ARAS: ADAPTIVE RANK-ALLOCATED SHARED SUBSPACES FOR LLM COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Deploying foundation models across diverse hardware platforms requires reducing model size without substantially degrading model quality. Existing low-rank compression methods often rely on global approximations, complete rows or columns, predefined spatial blocks, or fixed sharing patterns. We observe that Transformer weight matrices contain finer low-dimensional structure: spatially distant fine-grained weight blocks can often be represented by the same subspace. Based on this observation, we introduce ***A**daptive **R**ank-**A**llocated **S**hared Subspaces* (ARAS), a post-training compression framework that learns which fine-grained weight blocks should share a low-rank basis, independent of their locations within the matrix. ARAS further allocates rank adaptively using a small calibration set under a global storage budget. Across OPT models from 125M to 30B parameters and Llama models from 1B to 8B parameters and compression levels, ARAS reduces the overall model size by 67% while maintaining competitive perplexity and consistently outperforms the evaluated low-rank baselines.

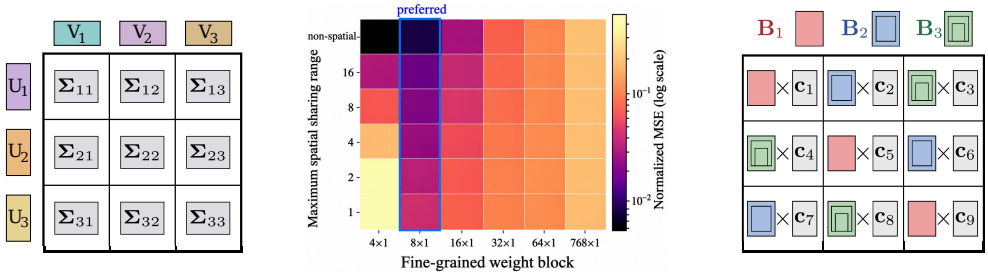
1 INTRODUCTION

Large Language Models (LLMs) have demonstrated strong capabilities across a wide range of applications Achiam et al. (2024); Touvron et al. (2023). However, their substantial storage and memory requirements remain a major obstacle to deployment on resource-constrained hardware, motivating extensive research on post-training weight compression. Among these approaches, singular value decomposition (SVD)-based low-rank approximation is widely used to represent each weight matrix within a low-dimensional subspace Yuan et al. (2023); Wang et al. (2025c).

Building on SVD to reduce the size of LLMs, several methods exploit shared basis across weight matrices, across layers, and within individual layers. Basis Sharing Wang et al. (2025a) and FiPS Üyük et al. (2026), for example, reuse low-rank representations across selected layers. Other approaches relax the assumption of a single subspace per matrix by modeling groups of weight vectors with distinct local low-rank subspaces Chen et al. (2018); Garcia et al. (2024). Complementary work has shown that weight matrices differ in their tolerance to low-rank approximation, motivating adaptive or non-uniform rank allocation Wang et al. (2025b); Xv et al. (2025); Abbasi et al. (2026). Another category of SVD-based methods identifies common subspace within a layer or weight matrix, such as Wang et al. (2025a); Üyük et al. (2026); Garcia et al. (2024); Lee et al. (2024); Sreeram et al. (2025), by partitioning the matrix into blocks and extracting common basis. Despite these advances, existing methods usually impose a fixed representation or sharing unit, such as a full matrix, a row, a column, or a spatial block. While reordering rows or columns can reduce dependence on the original matrix layout, the representation remains constrained by these predefined units.

We use a common formulation to compare several forms of low-rank representation and basis. An approximation of a weight matrix $\mathbf{W}$ can be written as $\mathbf{W} \approx \mathbf{BC}$, where $\mathbf{B} = \mathbf{U}_r$ denotes the basis matrix and $\mathbf{C} = \mathbf{\Sigma}_r \mathbf{V}_r^\top$ contains the corresponding coefficients. The basis $\mathbf{U}_r$, $\mathbf{\Sigma}_r$, and $\mathbf{V}_r$ are obtained from a rank-r SVD approximation. Methods such as ASVD Yuan et al. (2023) and SVD-LLM Wang et al. (2025c) apply this representation to individual weight matrices. Cross-layer methods extend this concept by sharing a basis across multiple layers, expressed as $\mathbf{W}^{(\ell)} \approx \mathbf{BC}^{(\ell)}$, where $\mathbf{B}$ is shared and $\mathbf{C}^{(\ell)}$ represents layer-specific coefficients.

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



(a) Block-structured sharing (b) shared weight-block subspaces (c) ARAS

Figure 1: Basis sharing schemes and opportunities within a weight matrix. (a) BLAST Lee et al. (2024) uses coarse local blocks, sharing $\mathbf{U}$ within a row block and $\mathbf{V}$ within a column block, where all blocks are spatially co-located. (b) Opportunities for sharing local subspaces increase across the matrix as weight blocks become more fine-grained. (c) ARAS relies on non-spatial, fine-grained shared local subspaces to compress the matrix.

At the matrix level, a weight matrix can instead be partitioned into fine-grained blocks to enable factor reuse across them. Figure 1a illustrates an example of this matrix-level reuse. Specifically, the sharing in Figure 1a follows a predefined block structure, as seen in prior structured approaches like BLAST Lee et al. (2024). As shown, all blocks in a given row share $\mathbf{U}$, while blocks in a given column share $\mathbf{V}$, with each block retaining its own dedicated $\mathbf{\Sigma}$. Although effective, this sharing scheme is constrained by the spatial layout of the matrix, preventing it from exploiting potential sharing opportunities across non-adjacent blocks.

Our observations show that LLM matrices exhibit opportunities for basis sharing across non-spatially adjacent blocks at a finer granularity than row or column subspaces. Figure 1b highlights these sharing opportunities across the matrix for fine-grained weight blocks on OPT-125M weight matrices. The x-axis indicates the granularity of the weight blocks, while the y-axis represents the extent to which spatial constraints on shared subspaces are relaxed, with the top of the y-axis corresponding to no spatial constraints. All compressions are performed at the same target rate of approximately 6 bits per weight (bpw). Under this equal compression budget, the 8×1, non-spatial configuration achieves the lowest normalized MSE of 0.0056, whereas the full-column 768×1 representation performs substantially worse at 0.2810. Crucially, fine granularity proves effective only when basis sharing is permitted beyond local spatial neighborhoods.

