivating low-rank approximation as a simple compression strategy [17]. More expressive block-structured variants, such as Monarch [11] and BLAST [22], improve the trade-off between efficiency and accuracy, while methods such as GBLR [21] and Einsum Linear [30] learn efficient weight structures directly. Another line of work incorporates activation information from calibration data to obtain input-aware weight decompositions [7, 38–40]. However, these methods still produce a static approximation before inference, and cannot adapt to the sample-dependent error induced by the current input. Our method addresses this limitation by approximating the realized output error on the fly for each input. Importantly, our method can be combined with advanced structured or input-aware decompositions to obtain a more faithful matrix multiplication under a fixed computational budget.

Low-Rank Approximation via Alternating Least Squares Alternating least squares (ALS) has been studied as an efficient method for low-rank approximation and reconstruction, as it reduces a rank-constrained problem to a sequence of tractable least-squares subproblems. Prior work has used ALS for low-rank matrix reconstruction from underdetermined measurements [41], structured low-rank recovery with ADMM-style updates [23], and spectral analyses connected to orthogonal iteration [12]. In contrast, we use ALS to approximate matrix products directly, targeting efficient matrix multiplication rather than reconstruction of a single matrix.

5 Conclusion

This work shows that the accuracy–efficiency trade-off of matrix multiplication in pre-trained DNNs can be improved by approximating the realized output product, rather than only the weight matrix. We formulate this output-domain approximation as a low-rank factorization problem and solve it efficiently with ALS during model inference. Building on this formulation, we propose DyRA, which augments static weight approximations with an input-adaptive residual correction. Our experiments show that this correction consistently improves the accuracy–efficiency trade-off across low-rank, Monarch, and BLAST approximations. In particular, DyRA preserves dense prediction performance under aggressive FLOPs reduction and translates these savings into practical speedups on a GPU.

Limitations and Future Work. DyRA is designed to accelerate dense matrix multiplication by combining dynamic residual approximation with efficient structured weight approximations. As a result, it is most effective in regimes with large matrix–matrix multiplications, whereas token-by-token vector–matrix workloads such as autoregressive decoding are not its primary target application.

Nevertheless, DyRA remains applicable to the prefill stage of autoregressive language models, where the full prompt induces large matrix multiplications. We provide supporting results in the supplementary material (Appendix E.1).

Finally, as a test-time acceleration approach, DyRA offers additional flexibility compared to static approximation methods whose rank must be fixed offline. In principle, the dynamic residual rank can be adjusted based on the input or computational budget at inference time. Exploring such adaptive rank allocation is left for future work.

Acknowledgments

This work was in part supported by MSTAR automotive microelectronics research award #N036453, and also by the Center on Cognitive Multispectral Sensors (COGNISENSE), one of seven centers in Joint University Microelectronics Program (JUMP) 2.0, a Semiconductor Research Corporation (SRC) program sponsored by Defense Advanced Research Projects Agency (DARPA). Daewon Chae would like to acknowledge the support from the Hyundai Motor Chung Mong-Koo Foundation.

References

[1] Pierre Abillama, Changwoo Lee, Juechu Dong, David Blaauw, Dennis Sylvester, and Hun-Seok Kim. Memory-efficient acceleration of block low-rank foundation models on resource constrained gpus. *arXiv preprint arXiv:2512.20861*, 2025.

[2] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

[3] Laura Balzano, Tianjiao Ding, Benjamin D Haeffele, Soo Min Kwon, Qing Qu, Peng Wang, Zhangyang Wang, and Can Yaras. An overview of low-rank structures in the training and adaptation of large models. *arXiv preprint arXiv:2503.19859*, 2025.

[4] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[5] Beidi Chen, Tri Dao, Kaizhao Liang, Jiaming Yang, Zhao Song, Atri Rudra, and Christopher Re. Pixelated butterfly: Simple and efficient sparse training for neural network models. In *International Conference on Learning Representations*, 2022.

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[7] Patrick Chen, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. Drone: Data-aware low-rank compression for large nlp models. *Advances in neural information processing systems*, 34: 29321–29334, 2021.

[8] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)*, pages 2924–2936, 2019.

[9] 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. *arXiv preprint arXiv:1803.05457*, 2018.

[10] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[11] Tri Dao, Beidi Chen, Nimit S Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pages 4690–4721. PMLR, 2022.

[12] Achiya Dax. The equivalence between orthogonal iterations and alternating least squares. *ADVANCES IN LINEAR ALGEBRA & MATRIX THEORY.*, 10(2):7–21, 2020.

[13] 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*, pages 248–255. Ieee, 2009.

[14] 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.

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

[16] Tomer Galanti, Zachary S Siegel, Aparna Gupte, and Tomaso A Poggio. SGD with weight decay secretly minimizes the ranks of your neural networks. In *The Second Conference on Parsimony and Learning (Proceedings Track)*, 2025.

[17] Habib Hajimolahoseini, Walid Ahmed, Mehdi Rezagholizadeh, Vahid Partovinia, and Yang Liu. Strategies for applying low rank decomposition to transformer-based models. In *36th Conference on Neural Information Processing Systems (NeurIPS2022)*, volume 6, 2022.

[18] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

[19] 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.

[20] Jacob Kahn, Morgane Riviere, Weiyi Zheng, Evgeny Kharitonov, Qiantong Xu, Pierre-Emmanuel Mazaré, Julien Karadayi, Vitaliy Liptchinsky, Ronan Collobert, Christian Fuegen, et al. Libri-light: A benchmark for asr with limited or no supervision. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7669–7673. IEEE, 2020.

[21] Changwoo Lee and Hun-Seok Kim. Differentiable learning of generalized structured matrices for efficient deep neural networks. In *The Twelfth International Conference on Learning Representations*, 2024.

