Low-rank Factorization for LLM Compression with Dynamic Capacity Allocation and Block-level Refinement

Atul Kumar Sinha [1] **François Fleuret** [1 2]

Abstract

We propose AA-SVD, a low-rank factorization-based compression framework for large language models that enables performance-preserving compression without full-model retraining. Our method finds a low-rank approximation for weight matrices of linear layers that minimizes the reconstruction error in activation space. A central principle underlying AA-SVD is that optimizing on original inputs alone ignores distribution shift from upstream compression, while conditioning on shifted inputs alone risks drifting away from the original outputs — AA-SVD accounts for both. Further, we refine each transformer block locally, minimizing block-level output distortion and allowing compressed layers to jointly compensate for accumulated errors. During block-level refinement, we also learn the rank allocation across layers, concentrating capacity where it most reduces block-output distortion. We evaluate AA-SVD across LLaMA and Qwen model families on language modeling and commonsense reasoning benchmarks. AA-SVD consistently outperforms existing SVD-based compression methods across a range of compression ratios, with the advantage becoming increasingly pronounced at aggressive compression ratios.

1. Introduction

Large language models (LLMs) now routinely contain tens to hundreds of billions of parameters (Brown et al., 2020; Touvron et al., 2023; Achiam et al., 2023), enabling remarkable generalization across a wide range of tasks (Kaplan et al., 2020). A key asymmetry emerges from this scale: training cost is *ephemeral*, whereas inference cost *accumulates* with every query from every user, making it the dominant expense in deployment and a growing source of carbon emissions (Patterson et al., 2021).

[1]University of Geneva, Geneva, Switzerland [2]FAIR, Meta. Correspondence to: Atul Kumar Sinha <atul.sinha@unige.ch>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

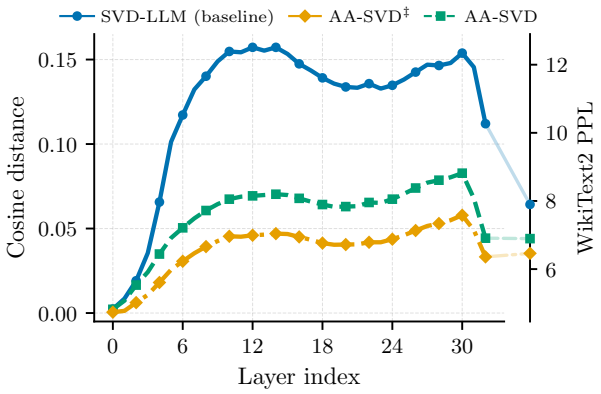


Figure 1. Distortion between *intermediate* features of the original and compressed model, measured as avg. cosine distance on 64 WikiText2 test samples. Diagonal lines link each method's final-layer distortion to its WikiText2 perplexity (right axis). AA-SVD suppresses compression error consistently across depth; with learned capacity allocation (AA-SVD‡), capacity is allocated to where it matters most, further suppressing distortion and yielding improved downstream performance.

Reducing this cost has motivated three broad strategies. *Training from scratch* yields compact, task-specific models but sacrifices the broad generalization of large pretrained networks. *Distillation* (Hinton et al., 2015; Sanh et al., 2019) trains a smaller student to mimic a larger teacher, but requires substantial retraining compute and data. *Post-training compression* (Frantar et al., 2022; Dettmers et al., 2022; Wang et al., 2025c) applies compression after training using only a small calibration set, offering a practical path to enabling efficient deployment of pretrained models at scale.

Post-training compression encompasses complementary techniques, each targeting distinct forms of redundancy: *pruning* (Han et al., 2015; Frantar & Alistarh, 2023) removes redundant weights or structures; *quantization* (Dettmers et al., 2022; Lin et al., 2024) reduces numerical precision; and *low-rank factorization* (Denton et al., 2014; Yuan et al., 2023; Wang et al., 2025c) decomposes weight matrices into compact structured factors. Hybrid methods that combine these approaches (Saha et al., 2024) demonstrate that targeting multiple sources of redundancy simultaneously pre-

serves downstream performance at increasingly aggressive compression ratios, exceeding what any single technique can achieve alone. Better low-rank compression creates an opportunity: paired with quantization or pruning, stronger low-rank factors could push the overall compression frontier to new limits.

Among these, low-rank factorization is particularly appealing: it yields dense structured factors compatible with standard hardware, reduces both parameters and FLOPs, and requires only a small calibration set. A fundamental challenge is which directions to preserve. Naïve truncation ignores activations entirely (Halko et al., 2011); activation-aware methods account for original activations but are blind to the distribution shift introduced by upstream compression (Yuan et al., 2023; Wang et al., 2025c; Li et al., 2025); and shift-aware methods condition on modified inputs but risk drifting from the original network's behavior (Wang et al., 2025a).

AA-SVD addresses these limitations through a two-stage compression framework. Each layer is first initialized with an activation-aware low-rank factorization. A *block-level refinement* step then jointly optimizes all compressed layers within each transformer block, minimizing the distortion between the original block's outputs and the compressed block's outputs on the shifted inputs it will actually receive at inference time; this allows compressed layers to collectively compensate for accumulated errors while remaining anchored to the original network's behavior. Beyond homogeneous capacity allocation across layers, we introduce *dynamic capacity allocation*: during block refinement, we jointly learn a differentiable mask over the factorized dimensions alongside the low-rank factors, enabling the model to automatically allocate rank capacity where it is most needed within each block. Figure 1 illustrates how AA-SVD consistently suppresses compression error across depth compared to prior methods.

2. Related Work

SVD-based post-training compression methods decompose weight matrices into dense structured factors compatible with standard hardware, reducing both parameters and FLOPs by exploiting $(\mathbf{U}\mathbf{V}^\top)\mathbf{X} = \mathbf{U}(\mathbf{V}^\top\mathbf{X})$. Methods differ primarily in their compression objective; Figure 2 (left) gives a visual overview of four categories.

The simplest *input-agnostic* objective minimizes $\|\mathbf{W} - \mathbf{W}'\|_F^2$ directly in parameter space, solved in closed form by truncated SVD (Halko et al., 2011; Sainath et al., 2013). No data is needed and layers are fully independent, but activations in deep networks lie on a highly anisotropic subspace (Ortiz-Jiménez et al., 2020), so the preserved directions may not align with those that matter for downstream performance (Chen et al., 2021; Idelbayev & Carreira-Perpinán, 2020). *Input-aware* methods address this by minimizing $\|\mathbf{W}\mathbf{X} - \mathbf{W}'\mathbf{X}\|_F^2$, where $\mathbf{X}$ contains calibration activations from the *original* network, as adopted by DRONE (Chen et al., 2021), ASVD (Yuan et al., 2023), SVD-LLM (Wang et al., 2025c), AdaSVD (Li et al., 2025), SVD-LLM V2 (Wang et al., 2025b), and DipSVD (Ding et al., 2025). Because $\mathbf{X}$ is fixed, layers are decoupled and can be compressed in any order. However, as layers are compressed sequentially, later layers receive inputs increasingly different from $\mathbf{X}$; since input-aware methods ignore this shift, errors accumulate across depth.

Shift-aware methods close this gap by conditioning on the shifted activations $\mathbf{X}'$ produced by the partially compressed network, minimizing $\|\mathbf{W}\mathbf{X}' - \mathbf{W}'\mathbf{X}'\|_F^2$, as in Dobi-SVD (Wang et al., 2025a). The same sequential-conditioning principle underlies GPTQ (Frantar et al., 2022) and SparseGPT (Frantar & Alistarh, 2023). This mitigates error propagation but imposes a strict layer ordering and risks amplifying divergence when upstream compression has already degraded representations. Hu et al. (2026) and Sinha & Fleuret (2026) introduced a fourth, *anchored adaptive* objective — $\|\mathbf{W}\mathbf{X} - \mathbf{W}'\mathbf{X}'\|_F^2$ — which anchors the reconstruction target to the original outputs while conditioning the compressed layer on the shifted inputs it will actually receive, combining an uncorrupted reference signal with distribution-shift awareness. This extends to block-level refinement, where all layers in a block are jointly optimized to minimize block-output error (Figure 2, right).

Orthogonal to the choice of objective, compression quality depends critically on how rank is allocated across layers; uniform allocation ignores heterogeneity in compressibility and functional importance. ASVD (Yuan et al., 2023) proposed sensitivity-based rank search (STRS), measuring perplexity change per layer at different truncation levels — effective but expensive due to repeated full-model passes. Adaptive strategies such as AdaSVD (Li et al., 2025) leverage layer-importance signals to allocate more rank where needed, in line with importance-based pruning approaches such as ShortGPT (Men et al., 2024). SVD-LLM V2 (Wang et al., 2025b) instead proposed a heuristic that reallocates rank based on the truncation loss $\|\mathbf{W}\mathbf{X} - \mathbf{W}'\mathbf{X}\|_F^2$ observed after uniform compression pass. More principled analytical approaches have recently emerged: PGSVD (Solgi et al., 2025) formulates rank allocation as a bi-objective optimization and proves that a uniform error tolerance across layers yields near Pareto-optimal heterogeneous ranks without any training, while ZS-SVD (Abbasi et al., 2026) scores each singular component by its first-order sensitivity to the calibration loss and applies a greedy zero-sum selection rule, automatically inducing heterogeneous ranks under a global budget. OWL (Yin et al., 2024) and DLP (Chen et al., 2025) assign capacity based on outlier statistics of activations,