We propose Adaptive Rank-Allocated Shared Subspaces (ARAS), a matrix compression method that enables basis sharing across fine-grained weight blocks throughout an LLM weight matrix, as motivated by Figure 1b. An example of ARAS-compressed matrix is shown in Figure 1c where three basis factors $\mathbf{B}$ are shared among 9 blocks, with each block maintaining its own coefficient c. The shared subspaces are selected from across the entire matrix rather than restricted to local neighborhoods. ARAS employs subspace clustering to identify fine-grained weight blocks with similar low-rank structures and assigns them to shared subspaces independently of their spatial positions. Additionally, ARAS uses an adaptive rank-allocation technique across layers to balance model accuracy and memory footprint. ARAS is extensively evaluated across various architectures, backed by empirical experiments to determine optimal weight block configurations on OPT and LLaMA models. ARAS reduces the overall model size by 67% while maintaining competitive perplexity (PPL) as the dense baseline.

2 Methodology

ARAS is a post-training compression method with two main parts. First, each target weight matrix is partitioned into fine-grained weight blocks, and fine-grained weight blocks with similar low-dimensional structure are assigned to shared subspaces independently of their spatial locations. Second, ARAS adapts the representation rank across weight matrices based on their sensitivity to rank

reduction, allocating additional capacity to matrices for which compression is estimated to have a larger effect on model loss.

2.1 Learning Shared Subspaces For Fine-grained Weight Blocks

For each target weight matrix $\mathbf{W}_m \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, ARAS first partitions the matrix into non-overlapping $c \times t$ fine-grained weight blocks. This allows ARAS to capture local weight patterns and share a basis among fine-grained weight blocks with similar structure, rather than representing the entire matrix with a single basis. Each fine-grained weight block is vectorized as $\mathbf{x}_i \in \mathbb{R}^d$, where $d = ct$. Unless otherwise specified, we use 8×1 fine-grained weight blocks. The effect of fine-grained weight block size and the choice of this configuration are examined in the ablation study 3.4.2.

To obtain an initial grouping of fine-grained weight blocks with similar patterns, ARAS applies mini-batch k-means Sculley (2010) with k-means++ initialization Arthur & Vassilvitskii (2007). During this initialization step only, each fine-grained weight block is normalized as $\widetilde{\mathbf{x}}_i = \mathbf{x}_i / \|\mathbf{x}_i\|_2$, so that the initial grouping depends mainly on the direction of the fine-grained weight block rather than its magnitude. After initialization, ARAS returns to the original unnormalized fine-grained weight blocks. These fine-grained weight blocks are used to initialize the basis of each group with truncated SVD and are then used throughout the subsequent subspace learning and reconstruction steps.

To represent similar fine-grained weight blocks compactly, ARAS groups the fine-grained weight blocks of each weight matrix $\mathbf{W}_m$ into k shared low-dimensional subspaces. Fine-grained weight blocks assigned to the same subspace share a common basis. The j-th subspace, $j \in \{1, \ldots, k\}$, is represented by an orthonormal basis $\mathbf{B}_{m,j} \in \mathbb{R}^{d \times r_1}$, where k is the number of fine-grained weight block groups, and therefore the number of distinct shared bases, while r_1 is the larger candidate rank considered during rank allocation. Let $z_i \in \{1, \ldots, k\}$ denote the subspace assigned to fine-grained weight block $\mathbf{x}_i$. For each weight matrix $\mathbf{W}_m$, ARAS jointly learns the subspace bases and fine-grained weight block assignments by minimizing the reconstruction error:

$$\min_{\{\mathbf{B}_{m,j}\},\{z_i\}} \sum_i \left\| \mathbf{x}_i - \mathbf{B}_{m,z_i} \mathbf{B}_{m,z_i}^\top \mathbf{x}_i \right\|_2^2, \qquad \text{s.t.} \quad \mathbf{B}_{m,j}^\top \mathbf{B}_{m,j} = \mathbf{I}_{r_1}, \quad \forall j. \tag{1}$$

The term $\mathbf{B}_{m,z_i} \mathbf{B}_{m,z_i}^\top \mathbf{x}_i$ is the reconstruction of fine-grained weight block $\mathbf{x}_i$ using the basis of its assigned subspace.

We solve equation 1 using K-subspaces, which alternates between updating the shared bases and reassigning fine-grained weight blocks. Given the current fine-grained weight block assignments, each subspace basis is updated using the leading r_1 left singular vectors of the fine-grained weight blocks assigned to that subspace. The fine-grained weight blocks are then reassigned to the subspace that gives the smallest projection error. Since this assignment is based on how well a basis represents a fine-grained weight block, rather than on the fine-grained weight block's position in the matrix, fine-grained weight blocks from distant regions can still share the same basis if they exhibit similar weight patterns.

After convergence, each fine-grained weight block $\mathbf{x}_i$ is represented at rank r_1 by its subspace identifier z_i and a coefficient vector $\mathbf{a}_i = \mathbf{B}_{m,z_i}^\top \mathbf{x}_i$, with reconstruction $\widehat{\mathbf{x}}_i = \mathbf{B}_{m,z_i} \mathbf{a}_i$. Thus, fine-grained weight blocks assigned to the same subspace share one basis, while each fine-grained weight block keeps its own coefficients to preserve its individual values. The subsequent rank-allocation stage then decides, for each weight matrix, whether these shared bases retain all r_1 directions or are reduced to the first r_0. Further theoretical and empirical comparisons with fixed spatial sharing are provided in the ablation study 3.4.1.

2.2 Sensitivity-Aware Rank Allocation

Different weight matrices may respond differently to rank reduction, so a rank that works well for one matrix may remove too much information from another. ARAS therefore selects the rank separately for each weight matrix. After learning the subspaces at rank r_1, ARAS chooses a final rank $r_m \in \{r_0, r_1\}$ for each weight matrix $\mathbf{W}_m$, where $r_0 < r_1$. All k subspaces associated with

the same weight matrix use this selected rank. The learned subspaces determine which fine-grained weight blocks share a basis: k controls how many distinct bases are available for sharing, while r_m controls the dimensionality of each basis.