[22] Changwoo Lee, Soo Min Kwon, Qing Qu, and Hun-Seok Kim. Blast: Block-level adaptive structured matrices for efficient deep neural network inference. *Advances in Neural Information Processing Systems*, 37:14996–15027, 2024.

[23] Kezhi Li, Martin Sundin, Cristian R Rojas, Saikat Chatterjee, and Magnus Jansson. Alternating strategies are good for low-rank matrix reconstruction. *arXiv preprint arXiv:1407.3410*, 2014.

[24] Chi-Heng Lin, Shangqian Gao, James Seale Smith, Abhishek Patel, Shikhar Tuli, Yilin Shen, Hongxia Jin, and Yen-Chang Hsu. ModeGPT: Modular decomposition for large language model compression. In *The Thirteenth International Conference on Learning Representations*, 2025.

[25] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[26] 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 *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2381–2391, 2018.

[27] Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, JUN ZHOU, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026.

[28] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

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

[30] Andres Potapczynski, Shikai Qiu, Marc Finzi, Christopher Ferri, Zixi Chen, Micah Goldblum, C Bayan Bruss, Christopher De, and Andrew G Wilson. Searching for efficient linear layers over a continuous space of structured matrices. *Advances in Neural Information Processing Systems*, 37:3857–3881, 2024.

[31] Shikai Qiu, Andres Potapczynski, Marc Anton Finzi, Micah Goldblum, and Andrew Gordon Wilson. Compute better spent: Replacing dense layers with structured matrices. In *Forty-first International Conference on Machine Learning*, 2024.

[32] Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. Imagenet-21k pretraining for the masses. *arXiv preprint arXiv:2104.10972*, 2021.

[33] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[34] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European conference on computer vision*, pages 746–760. Springer, 2012.

[35] Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. Dinov3. *arXiv preprint arXiv:2508.10104*, 2025.

[36] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.

[37] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[38] 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*, 2025.

[39] Hao Yu and Jianxin Wu. Compressing transformers: Features are low-rank, but weights are not! In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 11007–11015, 2023.

[40] 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. *arXiv preprint arXiv:2312.05821*, 2023.

[41] Dave Zachariah, Martin Sundin, Magnus Jansson, and Saikat Chatterjee. Alternating least-squares for low-rank matrix reconstruction. *IEEE Signal Processing Letters*, 19(4):231–234, 2012.

[42] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 633–641, 2017.

A DyRA Algorithm

Algorithm 1 summarizes the overall DyRA procedure, including offline preparation and inference-time dynamic residual correction.

Algorithm 1 DyRA with One-Step ALS Residual Approximation

procedure OFFLINE PREPARATION($\boldsymbol{W}, k, r, \lambda$)
  Compute a structured approximation $\widehat{\boldsymbol{W}}$ of $\boldsymbol{W}$
  $\boldsymbol{W}_{\text{res}} \leftarrow \boldsymbol{W} - \widehat{\boldsymbol{W}}$
  Initialize and fix $\boldsymbol{R}^{(0)} \in \mathbb{R}^{r \times c_2}$
  Precompute $\boldsymbol{P}_L \leftarrow \left(\boldsymbol{W}_{\text{res}} \boldsymbol{R}^{(0)\top}\right) \left(\boldsymbol{R}^{(0)} \boldsymbol{R}^{(0)\top} + \lambda \boldsymbol{I}_r\right)^{-1}$
  return $\widehat{\boldsymbol{W}}, \boldsymbol{W}_{\text{res}}, \boldsymbol{P}_L$
end procedure

procedure INFERENCE($\boldsymbol{X}, \widehat{\boldsymbol{W}}, \boldsymbol{W}_{\text{res}}, \boldsymbol{P}_L, \lambda$)
  $\boldsymbol{Y}_{\text{base}} \leftarrow \boldsymbol{X}\widehat{\boldsymbol{W}}$
  $\boldsymbol{L}^{(1)} \leftarrow \boldsymbol{X}\boldsymbol{P}_L$
  $\boldsymbol{R}^{(1)} \leftarrow \left(\boldsymbol{L}^{(1)\top}\boldsymbol{L}^{(1)} + \lambda \boldsymbol{I}_r\right)^{-1} \left(\boldsymbol{L}^{(1)\top}\boldsymbol{X}\right) \boldsymbol{W}_{\text{res}}$
  $\widehat{\boldsymbol{Y}} \leftarrow \boldsymbol{Y}_{\text{base}} + \boldsymbol{L}^{(1)}\boldsymbol{R}^{(1)}$
  return $\widehat{\boldsymbol{Y}}$
end procedure

B Background on Structured Matrices

Low-Rank Approximation Low-rank approximation represents a weight matrix $\boldsymbol{W} \in \mathbb{R}^{d \times m}$ as the product of two smaller matrices,

$$\widehat{\boldsymbol{W}} = \boldsymbol{L}\boldsymbol{R}, \qquad \boldsymbol{L} \in \mathbb{R}^{d \times r}, \quad \boldsymbol{R} \in \mathbb{R}^{r \times m},$$

where $r \ll \min(d, m)$. A standard choice is the rank-r truncated singular value decomposition (SVD). Given $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, the approximation is

$$\widehat{\boldsymbol{W}} = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r^\top, \qquad \boldsymbol{L} = \boldsymbol{U}_r\boldsymbol{\Sigma}_r, \quad \boldsymbol{R} = \boldsymbol{V}_r^\top.$$

This approximation keeps the top-r singular components of $\boldsymbol{W}$, providing the best rank-r approximation under the Frobenius norm.