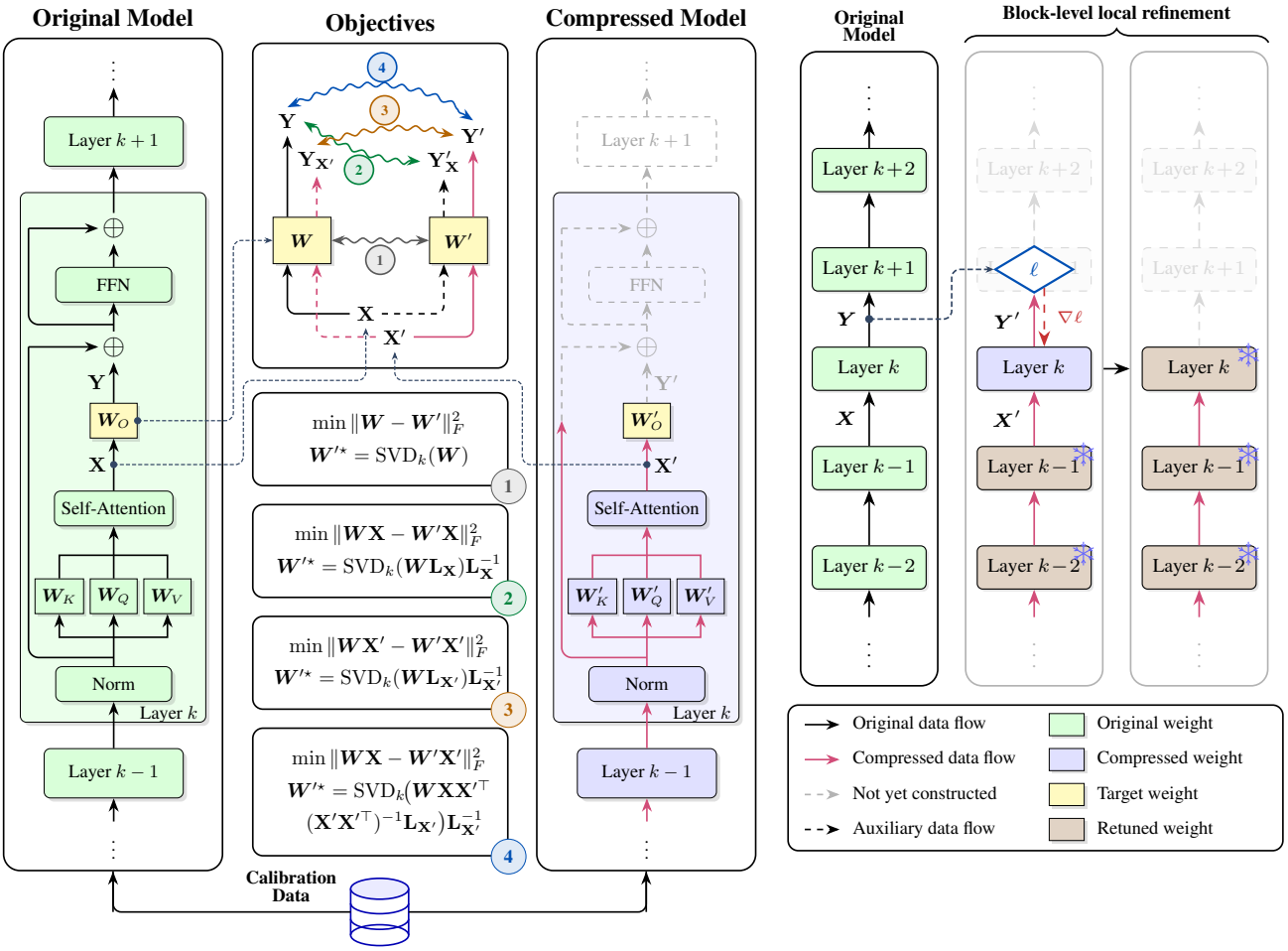


Figure 2. Overview of the two-stage compression pipeline. **Left:** Four layer-wise compression objectives, differing in which inputs and outputs are compared. (1) *Input-agnostic*: $\|\mathbf{W} - \mathbf{W}'\|_F^2$ — ignores activations entirely. (2) *Input-aware*: $\|\mathbf{W}\mathbf{X} - \mathbf{W}'\mathbf{X}\|_F^2$ — matches outputs on original inputs $\mathbf{X}$. (3) *Shift-aware*: $\|\mathbf{W}\mathbf{X}' - \mathbf{W}'\mathbf{X}'\|_F^2$ — matches outputs on the shifted inputs $\mathbf{X}'$ seen after upstream compression. (4) *Anchored adaptive* (ours): $\|\mathbf{W}\mathbf{X} - \mathbf{W}'\mathbf{X}'\|_F^2$ — anchors the target to the original output while conditioning on the shifted input. **Right:** Block-level refinement. Stage 1 factorizes each linear layer independently. Stage 2 jointly optimizes all factorized weights to minimize the block-output error $\ell = \|\mathcal{L}(\mathbf{X}) - \mathcal{L}'(\mathbf{X}')\|^2$, letting compressed layers compensate for each other's residual errors.

while FARMS (Hu et al., 2025) utilizes heavy-tailedness of weight matrices to locate undertrained layers and allocates capacity accordingly.

Mask-based and learned allocation methods optimize rank selection via gradient descent. ARS (Gao et al., 2024), LLRC (Sundrani et al., 2025), and ARA (Xv et al., 2025) learn per-component masks over precomputed SVD factors while keeping them frozen; Dobi-SVD (Wang et al., 2025a) learns a differentiable truncation point per module jointly with the task loss, then computes factors in a separate stage via IPCA. In this work, we incorporate learned allocation directly into block-level refinement, jointly optimizing compressed weights and rank distribution.

3. AA-SVD

AA-SVD bridges these perspectives through three ideas. We model the network as a sequence of transformer blocks, each containing multiple linear layers, and compress at two granularities: weight matrices individually, then transformer blocks jointly. Given a small calibration set, we denote by $\mathbf{X}$ the activations of a component in the original network and by $\mathbf{X}'$ the shifted activations it receives after upstream layers have been compressed; each weight matrix $\mathbf{W}$ is approximated by a rank-k factorization $\mathbf{W}' = \mathbf{U}\mathbf{V}^\top$ with $\mathbf{U} \in \mathbb{R}^{m\times k}$, $\mathbf{V} \in \mathbb{R}^{n\times k}$. First, a faithfully compressed layer must produce outputs close to the original while operating on the shifted inputs it will actually receive at inference time. Second, per-layer errors compound across layers within a block, making joint block-level optimization essential. Third, since layers within a block vary in

compressibility, we introduce *dynamic capacity allocation*: during block refinement, we jointly learn a differentiable mask over the factorized dimensions, automatically concentrating capacity where it is most needed.

We now establish the key mathematical results underlying our approach. We begin with the classical Eckart–Young–Mirsky theorem, which characterizes the optimal low-rank approximation of a matrix in Frobenius norm, and then use it to derive a closed-form solution for the AA-SVD layer-wise objective.

Lemma 3.1 (Eckart–Young–Mirsky)**.** *Let* $\mathbf{W} \in \mathbb{R}^{m\times n}$ *with thin SVD* $\mathbf{W} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$*. Then*

$$\min_{\operatorname{rank}(\mathbf{W}')\le k} \|\mathbf{W} - \mathbf{W}'\|_F^2 = \sum_{i>k} \sigma_i(\mathbf{W})^2,$$

and the unique minimizer is $\mathbf{W}'^\star = \operatorname{SVD}_k(\mathbf{W}) = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top$*, the truncation to the top-*k *singular components.*

Theorem 3.2. *Let* $\mathbf{W} \in \mathbb{R}^{m\times n}$ *be a fixed weight matrix and* $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times l}$ *be any two matrices. Fix a target rank* $k \in \mathbb{N}$*. Consider the optimization problem*

$$\min_{\operatorname{rank}(\mathbf{W}')\le k} \left\| \mathbf{W}\mathbf{A} - \mathbf{W}'\mathbf{B} \right\|_F^2. \tag{1}$$

Suppose $\mathbf{B}\mathbf{B}^\top$ *is invertible, and let* $\mathbf{L}_\mathbf{B}$ *be any invertible matrix satisfying* $\mathbf{B}\mathbf{B}^\top = \mathbf{L}_\mathbf{B}\mathbf{L}_\mathbf{B}^\top$[1]*. Then an optimal solution to* (1) *is*

$$\mathbf{W}'^\star \;=\; \operatorname{SVD}_k\!\Big(\mathbf{W}\mathbf{A}\mathbf{B}^\top\big(\mathbf{B}\mathbf{B}^\top\big)^{-1}\mathbf{L}_\mathbf{B}\Big)\,\mathbf{L}_\mathbf{B}^{-1},$$

where $\operatorname{SVD}_k(\cdot)$ *denotes the best rank-*k *approximation given by Lemma 3.1.*

Proof. See Appendix A. □

3.1. Linear Layer Compression

Our goal is to compress each linear transformation while ensuring that the resulting network remains *locally faithful* to the original model under the inputs it will actually encounter. Concretely, for a weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ with original inputs $\mathbf{X} \in \mathbb{R}^{n\times l}$ and shifted inputs $\mathbf{X}' \in \mathbb{R}^{n\times l}$ (after upstream compression), we seek a low-rank approximation $\mathbf{W}' \in \mathbb{R}^{m\times n}$ that solves

$$\min_{\mathbf{W}':\operatorname{rank}(\mathbf{W}')=k} \|\mathbf{W}\mathbf{X} - \mathbf{W}'\mathbf{X}'\|_F^2.$$

This objective enforces that the compressed outputs $\mathbf{W}'\mathbf{X}'$ stay close to the original outputs $\mathbf{W}\mathbf{X}$, anchoring the compressed network to the behavior of the uncompressed one while simultaneously adapting to the shifted input distribution. By explicitly constraining $\operatorname{rank}(\mathbf{W}') = k$, the problem is well-posed as a low-rank regression: we seek the best rank–k approximation of the mapping from $\mathbf{X}'$ to $\mathbf{W}\mathbf{X}$. This admits a closed-form solution as shown in Theorem 3.2. Figure 2 (left) illustrates the per-layer compression stage.

The solution operates only on the covariance matrices $\mathbf{X}\mathbf{X}'^\top$ and $\mathbf{X}'\mathbf{X}'^\top$, not on raw activations, so its cost is independent of the number of calibration tokens. We summarize the procedure in Algorithm 1 and provide further details in Appendix B.1. In practice, when followed by block-level refinement, we use the simpler input-aware objective ($\mathbf{X}' = \mathbf{X}$, Corollary A.1) as initialization, as it provides a better starting point for joint block optimization (see Section D).

3.2. Block-Level Local Refinement

Although each linear layer is compressed to minimize its own output error, the errors introduced by different layers within the same transformer block can interact. A small residual error at one layer shifts the activations seen by subsequent layers, so that even modest per-layer errors can compound into a larger distortion at the block output (see Figure 3). To address this, after all linear layers in a block have been compressed we introduce a *block-level local refinement* step. Concretely, for block $\mathcal{L}_i$ with original calibration inputs $\mathbf{X}$ and shifted inputs $\mathbf{X}'$ (received after upstream blocks are compressed), we minimize

$$\min_{\{\mathbf{U}_j,\mathbf{V}_j\},\,\boldsymbol{\theta}_i} \mathbb{E}_{\mathbf{X}\sim\mathcal{D}_i}\left[\|\mathcal{L}_i(\mathbf{X}) - \mathcal{L}_i'(\mathbf{X}')\|^2\right],$$

where $\mathcal{D}_i$ denotes the distribution of input activations to block $\mathcal{L}_i$ induced by the calibration data, $\mathcal{L}_i'$ denotes the block with each linear layer $\mathbf{W}_j$ replaced by its factorized approximation $\mathbf{U}_j\mathbf{V}_j^\top$, and $\boldsymbol{\theta}_i$ denotes the remaining trainable parameters of the block (e.g., normalization scales and biases). The optimization is thus over all factorized weights and block-local parameters jointly. This allows the compressed layers within a block to collectively compensate for one another's residual errors. The objective is minimized via gradient-based optimization. Because the refinement is confined to a single block and uses only a small calibration set, it adds only a modest overhead while substantially recovering block-output fidelity. Figure 2 (right) illustrates the block-level refinement stage.

3.3. Capacity Allocation

Different linear layers within a transformer block vary substantially in their compressibility (Yuan et al., 2023; Li et al., 2025): some tolerate aggressive rank reduction with little output distortion, while others are highly sensitive. The activation-weighted singular spectrum of each layer is likely correlated with its compressibility — layers with rapidly

[1] Such a decomposition can be found using Cholesky decomposition or eigenvalue decomposition.