To measure how reducing the rank from r_1 to r_0 changes a weight matrix, ARAS first reconstructs each matrix using the learned rank-r_1 subspaces. It then forms the rank-r_0 reconstruction by keeping only the first r_0 basis directions from those same subspaces. We define the resulting change as

$$\mathbf{\Delta}_m = \widehat{\mathbf{W}}_m^{(r_0)} - \widehat{\mathbf{W}}_m^{(r_1)}. \tag{2}$$

where $\widehat{\mathbf{W}}_m^{(r)}$ denotes the reconstruction of $\mathbf{W}_m$ at rank r. Because both reconstructions use the same learned subspaces, $\mathbf{\Delta}_m$ captures only the change caused by reducing the rank, rather than differences from learning a new set of subspaces.

The next question is whether this change is important to the model. To estimate its effect on the loss, ARAS uses a small calibration set to compute the dense-model gradient $\mathbf{G}_m = \partial \ell / \partial \mathbf{W}_m$, where ℓ is the language-modeling loss. Motivated by a first-order Taylor approximation, we define the sensitivity of reducing the rank of $\mathbf{W}_m$ as

$$\mathcal{I}_m = \sum_{u,v} \left| G_{m,uv} \Delta_{m,uv} \right|. \tag{3}$$

Here, $G_{m,uv}$ indicates how strongly the loss responds to a change in weight (u, v), while $\Delta_{m,uv}$ is the change introduced by reducing the rank from r_1 to r_0. A larger $\mathcal{I}_m$ therefore indicates that reducing the rank of $\mathbf{W}_m$ is expected to have a larger effect on the model loss. Keeping a matrix at the higher rank r_1 can better preserve important information, but it also requires more storage. Let $s_m^{(r)}$ denote the storage needed to represent weight matrix $\mathbf{W}_m$ at rank r. To decide where the available storage should be spent, ARAS compares the estimated sensitivity of each matrix with the additional storage required to keep it at rank r_1. We define

$$\eta_m = \frac{\mathcal{I}_m}{s_m^{(r_1)} - s_m^{(r_0)}}, \tag{4}$$

which measures the estimated sensitivity preserved per additional byte of storage. All matrices are initially assigned the lower rank r_0. ARAS then considers matrices in descending order of η_m and upgrades a matrix to r_1 whenever the additional storage fits within the global compression budget. The truncated rank-r_0 reconstruction in equation 2 is used only to estimate the sensitivity of reducing the rank. After the ranks are allocated, each weight matrix is compressed at its assigned rank r_m; matrices assigned to r_0 are therefore fitted directly at rank r_0, rather than obtained by truncating the rank-r_1 representation. We evaluate this rank-allocation strategy against alternative allocation schemes in Section 3.4.3.

2.3 The Effect of ARAS on Compression and Computation

This section first analyzes how ARAS impacts model memory and floating-point operations (FLOPs), and then discusses how it facilitates the integration of compression methods.

Compression in ARAS For a weight matrix $\mathbf{W}_m$, ARAS partitions it into $c \times t$ fine-grained weight blocks and assigns a rank r_m to the matrix. Each fine-grained weight block stores r_m FP16 coefficients and a $\lceil \log_2 k \rceil$-bit subspace identifier indicating its assigned subspace. Each matrix stores k subspaces, each of size ctr_m. Let $N_{\text{dense},m} = d_{\text{out}} d_{\text{in}}$ denote the number of dense FP16 parameters. The storage retention ratio ρ_m for weight matrix $\mathbf{W}_m$ is given by the equation below, with the details explained in Appendix A.4.

$$\rho_m = \underbrace{\frac{r_m}{ct}}_{\text{(1) Coefficients}} + \underbrace{\frac{kct\, r_m}{N_{\text{dense},m}}}_{\text{(2) Subspace}} + \underbrace{\frac{\lceil \log_2 k \rceil}{16ct}}_{\text{(3) Subspace IDs}} \tag{5}$$

Computation in ARAS The compression cost consists of three terms: encoding, subspace learning, and sensitivity-aware rank allocation. Encoding assigns each d-entry weight block to one of k

subspaces and stores its subspace index and rank-r coefficients, with cost scaling with the matrix parameter count N_{dense}. Subspace learning trains k bases from a fixed sample of S blocks for I iterations and is therefore independent of N_{dense}. Sensitivity-aware rank allocation uses T_{cal} calibration tokens to estimate loss sensitivity and assigns each matrix either rank r_0 or r_1 under the storage budget.

The analysis for computing ARAS in terms of FLOPs is given by the equation below.

$$FLOP_{\text{ARAS}} = \underbrace{2k(d+1)\big(r_1 \tfrac{N_{\text{dense}}}{d} + r_0 \tfrac{N_{\text{dense}}}{d}\big)}_{\text{(1) Encoding}} + \underbrace{2k(d+1)\, IS\big(r_1 + r_0\big)}_{\text{(2) Subspaces learning}} + \underbrace{8\, N_{\text{dense}}\, T_{\text{cal}}}_{\text{(3) Sensitivity aware}} \tag{6}$$

Enabling quantization and sparsification In addition to reducing memory and FLOPs, ARAS can support complementary compression techniques such as quantization and sparsification. ARAS represents each fine-grained weight block using a shared basis and a low-dimensional coefficient vector. The shared bases capture structure that can be reused across blocks, while the coefficients provide a compact representation that can be compressed further without changing the learned bases.

For a fine-grained weight block i assigned to subspace z_i, ARAS computes $\mathbf{a}_i = \mathbf{B}_{z_i}^\top \mathbf{x}_i$ and reconstructs the block as $\hat{\mathbf{x}}_i = \mathbf{B}_{z_i}\mathbf{a}_i$. We quantize these coefficients using vector K-means independently for each weight matrix. Each coefficient vector is replaced by the index of its nearest centroid, $q_i = \arg\min_j \|\mathbf{a}_i - \mathbf{c}_j\|_2^2$, giving $\hat{\mathbf{x}}_i = \mathbf{B}_{z_i}\mathbf{c}_{q_i}$. This replaces a full-precision coefficient vector with a compact code index and codebook entry, further reducing storage and coefficient memory traffic.

ARAS can also be combined with weight sparsification by pruning low-magnitude entries from the original weight matrices before applying the ARAS compression pipeline. This reduces the number of nonzero weights and, with sparse execution support, can also reduce arithmetic operations and memory accesses during inference. Quantization and sparsification are not the primary focus of ARAS, but these experiments show that ARAS is compatible with additional compression techniques that target memory and computation. We leave a more complete study of these extensions to future work. Additional implementation details and perplexity results for coefficient quantization are reported in Appendix A.2.