Monarch Matrix Monarch matrices [11] represent a weight matrix using two block-diagonal factors with an intermediate permutation. For $\boldsymbol{W} \in \mathbb{R}^{N \times N}$ with $N = b_1 b_2$, the approximation is written as

$$\widehat{\boldsymbol{W}} = \boldsymbol{P}\boldsymbol{L}\boldsymbol{P}^\top\boldsymbol{R}, \qquad \boldsymbol{L} = \operatorname{diag}(\boldsymbol{L}_1, \ldots, \boldsymbol{L}_{b_2}), \qquad \boldsymbol{R} = \operatorname{diag}(\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{b_1}),$$

where $\boldsymbol{L}_i \in \mathbb{R}^{b_1 \times b_1}$ for $i \in [b_2]$ and $\boldsymbol{R}_j \in \mathbb{R}^{b_2 \times b_2}$ for $j \in [b_1]$. Here, $\boldsymbol{P} \in \mathbb{R}^{N \times N}$ is a fixed permutation matrix that reorders the block structure, so that the overall product induces a structured block-wise pattern in $\widehat{\boldsymbol{W}}$. In practice, the factors can be obtained by reshaping the dense weight into blocks, applying a truncated SVD to each block, and rearranging the resulting factors.

BLAST BLAST [22] represents a weight matrix using shared block-row and block-column bases. For $\boldsymbol{W} \in \mathbb{R}^{d \times m}$, we partition the matrix into a grid of blocks $\boldsymbol{W}_{i,j} \in \mathbb{R}^{p \times q}$. Each block is parameterized as

$$\widehat{\boldsymbol{W}}_{i,j} = \boldsymbol{U}_i\boldsymbol{S}_{i,j}\boldsymbol{V}_j^\top, \qquad \boldsymbol{U}_i \in \mathbb{R}^{p \times r}, \quad \boldsymbol{S}_{i,j} \in \mathbb{R}^{r \times r}, \quad \boldsymbol{V}_j \in \mathbb{R}^{q \times r}.$$

Here, $\boldsymbol{U}_i$ is shared by all blocks in the same block row, and $\boldsymbol{V}_j$ is shared by all blocks in the same block column. The middle factor $\boldsymbol{S}_{i,j}$ is specific to each block and controls how the shared row and column bases are combined. This parameterization captures low-dimensional structure at the block level while reusing bases across the matrix. In practice, the BLAST factors are obtained by minimizing the reconstruction error between $\boldsymbol{W}$ and $\widehat{\boldsymbol{W}}$.

Input-Aware Decomposition Input-aware decomposition aims to better preserve output activations by incorporating calibration inputs into the weight approximation. Given a calibration set $\mathcal{S} = \{\boldsymbol{X}_s\}_{s=1}^{|\mathcal{S}|}$, these methods approximate $\boldsymbol{W}$ by reducing the output-space discrepancy measured on the calibration set, $\sum_{\boldsymbol{X}_s \in \mathcal{S}} \|\boldsymbol{X}_s \boldsymbol{W} - \boldsymbol{X}_s \widehat{\boldsymbol{W}}\|_F^2$. A common formulation introduces an invertible transformation matrix $\boldsymbol{S}$ and rewrites the weight as $\boldsymbol{W} = (\boldsymbol{W}\boldsymbol{S})\boldsymbol{S}^{-1}$. Truncated SVD is then applied to the transformed weight,

$$\boldsymbol{W}\boldsymbol{S} \approx \boldsymbol{U}_r \boldsymbol{\Sigma}_r \boldsymbol{V}_r^\top,$$

yielding the approximation

$$\widehat{\boldsymbol{W}} = \boldsymbol{U}_r \boldsymbol{\Sigma}_r \boldsymbol{V}_r^\top \boldsymbol{S}^{-1}. \tag{9}$$

The transform $\boldsymbol{S}$ determines how calibration activations influence the decomposition. For example, $\boldsymbol{S}$ can be chosen as a diagonal scaling matrix based on channel-wise activation magnitudes [40], or as a Cholesky factor of the input covariance matrix computed from the calibration set [38, 40].

C Proof of Proposition 1

Proof. We start from the one-step DyRA approximation in Eq. 6,

$$\boldsymbol{X}\boldsymbol{W} \approx \boldsymbol{X}\widehat{\boldsymbol{W}} + \boldsymbol{L}^{(1)}\boldsymbol{R}^{(1)}. \tag{10}$$

In the low-rank case, the static approximation is

$$\widehat{\boldsymbol{W}} = \boldsymbol{U}_{1:k}\boldsymbol{\Sigma}_{1:k}\boldsymbol{V}_{1:k}^\top, \tag{11}$$

and the residual is

$$\boldsymbol{W}_{\text{res}} = \boldsymbol{W} - \widehat{\boldsymbol{W}} = \boldsymbol{U}_{k+1:}\boldsymbol{\Sigma}_{k+1:}\boldsymbol{V}_{k+1:}^\top. \tag{12}$$

We initialize the right factor as $\boldsymbol{R}^{(0)} = \boldsymbol{V}_{k+1:k+r}^\top$. Since the singular vectors are orthonormal, $\boldsymbol{R}^{(0)}\boldsymbol{R}^{(0)\top} = \boldsymbol{I}$. Substituting this into the left-factor update in Eq. 4 yields

$$\begin{aligned}
\boldsymbol{L}^{(1)} &= \boldsymbol{X}\left(\boldsymbol{W}_{\text{res}}\boldsymbol{R}^{(0)\top}\right)\left(\boldsymbol{R}^{(0)}\boldsymbol{R}^{(0)\top}\right)^{-1} \\
&= \boldsymbol{X}\left(\boldsymbol{U}_{k+1:}\boldsymbol{\Sigma}_{k+1:}\boldsymbol{V}_{k+1:}^\top\boldsymbol{V}_{k+1:k+r}\right) \\
&= \boldsymbol{X}\boldsymbol{U}_{k+1:k+r}\boldsymbol{\Sigma}_{k+1:k+r}.
\end{aligned} \tag{13}$$

Let $\boldsymbol{A}_r^{\text{res}} = \boldsymbol{U}_{k+1:k+r}\boldsymbol{\Sigma}_{k+1:k+r}$, so that $\boldsymbol{L}^{(1)} = \boldsymbol{X}\boldsymbol{A}_r^{\text{res}}$.