Table 1. Comparison of AA-SVD with SOTA methods for SVD-based compression of LLaMA-7B at compression ratios 0.8, 0.6, and 0.4, on three language modeling benchmarks and seven commonsense reasoning tasks (zero-shot). Methods are grouped into two blocks per ratio: *homogeneous* (uniform rank allocation: SVD-LLM, SAES-SVD, AA-SVD) and *heterogeneous* (non-uniform or learned capacity allocation, marked ‡: ASVD, Dobi-SVD, Dip-SVD, AA-SVD‡), where AA-SVD‡ extends block-level refinement with jointly learned rank distribution. Best performance per block is marked in bold. (†) uses LoRA fine-tuning; results for baselines are taken from original papers where available.

Ratio	Method	PPL (↓)			Accuracy (↑)								
		Wiki2	PTB	C4	Openb.	ARC_e	ARC_c	WinoG.	PIQA	MathQA	HellaS.	Avg.	Drop (%)
1.0	Dense	5.68	8.34	7.34	0.34	0.75	0.42	0.69	0.79	0.27	0.57	0.55	–
0.8	SVD-LLM†	7.94	16.22	15.84	0.22	0.58	0.29	0.63	0.69	0.24	0.43	0.44	19.6%
	SAES-SVD	7.17	15.16	13.77	0.29	0.68	**0.36**	0.65	**0.75**	0.25	0.45	0.49	10.4%
	AA-SVD	**6.89**	**12.30**	**12.04**	**0.31**	**0.71**	**0.36**	**0.66**	0.72	**0.25**	**0.48**	**0.50**	**8.9%**
	ASVD‡	11.14	16.55	15.93	0.25	0.53	0.27	0.64	0.68	0.24	0.41	0.43	21.1%
	Dobi-SVD‡	8.54	14.83	**10.01**	0.26	0.59	0.31	0.66	0.70	0.23	0.44	0.46	16.7%
	Dip-SVD‡	7.95	15.60	14.07	0.27	0.63	0.33	0.64	0.71	0.24	0.45	0.47	14.6%
	AA-SVD‡	**6.46**	**10.44**	10.28	**0.30**	**0.72**	**0.38**	**0.68**	**0.75**	**0.25**	**0.50**	**0.51**	**7.0%**
0.6	SVD-LLM†	13.11	63.75	49.83	0.19	0.42	0.25	0.58	0.60	0.21	0.33	0.37	32.6%
	SAES-SVD	10.42	45.13	32.79	0.23	0.50	0.29	0.62	**0.65**	0.23	0.36	0.41	24.8%
	AA-SVD	**8.35**	**24.94**	**18.97**	**0.26**	**0.62**	**0.31**	**0.62**	**0.65**	**0.23**	**0.41**	**0.44**	**19.1%**
	ASVD‡	1407	3292	1109	0.13	0.28	0.22	0.48	0.55	0.19	0.26	0.30	44.9%
	Dobi-SVD‡	13.54	46.38	23.54	0.22	0.41	0.27	0.58	0.61	0.23	0.34	0.38	30.5%
	Dip-SVD‡	12.76	46.95	34.35	0.22	0.50	0.30	0.61	0.64	0.22	0.36	0.41	25.6%
	AA-SVD‡	**7.93**	**20.99**	**16.96**	**0.26**	**0.62**	**0.31**	**0.63**	**0.68**	**0.23**	**0.42**	**0.45**	**18.1%**
0.4	SVD-LLM†	53.74	438.58	383.07	0.14	0.28	0.22	0.50	0.55	0.21	0.27	0.31	43.3%
	SAES-SVD	22.01	116.83	93.97	0.16	0.33	**0.25**	0.52	0.54	**0.23**	0.30	0.33	39.2%
	AA-SVD	**13.67**	**74.64**	**46.14**	**0.19**	**0.44**	0.23	**0.55**	**0.60**	**0.23**	**0.32**	**0.37**	**33.2%**
	ASVD‡	57057	45218	43036	0.12	0.26	0.21	0.49	0.53	0.18	0.26	0.29	46.5%
	Dobi-SVD‡	46.18	238.91	190.62	0.15	0.31	0.20	0.52	0.54	0.22	0.28	0.32	42.0%
	AA-SVD‡	**11.89**	**63.88**	**36.97**	**0.21**	**0.46**	**0.24**	**0.57**	**0.61**	**0.22**	**0.34**	**0.38**	**31.1%**

decaying spectra tend to be more compressible, while those with flatter spectra require more rank to preserve output fidelity — suggesting that spectral structure can inform allocation. Existing learned allocation methods — ARS (Gao et al., 2024), LLRC (Sundrani et al., 2025), ARA (Xv et al., 2025), and Dobi-SVD (Wang et al., 2025a) — learn rank allocation via gradient descent on a composite objective combining a model-level task loss with an explicit ratio penalty. ARS, LLRC, and ARA model allocation as a soft mask over singular values, trained over a fixed precomputed decomposition while keeping the low-rank factors frozen. Dobi-SVD instead models allocation as a learnable truncation point k per module, and is two-stage: allocation is optimized first, after which linear layers are compressed with the fixed ranks via a separate factorization step. In all these methods, the task loss always favors retaining more singular components — whether modeled as a truncation point or a per-component mask — providing no directional differentiation between layers. The ratio penalty is thus the primary force driving allocation, but introduces a competing objective whose balance with the task loss is a sensitive hyperparameter, making it difficult to let data-driven rank selection emerge naturally.

We integrate dynamic capacity allocation directly into block-level refinement, eliminating the ratio penalty. For a block containing N linear layers with weight shapes (m_i, n_i), we introduce a learnable vector $\boldsymbol{\delta} \in \mathbb{R}^N$, initialized at $\mathbf{0}$, that parameterizes how a fixed parameter budget $B = \rho \sum_i m_i n_i$ is distributed across layers:

$$s_i = \log(m_i n_i) + \delta_i,$$
$$\boldsymbol{\alpha} = \mathrm{softmax}(\mathbf{s}),$$
$$k_i = \frac{\alpha_i B}{m_i + n_i}.$$

The softmax construction satisfies the budget constraint $\sum_i k_i(m_i + n_i) = B$ exactly for any $\boldsymbol{\delta}$, so no ratio penalty is needed. The $\log(m_i n_i)$ offset in s_i ensures that at $\boldsymbol{\delta} = \mathbf{0}$, $\boldsymbol{\alpha} \propto m_i n_i$, recovering a homogeneous allocation at exactly ρ for every layer regardless of shape. When k_i exceeds the break-even threshold $\bar{k}_i = m_i n_i/(m_i + n_i)$ — the rank at which the factorized form uses more parameters than dense storage — the module runs at full rank and the freed budget is redistributed to the remaining layers via a masked softmax over non-saturated modules.

With the budget satisfied by construction, MSE on block outputs is the sole training signal driving allocation. The gradient for k_i reflects the sensitivity of block-output distortion to the marginal singular component of layer i, naturally directing more capacity toward layers where retaining additional components most reduces distortion. Because the

Table 2. Comparison of AA-SVD against rank allocation baselines for SVD-based compression of LLaMA-2-7B and Qwen3-8B at a compression ratio of 0.6, evaluated on two language modeling benchmarks and seven commonsense reasoning tasks (zero-shot). Methods are grouped into two blocks: *homogeneous* (uniform rank across all layers: SVD-LLM, SAES-SVD, AA-SVD) and *heterogeneous* (non-uniform or dynamic rank, marked ‡: STRS, DLP, FARMS, ARS, Dobi-SVD, ARA, AA-SVD‡). **Bold** marks the best result within each block; ***bold italic*** marks the best overall for that model. SAES-SVD is omitted for Qwen3-8B as results were not reported in the original work. Results for heterogeneous baselines are taken from Xv et al. (2025).

Model	Method	PPL (↓)		Accuracy (↑)								
		Wiki2	C4	Openb.	ARC_e	ARC_c	WinoG.	PIQA	MathQA	HellaS.	Avg.	Drop (%)
LLaMA-2-7B	Dense	5.47	7.28	0.32	0.76	0.43	0.69	0.78	0.28	0.57	0.55	–
	SVD-LLM	16.47	73.12	0.18	0.39	0.21	0.53	0.58	0.22	0.31	0.35	36.8%
	SAES-SVD	11.35	40.57	–	0.43	–	0.58	0.59	–	0.32	–	–
	AA-SVD	**8.55**	**21.78**	***0.27***	**0.60**	***0.30***	***0.62***	**0.66**	***0.25***	***0.41***	**0.44**	**18.8%**
	STRS‡	18.12	82.58	0.16	0.37	0.22	0.54	0.57	0.22	0.29	0.34	38.1%
	DLP‡	14.24	55.85	0.20	0.39	0.22	0.57	0.60	0.22	0.32	0.36	34.2%
	FARMS‡	15.49	65.58	0.19	0.39	0.22	0.56	0.58	0.22	0.31	0.35	35.5%
	ARS‡	14.49	59.65	0.20	0.41	0.22	0.57	0.59	0.22	0.32	0.36	33.9%
	Dobi-SVD‡	12.95	41.93	0.20	0.46	0.23	0.57	0.62	0.22	0.33	0.38	31.3%
	ARA‡	10.85	27.57	0.23	0.56	0.26	0.61	0.64	0.23	0.36	0.41	24.5%
	AA-SVD‡	***7.99***	***19.17***	**0.26**	***0.63***	**0.29**	***0.62***	***0.67***	***0.25***	***0.41***	***0.45***	***18.3%***
Qwen3-8B	Dense	9.72	15.54	0.32	0.84	0.55	0.68	0.76	0.50	0.57	0.60	–
	SVD-LLM	29.15	111.57	0.15	0.40	0.22	0.54	0.60	0.23	0.31	0.35	41.9%
	AA-SVD	**15.67**	**45.04**	***0.26***	**0.60**	**0.31**	***0.63***	**0.64**	**0.26**	**0.38**	**0.44**	**27.0%**
	STRS‡	–	–	0.12	0.27	0.19	0.49	0.52	0.21	0.27	0.30	50.9%
	DLP‡	27.78	117.86	0.18	0.43	0.23	0.57	0.60	0.23	0.32	0.37	39.3%
	FARMS‡	34.50	168.17	0.16	0.41	0.22	0.54	0.59	0.23	0.31	0.35	41.7%
	ARS‡	52.00	235.32	0.13	0.32	0.19	0.51	0.57	0.21	0.29	0.32	47.4%
	Dobi-SVD‡	18.65	64.08	0.24	0.55	0.30	0.60	0.64	0.24	0.36	0.42	30.6%
	ARA‡	16.34	47.71	0.22	0.59	0.31	0.57	***0.66***	0.25	0.36	0.42	29.9%
	AA-SVD‡	***14.90***	***41.45***	***0.26***	***0.62***	***0.33***	**0.62**	***0.66***	***0.28***	***0.39***	***0.45***	***25.1%***

softmax couples all layers' allocations within a block, this signal induces genuine competition: rank flows toward layers where additional capacity most reduces block-output distortion, and away from those where it matters less.

Prior to optimization, each layer's factorization is expanded to full rank $r_i = \min(m_i, n_i)$: COMPRESSLAYER retains the tail factors $\mathbf{U}_{\text{tail}}, \mathbf{V}_{\text{tail}}$ beyond the initial rank, so that if allocation increases a layer's rank during refinement, the additional components have a principled initialization rather than starting from random values.