3 Experimental Results

3.1 Experimental Setup

Models and Task. We evaluate ARAS on two widely used LLM families: OPT-125M to OPT-30B Zhang et al. (2022), and Llama-3.2-1B, Llama-3.2-3B, and Llama-3.1-8B Grattafiori et al. (2024). We report perplexity on WikiText2 Merity et al. (2016) in the main paper, with additional results on PTB Marcus et al. (1993) and C4 Raffel et al. (2023) provided in the Appendix A.3.

Baseline Choice We compare ARAS against some state-of-the-art compression methods as baselines: ASVD Yuan et al. (2023), SVD-LLM Wang et al. (2025c), Basis Sharing Wang et al. (2025a) and BLAST Lee et al. (2024). We apply ARAS and other baselines to compress the matrices, including the attention module and the feed-forward network. Specifically, for both the OPT and Llama families, we compress `q_proj`, `k_proj`, `v_proj`, and `o_proj`. In addition, we compress `fc1` and `fc2` for OPT models, and `gate_proj`, `up_proj`, and `down_proj` for Llama models.

Calibration Data ARAS uses the first four sequences from the WikiText-2 training split, each containing 256 tokens, for rank allocation. The calibration data are used only to estimate the sensitivity of each weight matrix and do not update the parameters.

Hardware Used We run the experiment on a server compute node equipped with an AMD EPYC 9655 (Zen 5) CPU and an NVIDIA H100 GPU with 80 GB of memory. Unless otherwise specified, the models, compression procedure, datasets, and inference are all in FP16 precision.

3.2 Results

Perplexity Evaluation. Tables 1 and 2 report WikiText-2 perplexity for ARAS and the baselines on the OPT and Llama families, respectively. We use $K = 512$ shared subspaces for OPT and

Table 1: Perplexity (↓) on WikiText-2 for the OPT family models. The parameter ρ represents the storage reduction ratio.

Method	ρ	OPT-125M	OPT-350M	OPT-1.3B	OPT-2.7B	OPT-6.7B	OPT-13B	OPT-30B
Dense	1.0	27.65	22.00	14.62	12.47	10.86	10.13	9.56
ASVD	0.60	339.6	319.7	6496	3110	297.4	37.25	12.72
SVD-LLM		72.01	116.6	28.09	23.41	16.26	12.85	11.08
Basis Sharing		76.49	65.75	28.95	21.58	15.10	12.74	10.67
ARAS		**32.26**	**24.09**	**15.93**	**13.44**	**11.95**	**11.03**	**10.57**
ASVD	0.33	3298	2146	13544	16228	19276	12934	4478
SVD-LLM		501.5	1066	462.4	506.4	371.5	30.40	22.19
Basis Sharing		540.5	1231	575.0	451.7	274.0	29.06	18.01
ARAS		**32.71**	**24.21**	**16.06**	**13.48**	**11.94**	**11.08**	**10.61**

$K = 128$ for Llama, based on the study in Appendix A.1. We evaluate two target storage retention ratios, $\rho = 0.6$ and $\rho = 0.33$, where ρ is defined in equation 5.

Across all evaluated models and storage ratios, ARAS achieves lower perplexity than the strongest competing baseline at the same compression level. At $\rho = 0.6$, the improvement ranges from 0.9% on OPT-30B to 94.4% on Llama-3.2-1B. Under the more aggressive $\rho = 0.33$ setting, the gap becomes substantially larger, reaching up to 98.9% on Llama-3.1-8B. This trend indicates that ARAS preserves model quality more effectively as the compression constraint becomes tighter.

Table 2: Perplexity (↓) on WikiText-2 for the Llama family models (version-size).

Method	ρ	(3.2-1B)	(3.2-3B)	(3.1-8B)	Method	ρ	(3.2-1B)	(3.2-3B)	(3.1-8B)
Dense	1.0	9.75	7.81	6.24	Dense	1.0	9.75	7.81	6.24
ASVD	0.60	24921	4102	19279	ASVD	0.33	22887	11481	20664
SVD-LLM		696.5	295.7	177.9	SVD-LLM		2290	3295	28395
Basis Sharing		178.4	63.32	83.83	Basis Sharing		802.6	280.6	625.4
ARAS		**9.97**	**8.59**	**6.90**	ARAS		**10.27**	**8.79**	**7.10**

Across model families. We next examine whether ARAS behaves consistently across model families and whether the competing methods show similar robustness. At $\rho = 0.6$, the perplexity increase of ARAS relative to the dense model averages 10.4% across the seven OPT models and 7.6% across the three Llama models. At the more aggressive $\rho = 0.33$ setting, the corresponding increases are 10.9% and 10.6%, respectively. This shows that the degradation of ARAS remains relatively stable across both model families and compression levels.

In contrast, the competing baselines exhibit much larger variation across model sizes and families. Their perplexity can increase moderately for some larger OPT models but becomes extremely large for several smaller OPT and Llama models, particularly under $\rho = 0.33$. This difference suggests that ARAS is less sensitive to model architecture and scale than the evaluated baselines.

Across scales within a family. Within the OPT family, the perplexity gap between ARAS and the dense model generally decreases as model size increases, from 4.61 for OPT-125M to 1.01 for OPT-30B at $\rho = 0.6$. This suggests that larger OPT models tolerate the ARAS compression more effectively in absolute perplexity terms. For the Llama family, however, we do not observe the same monotonic trend: the gaps are 0.22, 0.78, and 0.66 for Llama-3.2-1B, Llama-3.2-3B, and Llama-3.1-8B, respectively. These results indicate that the effect of model scale is architecture-dependent rather than uniform across families.

3.3 Comparing with Intra-layer compression

We select BLAST as a closely related baseline for comparison because it is also an intra-layer compression method. BLAST partitions each weight matrix into a regular block structure and shares subspaces based on the spatial adjacency of blocks, row-block-wise and column-block-wise, whereas

ARAS operates on fine-grained weight blocks and allows spatially non-contiguous weight blocks to share the same subspace. We use the default BLAST setting, $b = 16$. Since ARAS is a post-training compression method that does not include retraining, we compare it against the compression-only stage of BLAST. Table 3 shows that ARAS achieves lower perplexity, fewer FLOPs, and lower memory usage than BLAST, suggesting that ARAS with fine-grained weight blocks, which provide non-spatial adjacent subspace sharing, offers more favorable performance than BLAST.