Substituting this into the right-factor update in Eq. 5, we obtain

$$\begin{aligned}
\boldsymbol{R}^{(1)} &= \left(\boldsymbol{L}^{(1)\top}\boldsymbol{L}^{(1)}\right)^{-1}\left(\boldsymbol{L}^{(1)\top}\boldsymbol{X}\right)\boldsymbol{W}_{\text{res}} \\
&= \left(\boldsymbol{A}_r^{\text{res}\top}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{A}_r^{\text{res}}\right)^{-1}\boldsymbol{A}_r^{\text{res}\top}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{W}_{\text{res}}.
\end{aligned} \tag{14}$$

Plugging the resulting one-step factors into Eq. 6, we obtain

$$\begin{aligned}
\boldsymbol{X}\boldsymbol{W} &\approx \boldsymbol{X}\widehat{\boldsymbol{W}} + \boldsymbol{L}^{(1)}\boldsymbol{R}^{(1)} \\
&= \boldsymbol{X}\boldsymbol{U}_{1:k}\boldsymbol{\Sigma}_{1:k}\boldsymbol{V}_{1:k}^\top + \boldsymbol{X}\boldsymbol{U}_{k+1:k+r}\boldsymbol{\Sigma}_{k+1:k+r}\boldsymbol{R}^{(1)} \\
&= \boldsymbol{X}\left(\boldsymbol{U}_{1:k}\boldsymbol{\Sigma}_{1:k}\boldsymbol{V}_{1:k}^\top + \boldsymbol{U}_{k+1:k+r}\boldsymbol{\Sigma}_{k+1:k+r}\boldsymbol{R}^{(1)}\right) \\
&= \boldsymbol{X}\begin{bmatrix}\boldsymbol{U}_{1:k}\boldsymbol{\Sigma}_{1:k} & \boldsymbol{U}_{k+1:k+r}\boldsymbol{\Sigma}_{k+1:k+r}\end{bmatrix}\begin{bmatrix}\boldsymbol{V}_{1:k}^\top \\ \boldsymbol{R}^{(1)}\end{bmatrix}.
\end{aligned} \tag{15}$$

This proves the stated form.

Finally, if $\boldsymbol{X}^\top\boldsymbol{X} = \boldsymbol{I}$, then

$$\boldsymbol{R}^{(1)} = \left(\boldsymbol{A}_r^{\text{res}\top}\boldsymbol{A}_r^{\text{res}}\right)^{-1}\boldsymbol{A}_r^{\text{res}\top}\boldsymbol{W}_{\text{res}}. \tag{16}$$

Using orthogonality of singular vectors,

$$\boldsymbol{A}_r^{\text{res}\top}\boldsymbol{A}_r^{\text{res}} = \boldsymbol{\Sigma}_{k+1:k+r}^2, \qquad \boldsymbol{A}_r^{\text{res}\top}\boldsymbol{W}_{\text{res}} = \boldsymbol{\Sigma}_{k+1:k+r}^2\boldsymbol{V}_{k+1:k+r}^\top,$$

which gives $\boldsymbol{R}^{(1)} = \boldsymbol{V}_{k+1:k+r}^\top$. Thus, DyRA reduces to the static rank-$(k+r)$ truncated SVD in the isotropic case. □

D Experimental Details

In this section, we provide additional details on the experimental setup. Unless otherwise specified, we use one ALS iteration for DyRA across all experiments. We initialize $\boldsymbol{R}^{(0)}$ with the top right singular vectors of the residual weight $\boldsymbol{W}_{\text{res}} = \boldsymbol{W} - \widehat{\boldsymbol{W}}$, following the subspace interpretation in Section 2.3. This initialization is performed once offline before inference and does not introduce input-dependent overhead.

Image Classification For image classification, we evaluate DyRA on ImageNet-1K using a ViT-L/32 model[5] [14] pretrained on ImageNet-21K [32]. We apply DyRA to all linear layers in the model, including the MLP layers and the query, key, value, and output projections in the attention blocks. We fix the rank of the dynamic residual approximation to $r = 8$ and control the FLOPs by varying the configuration of the static weight approximation.

For low-rank approximation, we vary the static rank over $\{100, 150, 200, 250, 300, 350, 400, 450\}$. For Monarch, we vary the Monarch block size over $\{3, 4, 6, 8\}$. For BLAST, we fix the block partition to 4 and vary the static rank over $\{100, 150, 200, 250, 300, 350, 400, 450\}$. For each structured approximation, we construct the corresponding weight-only baseline by sweeping its configuration to match the FLOPs of DyRA.

Dense Prediction For dense prediction, we evaluate DyRA on semantic segmentation and depth estimation using a ViT-7B-based DINOv3 model [35]. We evaluate semantic segmentation on ADE20K [42] and depth estimation on NYUv2 [34]. For both tasks, we follow the image preprocessing and decoding protocol of DINOv3 [6]. We apply DyRA uniformly to all linear layers in the model.