To enable gradient flow to $\boldsymbol{\delta}$, each layer's forward pass uses the soft tanh mask of Wang et al. (2025a):

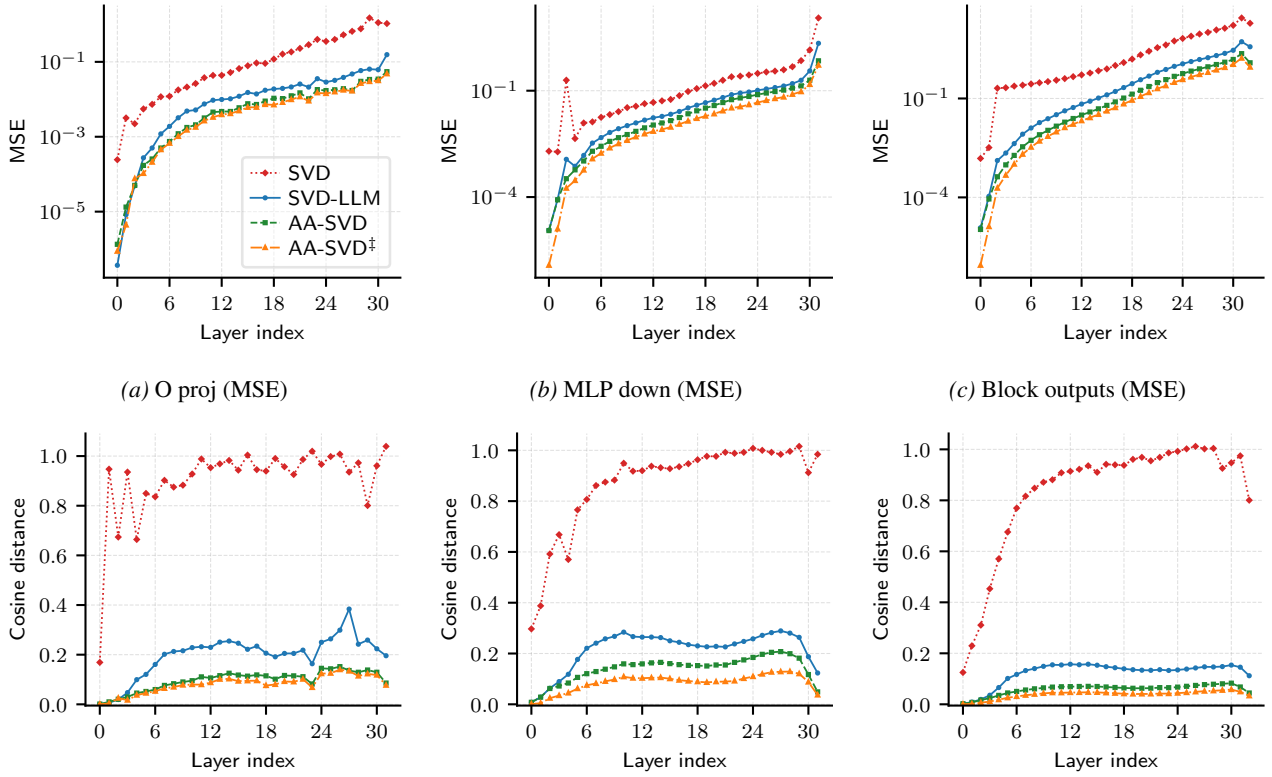
$$\hat{\mathbf{y}}_i = \mathbf{U}_i(\boldsymbol{\mu}_i \odot \mathbf{V}_i\mathbf{x}),$$
$$(\mu_i)_\ell = \tfrac{1}{2}\tanh\big(\beta(k_i - \ell)\big) + \tfrac{1}{2}, \quad \ell \in [0, r_i).$$

Training proceeds in two phases. In *Phase 1*, the low-rank factors $\{\mathbf{U}_i, \mathbf{V}_i\}$ and allocation logits $\boldsymbol{\delta}$ are jointly optimized. In *Phase 2*, $\boldsymbol{\delta}$ is frozen and the tanh mask is replaced by a hard binary mask $\mathbf{1}[\ell < \lfloor k_i \rceil]$; the UV factors are refined further at the converged rank boundaries. The hard mask during Phase 2 is identical to what is applied at inference, eliminating any train-deploy gap. Finally, $\mathbf{U}_i$ and $\mathbf{V}_i$ are trimmed to the learned rank k_i, yielding a compressed block whose total parameter count roughly matches B.

The complete end-to-end compression procedure is described in Algorithm 2, which processes the model block by block: within each block, COMPRESSLAYER (Algorithm 1) is applied to each linear layer in sequence, after which the block-level refinement step is performed before moving to the next block. Further implementation details and hyperparameters are provided in Appendix B.

4. Experiments

We evaluate AA-SVD across a diverse set of open-source pretrained language models, spanning multiple architecture families and parameter scales. Concretely, we compress models from the LLaMA (Touvron et al., 2023) and Qwen (Bai et al., 2023) families, which together cover a broad range of model sizes and training recipes representative of the current landscape. For calibration, we follow prior work (Wang et al., 2025c) and use 256 samples drawn from the WikiText2 (Merity et al., 2016) training split unless otherwise stated; our ablations show this modest budget is sufficient for stable compression (Appendix C). Memory and inference speedup analysis is provided in Appendix B.4. Compressed models are then evaluated along two axes: *language modeling perplexity* on WikiText2, C4 (Raffel et al., 2020), and PTB (Marcinkiewicz, 1994), which measures



(a) O proj (MSE) *(b)* MLP down (MSE) *(c)* Block outputs (MSE)

(d) O proj (cosine distance) *(e)* MLP down (cosine distance) *(f)* Block outputs (cosine distance)

Figure 3. Layer-wise error evolution across LLaMA-7B at ratio 0.8, evaluated on WikiText2 test split samples. Top row: MSE between original and compressed outputs. Bottom row: cosine distance between original and compressed outputs. Results are shown separately for attention output projections (O-proj), MLP-down projections, and full block outputs.

how well the model preserves distributional fidelity; and *zero-shot accuracy on commonsense reasoning* benchmarks — Winogrande (Sakaguchi et al., 2020), PIQA (Bisk et al., 2020), ARC-Easy and ARC-Challenge (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), HellaSwag (Zellers et al., 2019) and MathQA (Amini et al., 2019) — which captures practical downstream utility.

4.1. Main Results

Table 1 presents a detailed comparison on LLaMA-7B across compression ratios 0.8, 0.6, and 0.4, on three perplexity benchmarks and seven zero-shot commonsense reasoning tasks. Methods are grouped into two blocks: *homogeneous* baselines using uniform rank allocation (SVD-LLM, SAES-SVD) compared against AA-SVD, and *heterogeneous* baselines using non-uniform or dynamic allocation (ASVD, Dobi-SVD, Dip-SVD) compared against AA-SVD‡, which extends block-level refinement with jointly learned rank distribution. Table 2 further situates both variants against a broader set of rank allocation methods on LLaMA-2-7B and Qwen3-8B at ratio 0.6. Table 6 reports aggregated results across five additional models spanning the LLaMA-2, LLaMA-3, and Qwen-2.5 families; expanded per-benchmark breakdowns are provided in Appendix G.

Homogeneous allocation. Against uniform-rank baselines, AA-SVD achieves the best perplexity and average accuracy at all three ratios. At ratio 0.8, the nearest competitor (SAES-SVD) incurs a noticeably larger accuracy drop. The margin widens under heavier compression: at ratio 0.6, AA-SVD reduces perplexity substantially across all three benchmarks while matching or exceeding SAES-SVD on every reasoning task. At ratio 0.4, SVD-LLM degrades severely while AA-SVD continues to produce functional compressed models, cutting the accuracy drop by roughly six percentage points relative to SAES-SVD.

Heterogeneous allocation with learned capacity. AA-SVD‡ jointly optimizes compressed weights and rank distribution during block refinement, automatically concentrating capacity in layers where it is most beneficial. This yields consistent further gains over the heterogeneous baselines: at ratio 0.8 it outperforms Dip-SVD and Dobi-SVD on all metrics; at ratio 0.6, ASVD collapses to near-random perplexity while AA-SVD‡ maintains a functional model; at

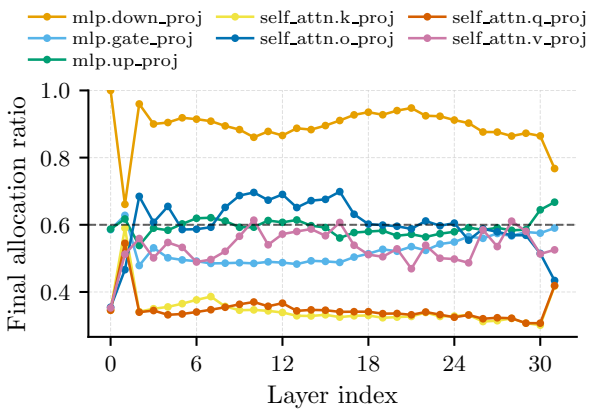


Figure 4. Learned rank allocation ratio per projection type across the 32 layers of LLaMA-2-7B at a target compression ratio of 0.6 (dashed line).

ratio 0.4 it retains the lowest accuracy drop of all methods.

Situating against rank allocation baselines. Table 2 broadens the heterogeneous comparison to LLaMA-2-7B and Qwen3-8B at ratio 0.6, evaluating AA-SVD‡ against six dedicated rank allocation strategies—STRS, DLP, FARMS, ARS, Dobi-SVD, and ARA. Across both models and all benchmarks, AA-SVD‡ achieves the lowest perplexity and highest average accuracy among all heterogeneous methods, with the best accuracy drop in both settings. The homogeneous variant AA-SVD likewise leads the uniform-rank block. These results confirm that the gains from block-level joint optimization and learned capacity allocation are robust across model families and hold up against methods specifically designed for rank allocation.

Generalization across architectures. The gains generalize broadly across architectures (Table 6). AA-SVD outperforms SVD-LLM on every model family at both evaluated ratios, with the largest gap on LLaMA-3-1B, where SVD-LLM's perplexity is roughly three times higher than AA-SVD's—suggesting compact modern architectures are especially sensitive to per-layer approximation error and benefit most from block-level joint optimization. At ratio 0.6, SVD-LLM collapses on both LLaMA-3 models, while AA-SVD retains functional representations throughout. These results consistently demonstrate state-of-the-art performance across ratios, metrics, and model families, with gains most pronounced precisely where competing methods fail. Appendix E further situates AA-SVD against structured pruning methods.

4.2. Ablations and Analysis

Error Evolution Across Layers. Figure 3 tracks MSE and cosine distance between original and compressed outputs across depth on LLaMA-7B at ratio 0.8, comparing AA-SVD and AA-SVD‡ against naive SVD and SVD-LLM (see also Figure 1). Naive SVD fails catastrophically: cosine distance saturates near 1.0 from the first layers. SVD-LLM recovers substantially but its error grows monotonically with depth, plateauing at $\sim$0.13–0.15 block-level cosine distance, and spikes sharply on attention output projections in the final layers. AA-SVD achieves consistently lower error: block-level cosine distance rises to only $\sim$0.07–0.08 and plateaus, maintaining a clear margin over SVD-LLM at every depth. These results confirm that anchoring to original outputs while accounting for shifted inputs curbs error accumulation, directly underlying the stronger end-task performance. AA-SVD‡ further reduces block-level distortion to $\sim$0.04–0.05 throughout, roughly halving the uniform-rank variant's cosine distance. The gains are concentrated on MLP down projections, where additional rank substantially reduces distortion, while O-proj error remains comparable across both variants—consistent with the learned allocation in Figure 4.