Table 3: Computational cost (FLOPs, ↓), memory usage (MB, ↓), and WikiText-2 perplexity (↓) for the compression stage. Dense baseline has no compression phase.

		OPT-125M			LLaMA-3.2-1B		
Method	ρ	FLOPs ↓	Memory (MB) ↓	PPL ↓	FLOPs ↓	Memory (MB) ↓	PPL ↓
Dense	1.00	–	–	27.65	–	–	9.75
BLAST	0.60	3.1×10^{14}	276	102.66	1.1×10^{16}	2 105	53
ARAS	0.60	$\mathbf{7.0\times10^{12}}$	**86**	**32.26**	$\mathbf{2.9\times10^{13}}$	**464**	**9.97**
BLAST	0.33	1.1×10^{14}	99	714.73	3.7×10^{15}	742	78
ARAS	0.33	$\mathbf{7.0\times10^{12}}$	**86**	**32.71**	$\mathbf{2.9\times10^{13}}$	**464**	**10.27**

3.4 Ablation Study

3.4.1 Why Learned Non-Spatial Basis Sharing?

We hypothesize that useful low-dimensional structure is not necessarily confined to neighboring regions of a weight matrix. Spatially distant fine-grained weight blocks may exhibit similar structure and therefore benefit from sharing the same basis.We test this hypothesis by learning block-to-basis assignments from reconstruction similarity rather than fixing them according to matrix location.

To compare these two choices, we use a fixed spatial-sharing baseline with the same block partition, number of bases, and subspace rank as ARAS. Let $g_m(i)$ denote the predefined spatial group containing block i in weight matrix $\mathbf{W}_m$. In the fixed baseline, all blocks in the same spatial group share the corresponding basis. ARAS instead learns the assignment of each block to a basis. Therefore, the fixed spatial assignment is one feasible assignment available to ARAS, but ARAS is not restricted to it and can assign spatially separated blocks to the same basis when they exhibit similar low-dimensional structure.

For OPT-125M, we evaluate all 72 decoder weight matrices using 8×1 blocks, $k = 512$, and $r = 4$. For Llama-3.2-1B, we evaluate all 112 decoder weight matrices with the same block size and rank, using $k = 128$. Across all 184 evaluated matrices, ARAS achieves lower reconstruction error than fixed spatial sharing. For OPT-125M, the mean relative reconstruction error decreases from 0.634 to 0.137, corresponding to a 78.4% reduction, while for Llama-3.2-1B it decreases from 0.689 to 0.206, a 70.1% reduction.

To verify that ARAS actually uses this additional flexibility, we measure the spatial support of each learned subspace as $\mathcal{S}_{m,j} = |\{g_m(i) : z_i = j\}|$, where z_i is the learned subspace assignment of block i. Thus, $\mathcal{S}_{m,j}$ counts how many distinct predefined spatial groups contribute blocks to learned subspace j. If $\mathcal{S}_{m,j} = 1$, the learned basis is used only within one spatial region; if $\mathcal{S}_{m,j} > 1$, the same basis is shared by blocks from different spatial regions. Across both models, every non-empty learned subspace satisfies $\mathcal{S}_{m,j} > 1$. This shows that ARAS does not simply reproduce the predefined spatial partition. Instead, it uses the additional assignment flexibility to share bases across spatially separated fine-grained weight blocks with similar low-dimensional structure.

3.4.2 Effect of Fine-grained weight block Granularity

We examine how fine-grained weight block granularity affects the learned subspace representation on representative layers of OPT-125M and Llama-3.2-1B. Here, c and t denote the fine-grained weight block height and width, respectively, so that each vectorized fine-grained weight block has dimension $d = ct$, while r denotes the subspace rank. Since changing the fine-grained weight block size also changes d, we scale r accordingly and keep the relative subspace dimension fixed at $r/(ct) = 0.5$ throughout this study.

Table 4: Effect of fine-grained weight block shape and width with a fixed relative subspace dimension $r/(ct) = 0.5$. Values report mean held-out captured energy (%).

(a) Block shape and orientation

Block	Rank	OPT	Llama
$4\times1/1\times4$	2	99.82/99.84	99.37/99.37
$8\times1/1\times8$	4	97.76/97.75	95.58/95.58
$16\times1/1\times16$	8	91.84/91.83	88.18/88.16

(b) Block width

Block	Rank	OPT	Llama
8×1	4	**97.76**	**95.58**
8×4	16	83.65	79.84
8×8	32	75.63	72.26

Fine-grained weight block height and orientation. We first vary the height of the fine-grained weight blocks while fixing the width to $t = 1$. As shown in Table 4 (a), smaller 4×1 blocks preserve the most energy, reaching 99.82% on OPT-125M and 99.37% on Llama-3.2-1B. Increasing the block size to 8×1 causes only a small reduction, to 97.76% and 95.58%, while 16×1 blocks reduce the captured energy more substantially, to 91.84% and 88.18%, respectively. We also evaluate the transposed block orientations. The results are nearly identical for 4×1 and 1×4, 8×1 and 1×8, and 16×1 and 1×16. This suggests that the observed behavior is driven primarily by block granularity rather than by whether the longer block dimension is aligned with rows or columns.

Fine-grained weight block width. We next fix the fine-grained weight block height to $c = 8$ and vary the width t. To keep the relative subspace dimension fixed at $r/(ct) = 0.5$, we increase the rank as the block becomes wider. As shown in Table 4(b), increasing the width consistently reduces the captured energy for both models. For OPT-125M, the energy decreases from 97.76% with 8×1 blocks to 83.65% and 75.63% with 8×4 and 8×8 blocks, respectively. Llama-3.2-1B shows the same trend, decreasing from 95.58% to 79.84% and 72.26%. This drop occurs even though the rank increases proportionally with the block dimension. Wider fine-grained weight blocks are therefore harder to represent with a single learned subspace, suggesting that grouping more neighboring weights within each block reduces the effectiveness of the learned low-dimensional representation. We therefore use $t = 1$ in the main experiments.

Together with the block-height study, these results support 8×1 as a practical fine-grained weight block configuration for the main experiments. Although 4×1 blocks capture slightly more energy, 8×1 preserves most of the captured energy while avoiding an unnecessarily fine representation. Larger 16×1 blocks show a more substantial loss in captured energy, and increasing the block width causes an even larger reduction.