We fix the rank of the dynamic residual approximation to $r = 48$ and vary the configuration of the static weight approximation to control the FLOPs reduction. For depth estimation, we sweep the low-rank approximation rank over $\{436, 750, 1000, 1250, 1500, 1750\}$, the BLAST rank over $\{433, 746, 995, 1244, 1492, 1741\}$, and the Monarch block size over $\{2, 3, 4, 6, 8\}$. For semantic segmentation, we sweep the low-rank approximation rank over $\{750, 1000, 1250, 1500, 1750\}$, the BLAST rank over $\{746, 995, 1244, 1492, 1741\}$, and the Monarch block size over $\{2, 3, 4, 6, 8\}$. The corresponding weight-only baselines are obtained by adjusting the same configuration parameters to match the FLOPs of DyRA.

Speech Recognition For speech recognition, we evaluate DyRA on Wav2Vec2-Large[7] [2], pretrained and fine-tuned on 960 hours of Libri-Light [20] and LibriSpeech [28]. We report word error rate (WER) on the LibriSpeech test set, following the standard audio preprocessing and CTC-based decoding protocol used for Wav2Vec2. DyRA is applied to the query and key projections in the attention blocks and to the two fully connected layers in each MLP block. For both DyRA and the corresponding weight-only baselines, rank parameters are selected by grid search over a predefined set of configurations under target FLOPs reduction ratios of 25% and 50%.

We summarize each DyRA configuration as $(r_{\text{MLP}}, r_{\text{attn}}, r_{\text{res}})$, where r_{MLP}, r_{attn}, and r_{res} denote the MLP rank, attention rank, and residual rank, respectively. The selected configurations are $(370, 210, 48)$ and $(200, 60, 36)$ for BLAST at 25% and 50% FLOPs reduction, respectively, and $(440, 160, 44)$ and $(200, 90, 36)$ for low-rank at 25% and 50% FLOPs reduction, respectively.

Language Modeling For language modeling, we evaluate DyRA on LLaDA-8B[8] [27], a large language diffusion model. We consider two benchmarks: HumanEval [6] for code generation and GSM8K [10] for mathematical reasoning. We apply low-rank-based DyRA to the query and key projections in the attention blocks and to the MLP layers. Following prior observations that query and key projections in LLMs are more amenable to compression [19], we use more aggressive FLOPs reduction for the query and key projections than for the MLP layers.

[5] https://huggingface.co/google/vit-large-patch32-384
[6] https://github.com/facebookresearch/dinov3
[7] https://huggingface.co/facebook/wav2vec2-large-960h-lv60-self
[8] https://huggingface.co/GSAI-ML/LLaDA-8B-Base

Table 2: Perplexity comparison on language modeling benchmarks without fine-tuning. Lower is better. Bold indicates the best performance under the same compression ratio.

CR	Method	WikiText	PG-19	C4
0%	Dense	**9.38**	**10.07**	**7.01**
20%	Low-Rank	16.00(+6.62)	15.93(+5.86)	10.81(+3.80)
	BLAST	12.94(+3.56)	13.05(+2.98)	9.26(+2.25)
	Low-Rank + DyRA	**12.03(+2.65)**	**12.48(+2.41)**	**8.75(+1.74)**

Table 3: Performance comparison across multiple-choice reasoning benchmarks. FR denotes the FLOP reduction ratio. Values in parentheses indicate accuracy changes relative to the dense model. Bold indicates the best performance under the same compression ratio.

FR	Method	PIQA	Wino	BoolQ	OBQA	ARC-e	ARC-c	Avg
0%	Dense	**81.66**	**74.90**	**77.03**	47.60	**80.18**	**50.85**	**68.70**
20%	Low-Rank	76.66(-5.00)	69.38(-5.52)	69.42(-7.61)	45.20(-2.40)	73.53(-6.65)	41.72(-9.13)	62.65(-6.05)
	BLAST	**78.56(-3.10)**	67.64(-7.26)	66.91(-10.12)	44.80(-2.80)	75.42(-4.76)	43.43(-7.42)	62.79(-5.91)
	Low-Rank + DyRA	77.69(-3.97)	**69.93(-4.97)**	**71.50(-5.53)**	**48.60(+1.00)**	**76.05(-4.13)**	**44.11(-6.74)**	**64.65(-4.05)**

For HumanEval, we use two DyRA configurations targeting 20% and 40% total FLOPs reduction. At 20% reduction, we set the query/key static rank to 840 with residual rank 40, and the MLP static rank to 1000 with residual rank 150. At 40% reduction, we set the query/key static rank to 300 with residual rank 50, and the MLP static rank to 300 with residual rank 150.

For GSM8K, we again use two configurations targeting 20% and 40% total FLOPs reduction. At 20% reduction, we set the query/key static rank to 512 with residual rank 240, and the MLP static rank to 1024 with residual rank 380. At 40% reduction, we set the query/key static rank to 320 with residual rank 180, and the MLP static rank to 630 with residual rank 270.

Fine-tuning ViT on ImageNet For the fine-tuning experiment, we use a ViT-L/32 model pretrained on ImageNet-21K and fine-tune it on ImageNet-1K. We apply DyRA uniformly to all linear layers, including the MLP layers and all attention projections. Each model is fine-tuned for 15 epochs using the AdamW optimizer [25] with a learning rate of 2×10^{-5} and a batch size of 1024.

We fix the dynamic residual approximation rank to $r = 16$ and vary the static weight rank over $\{96, 128, 160, 196\}$, training a separate DyRA model for each static rank. For each DyRA configuration, we construct a FLOPs-matched static low-rank baseline by adjusting its rank and fine-tune the corresponding baseline model using the same training setup.