Rank Allocation. Figure 4 shows the learned per-layer rank allocation for LLaMA-2-7B at ratio 0.6. The allocation is strikingly heterogeneous across projection types but largely stable across depth. MLP down projections consistently receive far more capacity than the target budget (allocation ratio $\approx$ 0.85–1.0), indicating that the optimizer identifies them as critical bottlenecks that resist compression. In contrast, query and key projections are heavily pruned well below the budget ($\approx$ 0.30–0.35), consistent with the known redundancy of attention head representations. The output projection and MLP gate/up projections cluster near the target ratio of 0.6, suggesting moderate compressibility. This pattern holds almost uniformly across the middle layers (indices 2–29), with notably higher variance in the first and last few layers. The learned allocation thus departs substantially from a naive uniform assignment: rather than spreading capacity evenly, the model concentrates rank where the reconstruction loss is most sensitive and frees it from layers where information can be discarded cheaply.

5. Conclusion

We presented AA-SVD, a post-training SVD compression framework with three contributions: (1) an *anchored adaptive* layer objective that conditions on shifted inputs from upstream compression to prevent error propagation; (2) *block-level local refinement* that jointly optimizes all layers in a block, suppressing cumulative approximation error across depth; and (3) *learned capacity allocation* that differentiably concentrates rank budget where reconstruction loss is most sensitive. Together, these yield state-of-the-art performance across model families and compression ratios, with gains most pronounced at aggressive compression ratios over competing methods.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Abbasi, A., Thrash, C., Qin, H., Sharma, S., Seifi, S., and Kolouri, S. Zero sum svd: Balancing loss sensitivity for low rank llm compression. *arXiv preprint arXiv:2602.02848*, 2026.

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Amini, A., Gabriel, S., Lin, P., Koncel-Kedziorski, R., Choi, Y., and Hajishirzi, H. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. *arXiv preprint arXiv:1905.13319*, 2019.

Ashkboos, S., Croci, M. L., Nascimento, M. G. d., Hoefler, T., and Hensman, J. Slicegpt: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*, 2024.

Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Bisk, Y., Zellers, R., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Chen, P., Yu, H.-F., Dhillon, I., and Hsieh, C.-J. Drone: Data-aware low-rank compression for large nlp models. *Advances in neural information processing systems*, 34: 29321–29334, 2021.

Chen, Y., Cheng, B., Han, J., Zhang, Y., Li, Y., and Zhang, S. Dlp: Dynamic layerwise pruning in large language models. In *International Conference on Machine Learning*, pp. 7934–7956. PMLR, 2025.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. In *Proceedings of EMNLP*, pp. 279–290, 2018.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, volume 27, 2014.

Dettmers, T., Lewis, M., Shleifer, S., and Zettlemoyer, L. Llm.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*, 2022.

Ding, X., Sun, R., Zhang, Y., Yan, X., Zhou, Y., Huang, K., Fu, S., Xie, C., and Zhu, Y. Dipsvd: Dual-importance protected svd for efficient llm compression. *arXiv preprint arXiv:2506.20353*, 2025.

Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023.

Frantar, E., Ashkboos, S., Stock, P., and Alistarh, D. Gptq: Accurate post-training quantization for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Gao, S., Hua, T., Hsu, Y.-C., Shen, Y., and Jin, H. Adaptive rank selections for low-rank approximation of language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 227–241, 2024.

Halko, N., Martinsson, P.-G., and Tropp, J. A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural networks. In *Advances in neural information processing systems*, volume 28, 2015.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. In *NeurIPS Deep Learning Workshop*, 2015.

Hu, X., Yang, D., Cheng, Y., Chen, Z., and Xu, Z. Saes-svd: Self-adaptive suppression of accumulated and local errors for svd-based llm compression. *arXiv preprint arXiv:2602.03051*, 2026.

Hu, Y., Goel, K., Killiakov, V., and Yang, Y. Eigenspectrum analysis of neural networks without aspect ratio bias. In *International Conference on Machine Learning*, pp. 24290–24313. PMLR, 2025.

Idelbayev, Y. and Carreira-Perpinán, M. A. Low-rank compression of neural nets: Learning the rank of each layer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8049–8059, 2020.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kolawole, S., Dery, L., Kagy, J.-F., Smith, V., Neubig, G., and Talwalkar, A. Everybody prune now: Structured pruning of llms with only forward passes. *arXiv preprint arXiv:2402.05406*, 2024.

Li, Z., Xia, M., Zhang, J., Hui, Z., Qin, H., Kong, L., Zhang, Y., and Yang, X. Adasvd: Adaptive singular value decomposition for large language models. *arXiv preprint arXiv:2502.01403*, 2025.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

Marcinkiewicz, M. A. Building a large annotated corpus of english: The penn treebank. *Using Large Corpora*, 273: 31, 1994.

Men, X., Xu, M., Zhang, Q., Wang, B., Lin, H., Lu, Y., Han, X., and Chen, W. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*, 2024.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of EMNLP*, pp. 2381–2391, 2018.

Ortiz-Jiménez, G., Modas, A., Moosavi, S.-M., and Frossard, P. Neural anisotropy directions. *Advances in Neural Information Processing Systems*, 33:17896–17906, 2020.

Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L., Rothchild, D., So, D. R., Texier, M., and Dean, J. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. In *Journal of Machine Learning Research*, volume 21, pp. 1–67, 2020.

Saha, R., Sagan, N., Srivastava, V., Goldsmith, A. J., and Pilanci, M. Compressing large language models using low rank and low precision decomposition. *Advances in Neural Information Processing Systems*, 37:88981–89018, 2024.

Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., and Ramabhadran, B. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6655–6659. IEEE, 2013.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *arXiv preprint arXiv:1910.01108*, 2019.

Sinha, A. K. and Fleuret, F. Aa-svd: Anchored and adaptive svd for large language model compression. *arXiv preprint arXiv:2604.02119*, 2026.

Solgi, R., Madinei, P., Tian, J., Swaminathan, R., Liu, J., Susanj, N., and Zhang, Z. Activation-informed pareto-guided low-rank compression for efficient llm/vlm. *arXiv preprint arXiv:2510.05544*, 2025.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Sundrani, S., Tudisco, F., and Minervini, P. Low-rank compression of language models via differentiable rank selection. *arXiv preprint arXiv:2512.13733*, 2025.

Touvron, H., Martin, L., Stone, K., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Wang, Q., Ke, J., Tomizuka, M., Chen, Y., Keutzer, K., and Xu, C. Dobi-svd: Differentiable svd for llm compression and some new perspectives. In *ICLR*, 2025a.

Wang, X., Alam, S., Wan, Z., Shen, H., and Zhang, M. Svd-llm v2: Optimizing singular value truncation for large language model compression. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human*

Language Technologies (Volume 1: Long Papers), pp. 4287–4296, 2025b.

Wang, X., Zheng, Y., Wan, Z., and Zhang, M. 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.

Xv, L., Gao, J., Gao, X., Liu, T., and Fu, Y. Ara: Adaptive rank allocation for efficient large language model svd compression. *arXiv preprint arXiv:2510.19389*, 2025.

Yin, L., Wu, Y., Zhang, Z., Hsieh, C.-Y., Wang, Y., Jia, Y., Li, G., Jaiswal, A. K., Pechenizkiy, M., Liang, Y., et al. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity. In *International Conference on Machine Learning*, pp. 57101–57115. PMLR, 2024.

Yuan, Z., Shang, Y., Song, Y., Wu, Q., Yan, Y., and Sun, G. Asvd: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821*, 2023.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Zhong, L., Wan, F., Chen, R., Quan, X., and Li, L. Blockpruner: Fine-grained pruning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 5065–5080, 2025.

A. Proofs and Discussion

A.1. Proof of Theorem 3.2

Problem statement. Let $\mathbf{W} \in \mathbb{R}^{m\times n}$ be a fixed weight matrix and $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times l}$ be any two matrices with $\mathbf{B}\mathbf{B}^\top$ invertible. Fix a target rank $k \in \mathbb{N}$. We seek a solution to

$$\min_{\operatorname{rank}(\mathbf{W}')\le k} \left\| \mathbf{W}\mathbf{A} - \mathbf{W}'\mathbf{B} \right\|_F^2.$$

We claim that, for any invertible $\mathbf{L_B}$ satisfying $\mathbf{B}\mathbf{B}^\top = \mathbf{L_B}\mathbf{L_B}^\top$, an optimal solution is

$$\mathbf{W}'^\star \;=\; \mathrm{SVD}_k\Big(\mathbf{W}\mathbf{A}\mathbf{B}^\top\big(\mathbf{B}\mathbf{B}^\top\big)^{-1}\mathbf{L_B}\Big)\,\mathbf{L_B}^{-1}.$$

Proof. Since $\mathbf{B}\mathbf{B}^\top$ is symmetric positive definite, an invertible $\mathbf{L_B}$ with $\mathbf{B}\mathbf{B}^\top = \mathbf{L_B}\mathbf{L_B}^\top$ always exists; two concrete choices are (i) the lower-triangular Cholesky factor, or (ii) $\mathbf{Q}\mathbf{\Lambda}^{1/2}$ from the eigendecomposition $\mathbf{B}\mathbf{B}^\top = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$.

Expanding the squared Frobenius norm gives

$$\|\mathbf{W}\mathbf{A} - \mathbf{W}'\mathbf{B}\|_F^2 = \operatorname{tr}(\mathbf{W}'\mathbf{B}\mathbf{B}^\top\mathbf{W}'^\top) - 2\,\operatorname{tr}(\mathbf{W}\mathbf{A}\mathbf{B}^\top\mathbf{W}'^\top) + \|\mathbf{W}\mathbf{A}\|_F^2.$$

Since $\mathbf{B}\mathbf{B}^\top = \mathbf{L_B}\mathbf{L_B}^\top$, the first term equals $\|\mathbf{W}'\mathbf{L_B}\|_F^2$. Completing the square yields

$$\|\mathbf{W}'\mathbf{L_B} - \mathbf{W}\mathbf{A}\mathbf{B}^\top\mathbf{L_B}^{-\top}\|_F^2 - \|\mathbf{W}\mathbf{A}\mathbf{B}^\top\mathbf{L_B}^{-\top}\|_F^2 + \|\mathbf{W}\mathbf{A}\|_F^2.$$

Setting $\mathbf{M} := \mathbf{W}\mathbf{A}\mathbf{B}^\top(\mathbf{B}\mathbf{B}^\top)^{-1}\mathbf{L_B} = \mathbf{W}\mathbf{A}\mathbf{B}^\top\mathbf{L_B}^{-\top}$, minimizing (1) is equivalent to minimizing $\|\mathbf{W}'\mathbf{L_B} - \mathbf{M}\|_F^2$ subject to $\operatorname{rank}(\mathbf{W}'\mathbf{L_B}) \le k$. Because $\mathbf{L_B}$ is invertible, $\operatorname{rank}(\mathbf{W}'\mathbf{L_B}) = \operatorname{rank}(\mathbf{W}')$. By Lemma 3.1, the unique minimizer of $\|\mathbf{Z} - \mathbf{M}\|_F^2$ over rank-k matrices $\mathbf{Z}$ is $\mathrm{SVD}_k(\mathbf{M})$, so $\mathbf{W}'\mathbf{L_B} = \mathrm{SVD}_k(\mathbf{M})$, giving

$$\mathbf{W}'^\star = \mathrm{SVD}_k(\mathbf{M})\,\mathbf{L_B}^{-1},$$

and the minimal value $\|\mathbf{W}\mathbf{A}\|_F^2 - \|\mathbf{M}\|_F^2 + \sum_{i>k}\sigma_i(\mathbf{M})^2$ follows directly from Lemma 3.1. □