3.4.3 Does First-Order Sensitivity Improve Rank Allocation?

The rank-allocation stage in ARAS uses first-order loss sensitivity to decide which weight matrices receive the higher rank. We test whether this loss information improves allocation compared with reconstruction magnitude alone or a random ordering, while keeping the candidate ranks, compression settings, and target storage budget fixed. Matrices are assigned either $r_0 = 4$ or $r_1 = 6$, and all policies use the same storage-constrained greedy allocation rule. They differ only in how candidate upgrades are ranked.

Sensitivity is the allocation rule used by ARAS. It computes the rank-reduction perturbation $\mathbf{\Delta}_m$ from equation 2, estimates its first-order effect on the loss using equation 3, and ranks upgrades by the sensitivity-per-storage score η_m in equation 4.

Reconstruction uses the same perturbation and storage cost but removes gradient information. Candidate upgrades are ranked by $\eta_m^{\text{rec}} = \frac{\|\mathbf{\Delta}_m\|_F}{s_m^{(r_1)} - s_m^{(r_0)}}$. It therefore prioritizes matrices with larger reconstruction changes under rank reduction, regardless of their effect on the model loss.

Random removes both sensitivity and reconstruction information. Matrices are processed in a random order under the same storage-constrained allocation rule. We report the mean and standard deviation over five random seeds.

For all policies, $\mathbf{\Delta}_m$ is used only to score candidate allocations. Once the ranks are assigned, the evaluated model is assembled from the corresponding independently compressed rank-r_0 and rank-r_1 artifacts. Thus, only the allocation rule changes across policies.

Table 5: Effect of rank-allocation policy on WikiText-2 and PTB. Lower perplexity is better.

Model	Allocation	WikiText-2 ↓	PTB ↓	Storage (MiB)
OPT-125M	Sensitivity	**31.74**	**37.42**	56.06
OPT-125M	Reconstruction	37.57	44.22	56.10
OPT-125M	Random	36.78	43.39	56.02
Llama-3.2-1B	Sensitivity	**11.07**	**18.04**	698.12
Llama-3.2-1B	Reconstruction	13.58	21.94	695.36
Llama-3.2-1B	Random	13.28	21.74	698.03

Table 5 shows that sensitivity-based allocation achieves lower perplexity than both reconstruction-based and random allocation on OPT-125M and Llama-3.2-1B, across WikiText-2 and PTB, at comparable storage. The reconstruction baseline isolates the contribution of the first-order loss term: both methods use the same rank-reduction perturbation and storage normalization, but only sensitivity accounts for the loss response. The random baseline further shows that the gain does not arise from the storage-constrained allocation procedure alone. These results indicate that first-order loss sensitivity provides useful information for deciding where additional rank should be allocated, without implying that the resulting assignment is globally optimal.

3.4.4 Cross-Layer Basis Similarity

ARAS currently learns shared bases independently within each weight matrix. As a preliminary study, we compare matched cross-layer subspaces with a random matching baseline on OPT-125M and Llama-3.2-1B. OPT-125M shows little difference from random matching, while Llama-3.2-1B exhibits a small positive gap of approximately 0.003 to 0.010. These results suggest that cross-layer similarity may exist, but is model- and matrix-dependent. We therefore leave cross-layer basis sharing as a direction for future work.

4 Related Work

Low-rank factorization is widely used for LLM compression. Methods such as ASVD and SVD-LLM improve matrix-level approximation using activation information or truncation-aware transformations (Yuan et al., 2023; Wang et al., 2025d). Other methods reuse low-rank structure across layers: Basis Sharing shares SVD bases across Transformer layers, while FiPS combines shared low-rank structure with sparse layer-specific factors (Wang et al., 2025a; Üyük et al., 2026).

Within individual weight matrices, prior work uses different representation granularities. GroupReduce learns low-rank approximations for groups of weight vectors (Chen et al., 2018), Compress to Impress uses projective clustering over consecutive matrix regions (Sreeram et al., 2025), and BLAST shares basis according to a predefined block-row and block-column structure (Lee et al., 2024). These approaches tie the representation or sharing pattern to predefined matrix regions. In contrast, ARAS learns shared subspaces among fine-grained weight blocks independently of spatial proximity, allowing distant blocks with similar low-dimensional structure to share the same subspace.

Related work also considers non-uniform rank allocation. SVD-LLM V2, ARA, Swift-SVD, and IO-SVD allocate different ranks or compression levels across weight matrices based on importance or sensitivity (Wang et al., 2025c; Qi et al., 2026; Abbasi et al., 2026). ARAS combines this idea with its learned shared-subspace representation under a global storage budget.

5 Conclusion

We introduced ARAS, a post-training compression framework that learns shared low-dimensional subspaces among fine-grained weight blocks without restricting sharing to spatially adjacent regions. ARAS further adapts rank across weight matrices under a fixed storage budget. Across OPT and Llama models, it consistently achieves lower perplexity than the evaluated low-rank baselines at matched compression levels. Future work could extend shared-subspace learning across layers and explore additional rank and quantization strategies.

AI USE STATEMENT

Generative AI tools were used to assist with rewriting and language editing to improve the clarity and readability of the manuscript. All AI-assisted text was reviewed and revised by the authors, who take full responsibility for the final content of the paper.

REFERENCES

Ali Abbasi, Chayne Thrash, Haoran Qin, Hamed Pirsiavash, and Soheil Kolouri. Io-svd: Input-output whitened svd for adaptive-rank llm compression, 2026. URL https://arxiv.org/abs/2605.15626.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, et al. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1027–1035, 2007.

Patrick Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/a2b8a85a29b2d64ad6f47275bf1360c6-Paper.pdf.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12284–12303. PMLR, 2024.

Daniel Garcia, Roberto Garcia, and Andri Vidarsson. Optimal brain projection: Neural network compression using mixtures of subspaces, 2024. URL https://web.stanford.edu/class/cs224n/final-reports/256910038.pdf. Stanford CS224N Custom Final Project.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Changwoo Lee, Soo Min Kwon, Qing Qu, and Hun-Seok Kim. BLAST: Block-level adaptive structured matrices for efficient deep neural network inference. In *Advances in Neural Information Processing Systems*, 2024.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL https://aclanthology.org/J93-2004/.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. URL https://arxiv.org/abs/1609.07843.