Comparison with Input-Aware Decomposition. For comparison with prior input-aware decomposition methods, we use the approaches proposed in [38, 40]. In Eq. 9, we instantiate $\boldsymbol{S}$ as the Cholesky factor of the input covariance matrix computed from calibration activations. Since input-aware decompositions can be sensitive to the choice of calibration data, we use the corresponding training split of each evaluation dataset to provide a strong baseline setting. Specifically, we use the ImageNet training set for ImageNet experiments and the GSM8K training set for GSM8K evaluation. Because HumanEval consists only of evaluation problems without a dedicated training split, we use the GSM8K training set as the calibration set for HumanEval experiments. For each target FLOPs reduction ratio, the ranks are kept identical to those of the corresponding DyRA configurations used in the language modeling experiments, except that the static low-rank factors are initialized using input-aware decomposition.

E Additional Experiment Results

E.1 Autoregressive LLM Prefilling

In addition to the diffusion language model setting, we examine whether DyRA can also be applied to autoregressive LLMs, particularly in the prefilling regime where computation is dominated by large matrix multiplications over the input sequence. Rather than measuring decoding throughput,

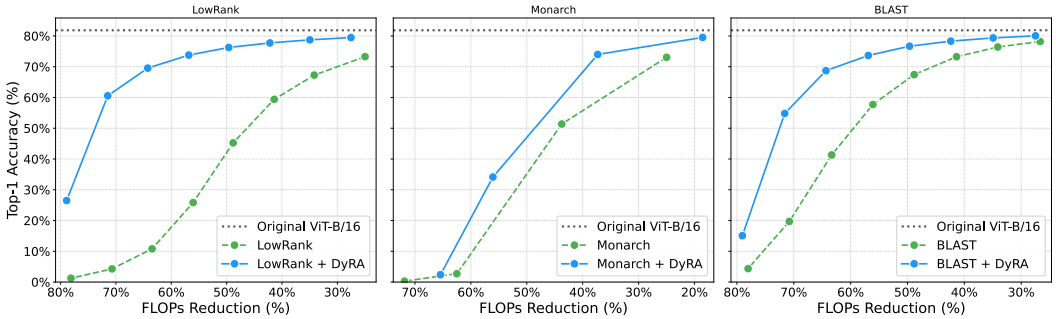


Figure 10: Top-1 accuracy versus FLOPs reduction on ViT-B/16. Across low-rank, Monarch, and BLAST approximations, combining DyRA consistently improves the accuracy–efficiency trade-off over static weight approximation baselines.

we evaluate this setting through language modeling perplexity and multiple-choice reasoning benchmarks [4, 8, 9, 26, 33], where each prompt and candidate continuation is processed in a forward pass. To better reflect non-trivial prefilling workloads, we use few-shot evaluation prompts that provide extended input contexts for multiple-choice reasoning benchmarks.

We apply low-rank-based DyRA to the linear layers of LLaMA-7B [37], targeting a 20% reduction in matrix multiplication FLOPs. As shown in Tables 2 and 3, DyRA consistently improves over the corresponding low-rank approximation and also outperforms the BLAST baseline under the same FLOPs reduction. These results suggest that DyRA can be effectively applied to the prefilling stage of autoregressive LLMs.

E.2 Image Classification with ViT-B

To further examine whether DyRA remains effective across different model scales, we additionally evaluate image classification on ViT-B/16 [9]. Using the same experimental setup as in Section D, we combine DyRA with static weight approximations and compare it against FLOPs-matched static baselines. As shown in Figure 10, DyRA consistently improves the accuracy–efficiency trade-off over the corresponding static weight approximations. These results demonstrate that DyRA's gains are not specific to the larger ViT-L model used in the main paper, but also generalize to a smaller ViT backbone.

E.3 Convergence of ALS Residual Approximation

We provide additional details and results for the ALS convergence experiment discussed in Section 2.2. We randomly sample 4,096 images from the ImageNet validation set and use them to extract the input activations $\boldsymbol{X}$ at each linear layer of ViT-L. For each layer, we construct the residual weight $\boldsymbol{W}_{\text{res}} = \boldsymbol{W} - \widehat{\boldsymbol{W}}$, where $\widehat{\boldsymbol{W}}$ is obtained via a rank-196 truncated SVD of $\boldsymbol{W}$. We then apply ALS to approximate the residual product $\boldsymbol{X}\boldsymbol{W}_{\text{res}}$ with target rank $r = 16$.

At each ALS iteration t, we compute the relative Frobenius error between the residual product and its low-rank approximation: $\frac{\|\boldsymbol{X}\boldsymbol{W}_{\text{res}} - \boldsymbol{L}^{(t)}\boldsymbol{R}^{(t)}\|_F}{\|\boldsymbol{X}\boldsymbol{W}_{\text{res}}\|_F}$.

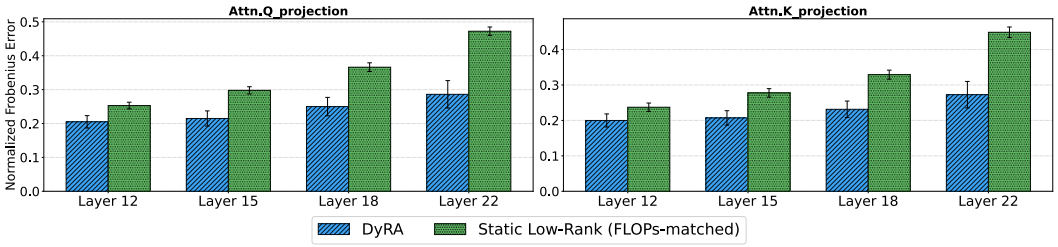
We compare this error with the optimal rank-r error obtained by applying SVD directly to $\boldsymbol{X}\boldsymbol{W}_{\text{res}}$.