Remark (Rank-deficient $\mathbf{B}$). If $\mathbf{B}\mathbf{B}^\top$ is singular, an invertible $\mathbf{L_B}$ satisfying $\mathbf{B}\mathbf{B}^\top = \mathbf{L_B}\mathbf{L_B}^\top$ does not exist. In this case replace $\mathbf{L_B}^{-1}$ by the Moore–Penrose factor $(\mathbf{B}\mathbf{B}^\top)^{+1/2}$, or equivalently use a Tikhonov-regularized factorization $\mathbf{B}\mathbf{B}^\top + \varepsilon\mathbf{I} = \mathbf{L}_\varepsilon\mathbf{L}_\varepsilon^\top$ and let $\varepsilon \to 0^+$. The same argument then shows that $\mathbf{W}'^\star = \mathrm{SVD}_k(\mathbf{M})\,(\mathbf{B}\mathbf{B}^\top)^{+1/2}$ with $\mathbf{M} := \mathbf{W}\mathbf{A}\mathbf{B}^\top(\mathbf{B}\mathbf{B}^\top)^{+1/2}$ is a minimum-norm optimizer, with minimal value given by the same formula.

Corollary A.1 (No distribution shift)**.** *If* $\mathbf{B} = \mathbf{A}$*, then* $\mathbf{A}\mathbf{B}^\top = \mathbf{B}\mathbf{B}^\top = \mathbf{L_B}\mathbf{L_B}^\top$*, so* $\mathbf{M} = \mathbf{W}\mathbf{L_B}^\top$*. The solution reduces to* $\mathbf{W}'^\star = \mathrm{SVD}_k\big(\mathbf{W}\mathbf{L_B}\big)\,\mathbf{L_B}^{-1}$*, the standard whitening-based low-rank regression solution.*

A.2. Discussion of Corollary A.1

Corollary A.1 applies whenever $\mathbf{A} = \mathbf{B}$, so that $\mathbf{A}\mathbf{B}^\top = \mathbf{B}\mathbf{B}^\top = \mathbf{L_B}\mathbf{L_B}^\top$ and the general solution reduces to the whitening-based form

$$\mathbf{W}'^\star = \mathrm{SVD}_k\big(\mathbf{W}\mathbf{L_B}\big)\,\mathbf{L_B}^{-1}.$$

Two natural instantiations arise in our setting: setting $\mathbf{B} = \mathbf{X}$ (original inputs) yields an input-aware solution, while setting $\mathbf{B} = \mathbf{X}'$ (shifted inputs) gives a shift-aware variant that adapts to the upstream-compressed distribution. SVD-LLM (Wang et al., 2025c) and SVD-LLM V2 (Wang et al., 2025b) both correspond to the $\mathbf{B} = \mathbf{X}$ case, differing only in their factorization of $\mathbf{X}\mathbf{X}^\top$: SVD-LLM uses the lower-triangular Cholesky factor $\mathbf{L_X}$, while SVD-LLM V2 uses the eigendecomposition $\mathbf{X}\mathbf{X}^\top = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$ with $\mathbf{L_X} = \mathbf{Q}\mathbf{\Lambda}^{1/2}$, giving

$$\mathbf{W}'^\star = \mathrm{SVD}_k\big(\mathbf{W}\mathbf{Q}\mathbf{\Lambda}^{1/2}\big)\,\mathbf{\Lambda}^{-1/2}\mathbf{Q}^\top.$$

Since the official SVD-LLM V2 implementation was not publicly available at the time of writing, we reproduced it from the paper description. Our reproduction showed no discernible performance difference relative to SVD-LLM under either homogeneous or heterogeneous compression ratio settings; we therefore report SVD-LLM as the representative baseline for this line of work. This choice is consistent with more recent methods, including DipSVD (Ding et al., 2025) and SAES-SVD (Hu et al., 2026), which similarly do not report V2 results.

B. Implementation Details

B.1. Linear Layer Compression

Theorem 3.2 establishes that the optimal rank-k compressed operator is obtained by whitening the modified inputs $\mathbf{X}'$ via their covariance, projecting the cross-term $\mathbf{WX}$ into this whitened space, applying truncated SVD, and mapping back. This closed-form solution generalizes the classical whitening construction ($\mathbf{X}' = \mathbf{X}$) and can be implemented efficiently with a Cholesky factorization. Importantly, our formulation operates only on the covariance matrices $\mathbf{XX}'^\top$ and $\mathbf{X}'\mathbf{X}'^\top$ rather than the raw activations themselves. This is especially advantageous when the number of samples is large (e.g. in our setting with 256 samples of length 2048, corresponding to over half a million effective columns), since the covariance matrices are fixed-size $d \times d$ regardless of the batch length.

The pseudocode in Algorithm 1 details the implementation of the linear layer compression step for a single layer, which is applied sequentially across layers within each block. In practice, covariance matrices being computed in Step 2 can be implemented efficiently in batches and by additively accumulating the outer products ($\mathbf{XX}^\top$, $\mathbf{XX}'^\top$ and $\mathbf{X}'\mathbf{X}'^\top$), without explicitly materializing the full activation matrices. The Cholesky or eigenvalue decomposition in Step 3 is efficient for the moderate hidden dimensions of interest (e.g. $d = 2^{12} - 2^{16}$) with modern GPU-accelerated linear algebra libraries. Further, multiple linear layers can share the same covariance matrix if they operate on the same input distribution (e.g. query, key and value projections or MLP gate and up projections within the same block), so the covariance can be reused across layers to amortize the cost of Step 2 and Step 3.

B.2. Block-level Local Refinement

The block-level refinement step (Step 9 of Algorithm 2) jointly optimizes the low-rank factors $\{\mathbf{U}_j, \mathbf{V}_j\}$ and block-local parameters $\boldsymbol{\theta}_i$ to minimize the MSE between the original and compressed block outputs, as described in the main text. When used without capacity allocation, the objective is minimized via gradient-based optimization: we use the AdamW optimizer (Loshchilov & Hutter, 2017) with a learning rate of 10^{-4}, trained for 25 epochs over the calibration data with a cosine learning rate schedule and linear warmup, with a batch size of 32. In our experiments, we find this training configuration to be effective across model families and compression ratios, providing a good balance between recovery quality and computational cost.

Several steps of Algorithm 2 also admit straightforward implementation optimizations. Steps 1 and 10 compute the block-input activations $\mathbf{X}$ and $\mathbf{X}'$ for the original and compressed models, respectively; the size of these tensors scales with the number of calibration sequences and their length (e.g. $N_{\text{cal}} \times L \times d$), and can exceed GPU memory for larger calibration sets. In practice, these forward passes can be executed in batches on GPU with the resulting activations offloaded to CPU memory between blocks, keeping peak GPU memory usage bounded. Finally, since the block-level refinement in Step 9 is optimized via standard backpropagation, it can be carried out over batches of calibration sequences on GPU.

B.3. Capacity Allocation

The two-phase training procedure for capacity allocation extends block-level refinement as described in Section 3. Prior to optimization, each layer's factorization is expanded to full rank $r_i = \min(m_i, n_i)$, retaining the tail factors $\mathbf{U}_{\text{tail}}, \mathbf{V}_{\text{tail}}$ so that any rank increase during Phase 1 has a principled initialization.

In *Phase 1*, the low-rank factors $\{\mathbf{U}_i, \mathbf{V}_i\}$, block-local parameters $\boldsymbol{\theta}_i$, and allocation logits $\boldsymbol{\delta}$ are jointly optimized for 20 epochs using AdamW (Loshchilov & Hutter, 2017) with a batch size of 32; UV factors and block-local parameters use a learning rate of 10^{-4} with a cosine schedule and linear warmup, while $\boldsymbol{\delta}$ uses a fixed learning rate of 5×10^{-3}. The soft tanh mask ensures gradient flow to $\boldsymbol{\delta}$ throughout this phase. In *Phase 2*, $\boldsymbol{\delta}$ is frozen at its converged values and the tanh mask is replaced by a hard binary mask $\mathbf{1}[\ell < \lfloor k_i \rceil]$; the UV factors and block-local parameters are then refined for a further 10 epochs at learning rate 10^{-4} with the same cosine schedule and batch size, closing the gap between training and inference. Finally, $\mathbf{U}_i$ and $\mathbf{V}_i$ are trimmed to the learned rank $\lfloor k_i \rceil$, yielding a compressed block whose total parameter count roughly matches the target budget B.

B.4. Memory and Speedup

Low-rank factorization reduces both parameter count and compute cost by replacing a dense matrix with the product of two thin factors. Consider a linear layer $\mathbf{W} \in \mathbb{R}^{m \times n}$. The original layer requires mn parameters and $O(mn)$ FLOPs

per forward pass. A rank-k factorization stores $mk + nk$ parameters and incurs $O(mk + nk)$ FLOPs, which is cheaper whenever $k \ll \min(m, n)$. The effective compression ratio is

$$\rho = \frac{mk + nk}{mn}.$$

For example, with $m = n = 4096$ and $k = 512$ ($\rho = 0.125$), the parameter count drops from 16.8M to 4.2M (a $4\times$ reduction), and FLOPs per forward pass reduce by the same factor.

Beyond weights and FLOPs, low-rank factorization can also reduce the memory footprint of the key–value (KV) cache during autoregressive inference. Since attention projections are compressed, the activations stored in the cache scale with k rather than n, yielding proportional savings in both memory and bandwidth. As highlighted in SVD-LLM (Wang et al., 2025c) and follow-up works (Wang et al., 2025a; Hu et al., 2026), this reduction is crucial for long-context inference where KV-cache dominates memory usage.

Our method (AA-SVD) preserves this structural efficiency: the cost of computing compressed weights is incurred once during compression, while inference cost and KV-cache size match those of standard low-rank layers. Thus, AA-SVD offers the same runtime and memory benefits as prior SVD-based methods, with its main advantage lying in improved approximation quality under aggressive compression.

B.5. Dobi-SVD Remapping

Standard SVD-based compression stores a rank-k approximation of an $m \times n$ weight matrix as two dense factors of total size $k(m+n)$, giving a compression ratio $\rho = k(m+n)/(mn)$. Dobi-SVD (Wang et al., 2025a) proposes a *remapping* that stores the smaller factor and the top $\min(m, n)$ rows/columns of the larger factor in half precision (16-bit $\rightarrow$ 8-bit), with the remaining $(\max(m, n) - \min(m, n))$ rows/columns kept in full precision. The total storage in full-precision-equivalent units reduces to $\max(m, n) \cdot k (= 0.5 \cdot 2 \min(m, n) \cdot k + (\max(m, n) - \min(m, n)) \cdot k)$. This gives a compression ratio of $\rho = \max(m, n) \cdot k/(mn) = k/\min(m, n)$, so that every target ratio $\rho \in [0, 1]$ maps to a unique truncation rank $k = \rho \cdot \min(m, n)$, spanning the full valid range $k \in [0, \min(m, n)]$[2].