Ruoling Qi, Yirui Liu, Xuaner Wu, Xiangyu Wang, Ming Li, Chen Chen, Jian Chen, Yin Chen, and Qizhen Weng. Swift-SVD: Theoretical optimality meets practical efficiency in low-rank LLM compression. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=nAQ4h8FpdM.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023. URL https://arxiv.org/abs/1910.10683.

D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, pp. 1177–1178, 2010.

Shiva Sreeram, Alaa Maalouf, Pratyusha Sharma, and Daniela Rus. Compress to impress: Efficient llm adaptation using a single gradient step on 100 samples. In *Advances in Neural Information Processing Systems*, 2025.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, et al. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Jingcun Wang, Yu-Guang Chen, Ing-Chao Lin, Bing Li, and Grace Li Zhang. Basis sharing: Cross-layer parameter sharing for large language model compression. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025a.

Xin Wang, Samiul Alam, Zhongwei Wan, Hui Shen, and Mi Zhang. Svd-llm v2: Optimizing singular value truncation for large language model compression, 2025b. URL https://arxiv.org/abs/2503.12340.

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

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. Svd-llm: Truncation-aware singular value decomposition for large language model compression. In *International Conference on Learning Representations*, 2025d.

Lin Xv, Jingsheng Gao, Xian Gao, Ting Liu, and Yuzhuo Fu. Ara: Adaptive rank allocation for efficient large language model svd compression, 2025. URL https://arxiv.org/abs/2510.19389.

Zhihang Yuan, Yuzhang Shang, Yue Song, Qiang Wu, Yan Yan, and Guangyu Sun. Asvd: Activation-aware singular value decomposition for compressing large language models, 2023.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022. URL https://arxiv.org/abs/2205.01068.

Cem Üyük, Mike Lasby, Mohamed Yassin, Utku Evci, and Yani Ioannou. Learning fine-grained parameter sharing via sparse tensor decomposition, 2026. URL https://arxiv.org/abs/2411.09816.

A Appendix

A.1 Choosing the Number of Shared Subspaces

The number of shared subspaces, K, determines a trade-off among model quality, representation size, and compression cost. Increasing K provides more specialized bases and can therefore reduce approximation error, but it also requires storing more bases and increases the cost of fitting the subspaces. Rather than tuning K separately for every model, we select one value for each model family using a representative model and keep it fixed for all other models in that family.

For the OPT family, we perform this ablation on OPT-125M. Increasing K from 64 to 512 substantially reduces perplexity, from 53.22 to 32.01. Beyond this point, the gain becomes small: increasing K from 512 to 1024 reduces perplexity only from 32.01 to 31.43, corresponding to a further improvement of 1.79

Table 6: Comparison of ARAS with AQLM under the $1 \times 65{,}536$ setting, without calibration or fine-tuning. Lower perplexity is better.

Model	Method	WikiText2	PTB	C4
OPT-125M	AQLM	36.56	54.68	34.16
	ARAS	**36.32**	**54.08**	**34.09**
Llama-3.2-1B	AQLM	147.41	289.99	189.71
	ARAS	**19.37**	**36.42**	**28.02**

For the Llama family, the cost–quality trade-off reaches diminishing returns at a smaller value of K. On Llama-3.2-1B, increasing K from 32 to 128 reduces perplexity from 19.58 to 12.74. Increasing K further to 256 and 512 reduces perplexity to 11.76 and 11.12, respectively, but requires progressively more storage and computation. The representation size increases from 666.27,MB at $K = 128$ to 681.64,MB at $K = 256$ and 698.12,MB at $K = 512$, while compression time increases from 282.4,s to 439.0,s and 1520.8,s, respectively. We therefore use $K = 128$ for all Llama models, which retains most of the improvement obtained by increasing the number of subspaces while avoiding the higher storage and compression costs of larger K.

These values are selected once at the model-family level rather than optimized independently for each evaluated model. We subsequently fix $K = 512$ for OPT and $K = 128$ for Llama throughout the remaining experiments, avoiding model-specific tuning while accounting for the different empirical cost–quality trade-offs of the two model families.

A.2 Coefficient Quantization and Sparsification Results

We evaluate coefficient quantization on OPT-125M and Llama-3.2-1B and compare it with AQLM Egiazarian et al. (2024) under the same $1 \times 65{,}536$ setting. To isolate the effect of coefficient quantization, neither method uses calibration or fine-tuning in this comparison. Table 6 reports perplexity on WikiText2, PTB, and C4.

On OPT-125M, ARAS and AQLM obtain similar perplexity, with ARAS giving slightly lower values on all three datasets. On Llama-3.2-1B, ARAS maintains substantially lower perplexity across WikiText2, PTB, and C4. These results indicate that quantizing the low-dimensional coefficient representation can further reduce storage without requiring calibration or fine-tuning.

We also evaluate weight sparsification on OPT-125M. At an effective sparsity of approximately 20%, WikiText2 perplexity increases from 27.65 for the dense model to 40.73 after sparsification. Although sparsification causes a larger quality degradation than coefficient quantization in this setting, it shows that ARAS can be combined with weight sparsification to reduce the number of stored nonzero values. A more complete study of the sparsity–accuracy trade-off, including comparisons with dedicated sparsification methods, is left for future work.

A.3 Additional perplexity evaluation

We evaluate the ARAS and other baseline models' perplexity performance on three language modeling datasets: WikiText-2 Merity et al. (2016), PTB Marcus et al. (1993), and C4 Raffel et al. (2023). All baselines run at their reference defaults, where ASVD with $\alpha = 0.5$ and $n_{\text{calib}} = 32$; SVD-LLM with $n_{\text{whiten}} = 256$ at `seqlen` $= 2048$, without its optional LoRA recovery step; Basis Sharing with `group_size` $= 2$ and `context_length` $= 2048$.

Tables 7 and 8 show the perplexity for models in the OPT family and Llama family, respectively.

A.4 storage retention ratio

We derive the size of the compressed model and storage retention ratio for a weight matrix $\mathbf{W}_m \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. As defined in Section 2.1, ARAS partitions $\mathbf{W}_m$ into non-overlapping $c \times t$ fine-grained weight blocks and vectorizes each fine-grained weight block as $\mathbf{x}_i \in \mathbb{R}^d$, where $d = ct$. Each weight matrix uses k shared subspace bases and is assigned a final rank $r_m \in \{r_0, r_1\}$ as described

Table 7: Perplexity (↓) on WikiText-2 (WT2), PTB and C4 for the OPT family models comparing ARAS and the baseline methods.