Figure 12 reports the convergence results averaged over the sampled images across all ViT linear layers. ALS rapidly approaches the optimal rank-r error in nearly all layers, typically within two iterations.

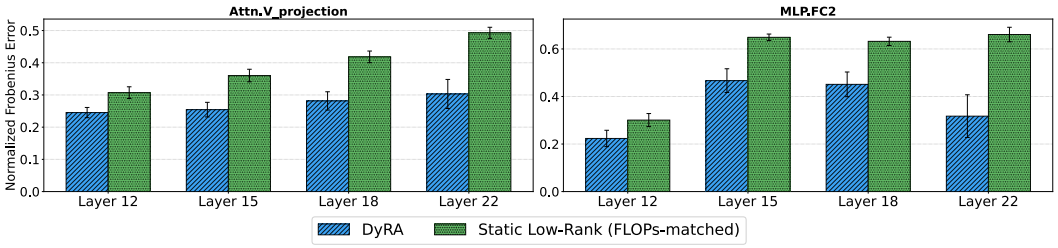
E.4 FLOPs-Matched Output Approximation Error

We provide additional details and results for the FLOPs-matched output error comparison discussed in Section 2.3. Using the same ViT-L and ImageNet validation setup as in Section E.3, we collect input activations $\boldsymbol{X}$ from 4,096 validation images for each target linear layer. For each layer, we

[9] https://huggingface.co/google/vit-base-patch16-224



(a) Attention query and key projections.



(b) Attention value projection and second MLP linear layer.

Figure 11: Additional FLOPs-matched output approximation error results on ViT-L sublayers omitted from the main paper. DyRA consistently achieves lower normalized Frobenius output error than the static low-rank baseline.

compute a rank-196 truncated SVD approximation $\widehat{\boldsymbol{W}}$ and apply a rank-16 DyRA correction to the residual product $\boldsymbol{X}(\boldsymbol{W} - \widehat{\boldsymbol{W}})$ using one ALS iteration.

For the FLOPs-matched static baseline, we increase the low-rank approximation rank so that its exact FLOPs match the total FLOPs of DyRA, including the dynamic correction overhead. We then compare the remaining normalized Frobenius output error, $\|\boldsymbol{X}\boldsymbol{W} - \widehat{\boldsymbol{Y}}\|_F / \|\boldsymbol{X}\boldsymbol{W}\|_F$, where $\widehat{\boldsymbol{Y}}$ denotes the approximated output.

Figure 11 reports additional results for the remaining sublayers not shown in the main text: the query, key, and value projections in attention, and the second MLP linear layer. Across these additional sublayers, DyRA consistently yields lower output approximation error than the FLOPs-matched higher-rank static low-rank baseline.

E.5 Sensitivity to $\boldsymbol{R}^{(0)}$ Initialization

In our experiments, we initialize $\boldsymbol{R}^{(0)}$ using the right singular vectors of $\boldsymbol{W}_{\text{res}}$, consistent with the analysis in Proposition 1. To evaluate the sensitivity of DyRA to this initialization choice, we additionally initialize $\boldsymbol{R}^{(0)}$ with random Gaussian values sampled from $\mathcal{N}(0, 1)$ using three different random seeds. We evaluate these variants under the same settings as our main experiments at 50% FLOPs reduction. Table 4 reports the results on ViT image classification and DINO semantic segmentation. Random initialization yields consistent performance across different seeds, with only minor degradation compared to SVD initialization. These results suggest that DyRA is robust to the choice of $\boldsymbol{R}^{(0)}$ initialization.

Table 4: Effect of $\boldsymbol{R}^{(0)}$ initialization on DyRA performance. Random initialization results are reported as mean $\pm$ standard deviation over three runs.

Model	Original	Vanilla Low-Rank	DyRA (SVD)	DyRA (Random)
ViT Acc. (↑)	81.74	49.17	80.55	80.19 ± 0.04
DINO mIoU (↑)	62.52	15.29	61.36	61.20 ± 0.05

E.6 Inference on Edge Devices

To evaluate the efficiency of DyRA on edge devices, we conduct a layer-wise latency analysis of DINOv3 on an NVIDIA Jetson Orin Nano 8GB. Specifically, we benchmark representative attention

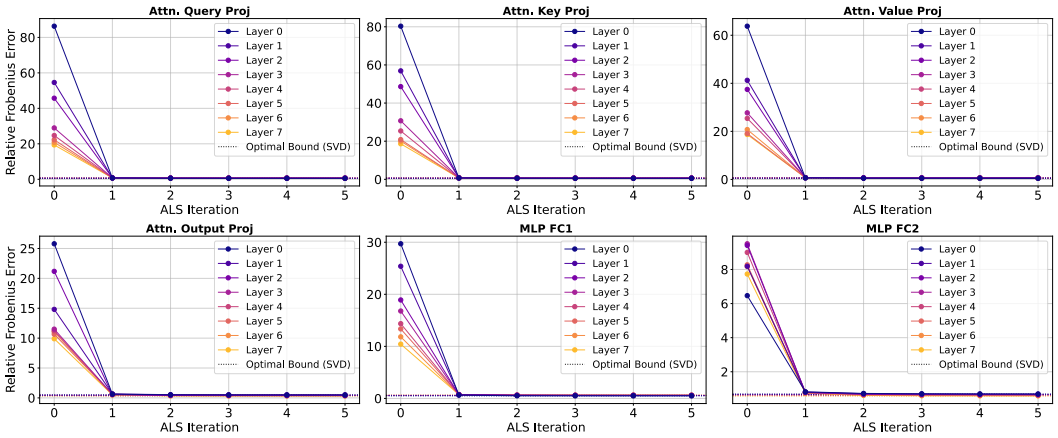
Table 5: Layer-wise inference performance of DyRA on NVIDIA Jetson Orin Nano 8GB using DINOv3 layers. Speedups are measured relative to the original dense implementations.