Because this remapping changes what a stated compression ratio means in terms of actual parameter counts, a direct comparison between Dobi-SVD and methods using the standard formula at the same nominal ratio is unfair. To address this, we report results both without remapping (standard formula, comparable across all methods) and with remapping enabled for AA-SVD, denoted AA-SVD^q, at the same effective parameter budget as Dobi-SVD‡ and Dobi-SVD^{‡,q}.

C. Impact of number of calibration samples

Figure 5 shows the effect of calibration set size on WikiText2 perplexity (left), C4 perplexity (middle), and average zero-shot accuracy (right) at ratios 0.8 and 0.6 for LLaMA-7B. Perplexity and accuracy exhibit qualitatively different behaviors: perplexity drops sharply with the first $\sim$64 samples and largely saturates thereafter at both ratios, indicating that a small calibration set is sufficient for good language modeling fidelity. Accuracy, by contrast, continues to improve substantially beyond 64 samples—particularly at ratio 0.8, where accuracy rises steeply between 64 and 128 samples before gradually plateauing—suggesting that downstream task performance is more sensitive to calibration budget. Our default of 256 samples strikes a practical balance: perplexity has saturated and accuracy is near its plateau, while the calibration cost remains modest.

D. Impact of linear layer compression objective and refinement

Table 3 isolates the contributions of the layer-wise objective and the block-level refinement step across four objective variants and two compression ratios on LLaMA-7B. Without refinement, the input-agnostic objective ($\|\mathbf{W} - \mathbf{W}'\|_F^2$) is completely degenerate at both ratios, confirming that minimizing weight distance in isolation is insufficient for preserving model behavior. The input-aware ($\|\mathbf{WX} - \mathbf{W}'\mathbf{X}\|_F^2$) and the adaptive and anchored ($\|\mathbf{WX} - \mathbf{W}'\mathbf{X}'\|_F^2$) objectives both recover reasonable performance without refinement, with the adaptive and anchored objective performing slightly better in that setting. The shift-aware objective ($\|\mathbf{WX}' - \mathbf{W}'\mathbf{X}'\|_F^2$) performs comparably to the input-aware variant without refinement but does not surpass it after refinement, suggesting that anchoring the reference at the compressed input does not

[2]Under the standard formula, $\rho \leq 1$ restricts $k \leq mn/(m+n)$, precluding high-rank approximations.

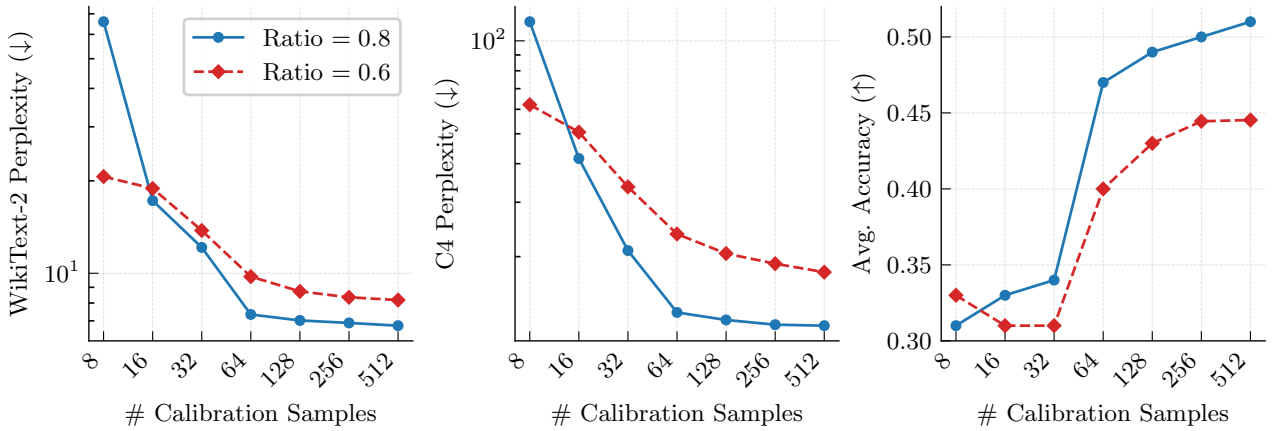


Figure 5. Impact of calibration set size on compression performance. Performance is measured by perplexity on WikiText2 (left) and C4 (middle), and average accuracy across seven zero-shot reasoning tasks (right).

Table 3. WikiText2 Perplexity and average accuracy over seven commonsense reasoning tasks for AA-SVD applied to Llama-7B with different layer-wise objectives and block-level refinement. Best performance is marked in bold.

Ratio	Objective	Refinement	PPL (↓)	Acc. (↑)
1.0	Dense	–	5.68	0.55
0.8	$\|\mathbf{W}-\mathbf{W}'\|_F^2$	✗	$2e4$	0.31
		✓	7.35	0.50
	$\|\mathbf{WX}-\mathbf{W}'\mathbf{X}\|_F^2$	✗	7.89	0.45
		✓	**6.89**	**0.50**
	$\|\mathbf{WX}'-\mathbf{W}'\mathbf{X}'\|_F^2$	✗	8.22	0.45
		✓	7.28	0.45
	$\|\mathbf{WX}-\mathbf{W}'\mathbf{X}'\|_F^2$	✗	**7.68**	**0.46**
		✓	7.08	0.48
0.6	$\|\mathbf{W}-\mathbf{W}'\|_F^2$	✗	$5e5$	0.30
		✓	10.93	**0.45**
	$\|\mathbf{WX}-\mathbf{W}'\mathbf{X}\|_F^2$	✗	13.11	0.37
		✓	**8.35**	0.44
	$\|\mathbf{WX}'-\mathbf{W}'\mathbf{X}'\|_F^2$	✗	14.87	0.36
		✓	8.54	0.44
	$\|\mathbf{WX}-\mathbf{W}'\mathbf{X}'\|_F^2$	✗	**12.19**	**0.38**
		✓	8.52	0.43

provide additional benefit once block-level optimization is applied. Refinement consistently and substantially improves all objectives: most strikingly, it rescues the input-agnostic objective from degeneracy and lifts the input-aware objective to the best overall performance at both ratios—6.89 PPL and 0.50 accuracy at ratio 0.8, and 8.35 PPL at ratio 0.6—establishing that block-level joint optimization is the dominant factor driving final quality. These results indicate that the choice of layer-wise objective matters both with and without refinement: without refinement, it directly determines compression quality, while with refinement it serves as the initialization for block-level refinement, and the final performance remains sensitive to this initialization. Accordingly, our method pairs the input-aware objective with block-level refinement.

E. Comparison with pruning methods

Table 4 compares zero-shot accuracy on LLaMA-2-7B against four structured pruning methods—LLM-Pruner (Ma et al., 2023), SliceGPT (Ashkboos et al., 2024), Bonsai (Kolawole et al., 2024), and Wanda-sp (Sun et al., 2023)—at ratios 0.6 and 0.5, and Table 5 reports WikiText2 perplexity on LLaMA-7B under fixed GPU memory budgets and compares AA-SVD against LLM-Pruner, SliceGPT and BlockPruner (Zhong et al., 2025). Together, they situate AA-SVD relative to methods that remove entire model components and therefore benefit from dense-kernel efficiency at inference time. Without remapping (Appendix B.5), AA-SVD is competitive with the best pruning methods at ratio 0.6 (only a 19.8% accuracy drop vs. 18.3% for Bonsai), a notable result given that SVD-LLM lags substantially behind all pruning baselines at the same ratio (37.5% drop); with remapping, AA-SVD^q surpasses every pruning method by a clear margin, achieving a 7.1% accuracy drop at ratio 0.6—less than half that of Bonsai—and 20.7% at ratio 0.4, competitive with Bonsai's performance at the less aggressive setting. The memory-budget comparison tells a similar story: AA-SVD achieves the lowest perplexity at every budget from 10GB down to 7GB, and the advantage over pruning methods grows as the budget tightens, with structured pruning baselines deteriorating far more sharply under stricter constraints.

Table 4. Comparison of AA-SVD with structured pruning methods on compression performance of LLaMA-2-7B across five common-sense reasoning benchmarks (zero-shot evaluation). Results for baseline methods are taken from Wang et al. (2025a).

Ratio	Method	Accuracy (↑)						
		PIQA	HellaS.	WinoG.	ARC_e	ARC_c	Avg.	Drop (%)
1.0	Dense	0.78	0.57	0.69	0.76	0.43	0.65	–
0.6	LLM-Pruner	0.70	0.41	0.53	0.53	0.27	0.48	24.5%
	SliceGPT	0.65	0.57	0.60	0.43	0.32	0.51	20.4%
	Bonsai	0.72	0.45	0.58	0.59	0.30	0.53	18.3%
	Wanda-sp	0.70	0.42	0.53	0.57	0.29	0.50	22.3%
	SVD-LLM	0.58	0.31	0.53	0.39	0.21	0.40	37.5%
	AA-SVD	0.66	0.41	0.62	0.60	0.30	0.52	19.8%
	Dobi-SVD^{‡,q}	0.72	0.45	0.64	0.67	0.31	0.56	13.6%
	AA-SVD^q	0.73	0.50	0.66	0.72	0.39	0.60	7.1%
0.5	LLM-Pruner	0.67	0.35	0.52	0.48	0.22	0.45	30.7%
	SliceGPT	0.58	0.46	0.55	0.37	0.28	0.45	30.7%
	Bonsai	0.66	0.40	0.54	0.49	0.26	0.47	27.2%
	Wanda-sp	0.63	0.32	0.53	0.43	0.20	0.42	34.7%
0.4	SVD-LLM	0.53	0.27	0.49	0.27	0.22	0.36	44.9%
	AA-SVD	0.60	0.32	0.56	0.44	0.24	0.43	33.1%
	Dobi-SVD^{‡,q}	0.67	0.38	0.57	0.55	0.26	0.49	24.8%
	AA-SVD^q	0.65	0.40	0.61	0.60	0.30	0.51	20.7%

Table 5. Perplexity (WikiText2, ↓) comparison of AA-SVD and structured pruning baselines on LLaMA-7B under different memory budgets. Results for baseline methods are taken from Hu et al. (2026).

Memory	LLM-Pruner	SliceGPT	BlockPruner	SAES-SVD	AA-SVD (Ours)
10GB	9.88	8.78	9.40	7.17	**6.89**
9GB	12.21	12.73	12.76	8.22	**7.14**
8GB	18.94	16.39	19.78	8.96	**7.84**
7GB	21.68	27.41	43.05	10.15	**8.35**

F. Pseudo-code for AA-SVD

The pseudo-code for the main components of AA-SVD is provided in Algorithms 1 and 2.