Method	ρ	OPT-125M			OPT-350M			OPT-1.3B			OPT-2.7B			OPT-6.7B			OPT-13B			OPT-30B		
		WT2↓	PTB↓	C4↓	WT2↓	PTB↓	C4↓	WT2↓	PTB↓	C4↓	WT2↓	PTB↓	C4↓	WT2↓	PTB↓	C4↓	WT2↓	PTB↓	C4↓	WT2↓	PTB↓	C4↓
Dense	1.0	27.65	38.98	26.56	22.00	25.20	21.98	14.62	16.36	15.71	12.47	17.15	14.70	10.86	15.76	12.71	10.13	14.52	12.06	9.56	14.03	11.44
ASVD	0.60	339.6	508.2	293.3	319.7	401.9	242.3	6496	7592	5351	3110	7125	3666	297.4	1619	454.8	37.25	113.6	48.83	12.72	25.92	17.67
SVD-LLM		72.01	356.9	165.3	116.6	252.3	164.5	28.09	120.0	108.0	23.41	87.79	79.52	16.26	38.28	36.26	12.85	22.09	21.04	11.08	17.92	15.37
Basis Sharing		76.49	436.2	165.9	65.75	140.1	84.60	28.95	153.2	104.7	21.58	74.92	61.50	15.10	34.17	29.54	12.74	21.62	18.95	10.67	17.30	14.62
ARAS		**32.26**	**46.93**	**30.15**	**24.09**	**27.57**	**23.77**	**15.93**	**18.18**	**17.42**	**13.44**	**18.95**	**18.95**	**11.95**	**18.76**	**14.43**	**11.03**	**16.04**	**13.02**	**10.57**	**15.70**	**12.31**
ASVD	0.33	3298	4878	2808	2146	1674	1471	13544	11738	8699	16228	15468	13990	19276	18551	12698	12934	12796	9651	4478	5695	3119
SVD-LLM		501.5	3221	1856	1066	2660	1540	462.4	5135	3908	506.4	4825	4201	371.5	3257	2446	30.40	205.2	148.6	22.19	78.72	67.28
Basis Sharing		540.5	3438	1739	1231	2539	1703	575.0	3564	2492	451.7	4262	2752	274.0	5516	1797	29.06	107.6	96.06	18.01	62.20	47.78
ARAS		**32.71**	**46.91**	**30.38**	**24.21**	**27.66**	**23.82**	**16.06**	**18.30**	**17.53**	**13.48**	**19.05**	**15.83**	**11.94**	**18.81**	**14.46**	**11.08**	**16.12**	**13.06**	**10.61**	**15.75**	**12.33**

Table 8: Perplexity (↓) on WikiText-2 (WT2), PTB and C4 for the Llama family models comparing ARAS and the baseline methods.

Method	ρ	LLaMA-3.2-1B			LLaMA-3.2-3B			LLaMA-3.1-8B		
		WT2↓	PTB↓	C4↓	WT2↓	PTB↓	C4↓	WT2↓	PTB↓	C4↓
Dense	1.0	9.75	17.58	14.01	7.81	11.93	11.27	6.24	11.14	9.54
ASVD	0.60	24921	30814	23770	4102	3250	2897	19279	16452	9308
SVD-LLM		696.5	3743	2205	295.7	1929	856.1	177.9	2336	1038
Basis Sharing		178.4	648.3	655.3	63.32	207.6	240.0	83.83	1299	470.9
ARAS		**9.97**	**18.00**	**14.42**	**8.59**	**13.02**	**12.77**	**6.90**	**11.89**	**10.84**
ASVD	0.33	22887	29472	25539	11481	6755	7309	20664	16082	13750
SVD-LLM		2290	6162	5140	3295	6154	6134	28395	69740	53124
Basis Sharing		802.6	3491	2859	280.6	1763	1037	625.4	5406	3229
ARAS		**10.27**	**18.66**	**14.94**	**8.79**	**13.21**	**13.06**	**7.10**	**12.13**	**11.19**

in Section 2.2. The fine-grained weight block coefficients and shared bases are stored in FP16, while each subspace identifier z_i is stored using $\lceil \log_2 k \rceil$ bits.

The dense weight matrix contains $N_{\text{dense},m} = d_{\text{out}} d_{\text{in}}$ FP16 parameters, and the number of fine-grained weight blocks is $N_{\text{blocks},m} = d_{\text{out}} d_{\text{in}}/(ct)$.

(1) Fine-grained weight block coefficients. Each weight block stores a coefficient vector of length r_m. The coefficient storage, expressed in FP16-equivalent units, is $N_{\text{coeff},m} = N_{\text{blocks},m} r_m = d_{\text{out}} d_{\text{in}} r_m/(ct)$.

(2) Shared subspace. Each weight matrix contains k shared subspaces $\mathbf{B}_{m,j} \in \mathbb{R}^{d \times r_m}$, $j \in \{1, \ldots, k\}$. Since $d = ct$, the corresponding basis storage is $N_{\text{subspaces},m} = kct\, r_m$.

(3) Subspace IDs. Each weight block stores its subspace ID $z_i \in \{1, \ldots, k\}$ using $\lceil \log_2 k \rceil$ bits. Expressed in FP16-equivalent units, the ID storage is $N_{\text{id},m} = \frac{d_{\text{out}} d_{\text{in}}}{ct} \frac{\lceil \log_2 k \rceil}{16}$.

Consistent with Section 2.2, let $s_m^{(r)}$ denote the storage required to represent weight matrix $\mathbf{W}_m$ at rank r. For the selected rank r_m, the total compressed storage is therefore $s_m^{(r_m)} = \frac{d_{\text{out}} d_{\text{in}}}{ct} r_m + kct\, r_m + \frac{d_{\text{out}} d_{\text{in}}}{ct} \frac{\lceil \log_2 k \rceil}{16}$.

The storage retention ratio for $\mathbf{W}_m$ is: $\rho_m = \frac{s_m^{(r_m)}}{d_{\text{out}} d_{\text{in}}} = \frac{r_m}{ct} + \frac{kct\, r_m}{d_{\text{out}} d_{\text{in}}} + \frac{\lceil \log_2 k \rceil}{16ct}$.