Layer	Weight Dim.	Static/Dynamic Rank	FLOPs Reduction	Speedup
Attn output Proj.	4096×4096	1280/16	35.8%	1.08×
Attn QKV Proj.	4096×12288	1280/16	57.2%	1.55×
MLP up Proj.	4096×8192	1280/16	51.8%	1.39×
MLP down Proj.	8192×4096	1280/16	51.6%	1.90×

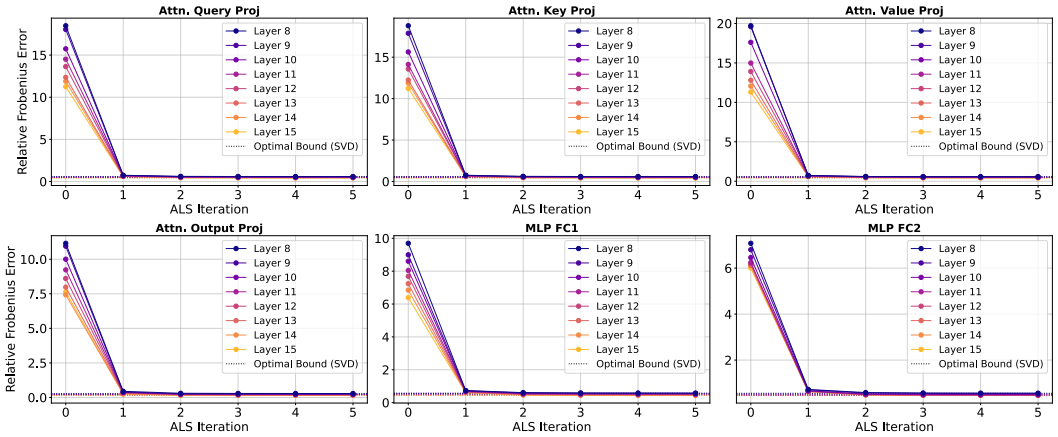
and MLP layers and compare their inference latency against the original dense implementations. As shown in Table 5, DyRA achieves speedups ranging from 1.08× to 1.90× across the evaluated layers, showing latency improvements on edge hardware.

F Broader Impacts

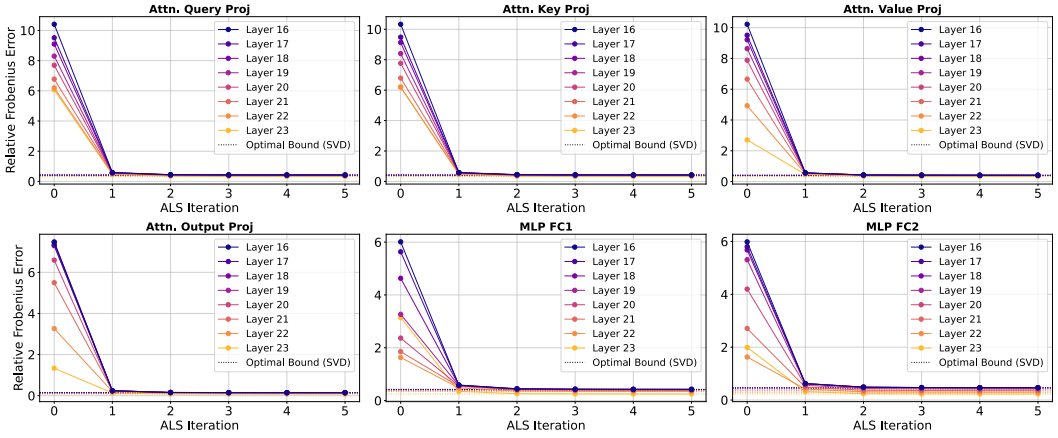
DyRA is a test-time inference acceleration method designed to reduce the computational cost of neural network inference. Its potential positive societal impact lies in improving the resource efficiency of AI systems, enabling model deployment under limited computational budgets and reducing the energy costs associated with inference. Since DyRA does not introduce a new application domain or change the intended use of the underlying model beyond approximation, we do not identify negative societal impacts specific to the proposed method. However, by making inference more efficient, DyRA may facilitate broader deployment of existing AI systems, and therefore may inherit the societal risks associated with those systems and their applications.



(a) Layers 0–7 of ViT-L.



(b) Layers 8–15 of ViT-L.



(c) Layers 16–23 of ViT-L.

Figure 12: Convergence of ALS-based residual product approximation across ViT-L linear layers.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly reflect DyRA's central contribution: improving structured matrix approximation by dynamically correcting input-dependent output errors at inference time.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a limitation discussion in the conclusion section.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The paper provides a proof of Proposition 1 in Appendix C.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide the experimental details needed to reproduce the main results in Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide the code and instructions in the supplemental material to reproduce the main experimental results.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We provide the experimental details needed to reproduce the main results, including hyperparameters, in Appendix D.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We report appropriate statistics for experiments involving randomness. For latency measurements, we report the average over 50 runs, and Figure 3 includes error bars.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We specify the GPU types used for latency measurements.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The research presented in this paper conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss both potential positive and negative societal impacts in Appendix F.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk models or datasets that require specific safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper properly credits the existing datasets and models used in the experiments and provides relevant links in the appendix.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide the code with accompanying documentation and reproduction instructions in the supplemental material.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method and experiments in this work do not involve LLMs as important or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.