Algorithm 1 COMPRESSLAYER: AA-SVD layer-wise low-rank compression

Require: Weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$, original inputs $\mathbf{X} \in \mathbb{R}^{n\times l}$, shifted inputs $\mathbf{X}' \in \mathbb{R}^{n\times l}$, target rank k

Set $\mathbf{A} = \mathbf{X}$, $\mathbf{B} = \mathbf{X}'$ {shift-aware: $\mathbf{A}=\mathbf{B}=\mathbf{X}'$; input-aware: $\mathbf{A}=\mathbf{B}=\mathbf{X}$}
Compute $\mathbf{C} = \mathbf{A}\mathbf{B}^\top$ and $\mathbf{S} = \mathbf{B}\mathbf{B}^\top$
Factorize: $\mathbf{S} = \mathbf{R}\mathbf{R}^\top$ {e.g. Cholesky or EVD}
Compute $\mathbf{M} = \mathbf{W}\mathbf{C}\mathbf{S}^{-1}\mathbf{R}$
Truncated SVD: $[\mathbf{U}_k, \mathbf{\Sigma}_k, \mathbf{V}_k] = \mathrm{SVD}_k(\mathbf{M})$
return factorized weight $\mathbf{U} = \mathbf{U}_k\mathbf{\Sigma}_k,\ \mathbf{V} = \mathbf{R}^{-\top}\mathbf{V}_k$, so that $\mathbf{W}' = \mathbf{U}\mathbf{V}^\top$

Algorithm 2 AA-SVD: end-to-end block-wise compression with local refinement

Require: Model $\mathcal{M}$ with blocks $\{\mathcal{L}_i\}_{i=1}^{B}$, calibration data, target rank k

Extract input activations $\mathbf{X} \leftarrow \mathbf{X}' \leftarrow \mathcal{E}(\text{calibration data})$ from the embedding layer $\mathcal{E}$
for each block $\mathcal{L}_i$ in $\mathcal{M}$ **do**
  Initialize compressed block $\mathcal{L}_i' \leftarrow \mathcal{L}_i$
  for each linear layer $\mathbf{W}_j$ in $\mathcal{L}_i'$ **do**
    Collect $\mathbf{X}_j$ from $\mathcal{L}_i$ and $\mathbf{X}_j'$ from $\mathcal{L}_i'$ by forward pass up to layer j
    $[\mathbf{U}_j, \mathbf{V}_j] \leftarrow$ COMPRESSLAYER$(\mathbf{W}_j,\ \mathbf{X}_j,\ \mathbf{X}_j',\ k)$
    Update $\mathbf{W}_j \leftarrow \mathbf{U}_j\mathbf{V}_j^\top$ in $\mathcal{L}_i'$
  end for
  Block-level refinement: optimize $\{\mathbf{U}_j, \mathbf{V}_j\}$ and block-local parameters $\boldsymbol{\theta}_i$ (e.g. norms, biases) jointly to minimize $\mathrm{MSE}(\mathcal{L}_i(\mathbf{X}),\ \mathcal{L}_i'(\mathbf{X}'))$
  Update inputs for next block: $\mathbf{X} \leftarrow \mathcal{L}_i(\mathbf{X}), \quad \mathbf{X}' \leftarrow \mathcal{L}_i'(\mathbf{X}')$
end for
return compressed model $\mathcal{M}'$ with blocks $\{\mathcal{L}_i'\}_{i=1}^{B}$

G. Additional Results and Analysis

Table 6 summarizes AA-SVD against SVD-LLM across five models and two compression ratios; full per-benchmark breakdowns follow in Tables 7–11.

Table 6. Comparison of AA-SVD with SOTA methods across multiple models at compression ratios 0.8 and 0.6. PPL refers to WikiText2 perplexity; Accuracy is averaged over seven commonsense reasoning benchmarks (zero-shot). Best performance is marked in bold.

Ratio	Method	LLaMA-2-7B		LLaMA-2-13B		LLaMA-3-1B		LLaMA-3-8B		Qwen-2.5-7B	
		PPL (↓)	Acc. (↑)	PPL (↓)	Acc. (↑)	PPL (↓)	Acc. (↑)	PPL (↓)	Acc. (↑)	PPL (↓)	Acc. (↑)
1.0	Baseline	5.47	0.55	4.88	0.58	9.75	0.48	6.24	0.60	6.84	0.60
0.8	SVD-LLM	8.41	0.43	6.65	0.48	45.62	0.32	14.16	0.44	10.69	0.47
	AA-SVD	**6.84**	**0.50**	**5.95**	**0.53**	**15.12**	**0.39**	**9.58**	**0.50**	**8.53**	**0.53**
0.6	SVD-LLM	16.47	0.35	10.79	0.38	402.76	0.30	76.31	0.32	28.67	0.33
	AA-SVD	**8.55**	**0.44**	**7.44**	**0.46**	**23.74**	**0.35**	**13.66**	**0.41**	**11.00**	**0.44**

Tables 7–11 provide full per-benchmark breakdowns for the five additional models summarized in Table 6 of the main text. The results consistently replicate the trends observed on LLaMA-7B, confirming that the gains from block-level joint optimization generalize across model families (LLaMA-2, LLaMA-3, Qwen-2.5) and scales (1B–13B parameters). SVD-LLM results are reproduced by us. For other baselines, numbers are taken from their respective papers where available for the given model and compression ratio; entries are left blank where results were not reported.

Table 7. Comparison of AA-SVD with SOTA methods for SVD-based compression of LLaMA-3-1B on three language modeling tasks and seven commonsense reasoning benchmarks (zero-shot evaluation). Best performance is marked in bold.

Ratio	Method	PPL (↓)			Accuracy (↑)								
		Wiki2	PTB	C4	Openb.	ARC_e	ARC_c	WinoG.	PIQA	MathQA	HellaS.	Avg.	Drop (%)
1.0	Dense	9.75	15.40	13.82	0.26	0.65	0.31	0.61	0.74	0.29	0.48	0.48	–
0.8	SVD-LLM	45.62	158.15	206.18	0.14	0.37	0.19	0.51	0.56	0.21	0.28	0.32	32.3%
	AA-SVD	**15.12**	**36.81**	**37.54**	**0.20**	**0.51**	**0.23**	**0.55**	**0.64**	**0.23**	**0.36**	**0.39**	**18.6%**
0.6	SVD-LLM	402.76	2027.07	1449.82	0.12	0.27	0.20	0.51	0.52	0.20	0.26	0.30	37.7%
	AA-SVD	**23.74**	**72.00**	**91.02**	**0.19**	**0.42**	**0.22**	**0.53**	**0.58**	**0.23**	**0.30**	**0.35**	**26.1%**
0.4	SVD-LLM	1369.77	5082.80	3520.70	0.13	0.26	**0.21**	0.51	0.53	0.20	0.26	0.30	37.1%
	AA-SVD	**51.01**	**192.65**	**246.63**	**0.16**	**0.35**	0.20	**0.52**	**0.56**	**0.21**	**0.27**	**0.32**	**32.1%**

Table 8. Comparison of AA-SVD with SOTA methods for SVD-based compression of LLaMA-2-7B on two language modeling tasks and seven commonsense reasoning benchmarks (zero-shot evaluation). Best performance is marked in bold. ‡ uses dynamic or non-uniform ratio allocation, and ^{q} applies Dobi-SVD remapping (see Appendix B.5).

Ratio	Method	PPL (↓)		Accuracy (↑)								
		Wiki2	C4	Openb.	ARC_e	ARC_c	WinoG.	PIQA	MathQA	HellaS.	Avg.	Drop (%)
1.0	Dense	5.47	7.28	0.32	0.76	0.43	0.69	0.78	0.28	0.57	0.55	–
0.8	SVD-LLM	8.41	20.34	0.26	0.57	0.26	0.62	0.66	0.24	0.39	0.43	21.7%
	AA-SVD	**6.84**	**13.19**	**0.30**	**0.71**	**0.37**	**0.64**	**0.72**	**0.27**	**0.48**	**0.50**	**8.9%**
	Dobi-SVD^{‡,q}	5.92	–	–	–	–	–	–	–	–	–	–
	AA-SVD^{q}	5.92	8.41	0.31	0.74	0.42	0.69	0.77	0.29	0.55	0.54	**1.6%**
0.6	SVD-LLM	16.47	73.12	0.18	0.39	0.21	0.53	0.58	0.22	0.31	0.35	36.8%
	SAES-SVD	11.35	40.57	–	0.43	–	0.58	0.59	–	0.32	–	–
	AA-SVD	**8.55**	**21.78**	**0.27**	**0.60**	**0.30**	**0.62**	**0.66**	**0.25**	**0.41**	**0.44**	**18.8%**
	Dobi-SVD^{‡,q}	7.88	–	–	0.67	0.31	0.64	0.72	–	0.45	–	–
	AA-SVD^{q}	6.77	11.64	0.29	0.72	0.39	0.66	0.73	0.28	0.50	0.51	**6.8%**
0.4	SVD-LLM	97.43	615.24	0.13	0.27	0.22	0.49	0.53	0.20	0.27	0.30	44.9%
	SAES-SVD	23.89	100.42	–	0.31	–	0.52	0.55	–	0.30	–	–
	AA-SVD	**14.58**	**53.22**	**0.20**	**0.44**	**0.24**	**0.56**	**0.60**	**0.24**	**0.32**	**0.37**	**32.1%**
	Dobi-SVD^{‡,q}	9.47	–	–	0.55	0.26	0.57	0.67	–	0.38	–	–
	AA-SVD^{q}	8.86	22.48	0.26	0.60	0.30	0.61	0.65	0.25	0.40	0.44	**19.8%**

Table 9. Comparison of AA-SVD with SOTA methods for SVD-based compression of LLaMA-3-8B on three language modeling tasks and seven commonsense reasoning benchmarks (zero-shot evaluation). Best performance is marked in bold.

Ratio	Method	PPL (↓)			Accuracy (↑)								
		Wiki2	PTB	C4	Openb.	ARC_e	ARC_c	WinoG.	PIQA	MathQA	HellaS.	Avg.	Drop (%)
1.0	Dense	6.24	9.89	9.57	0.34	0.82	0.52	0.74	0.80	0.40	0.60	0.60	–
0.8	SVD-LLM	14.16	64.01	79.14	0.24	0.59	0.30	0.64	0.66	0.26	0.37	0.44	27.5%
	SAES-SVD	11.49	–	–	0.25	0.59	0.28	0.66	0.67	0.27	0.39	0.44	26.3%
	AA-SVD	**9.58**	**28.11**	**33.12**	**0.26**	**0.70**	**0.37**	**0.69**	**0.72**	**0.30**	**0.46**	**0.50**	**17.1%**
0.6	SVD-LLM	76.31	971.56	662.65	0.14	0.32	0.19	0.52	0.55	0.21	0.28	0.32	47.6%
	SAES-SVD	23.30	–	–	0.16	0.34	0.20	0.55	0.55	0.22	0.30	0.33	45.0%
	AA-SVD	**13.66**	**56.33**	**74.48**	**0.23**	**0.54**	**0.27**	**0.61**	**0.64**	**0.24**	**0.37**	**0.41**	**31.3%**
0.4	SVD-LLM	649.12	8403.95	3375.48	0.12	0.27	0.20	0.51	0.52	0.20	0.26	0.30	50.7%
	SAES-SVD	63.09	–	–	0.13	0.29	**0.22**	0.53	0.54	0.23	0.28	0.32	47.4%
	AA-SVD	**32.23**	**263.02**	**323.43**	**0.18**	**0.38**	0.20	**0.52**	**0.58**	**0.22**	**0.30**	**0.34**	**43.6%**

Table 10. Comparison of AA-SVD with SOTA methods for SVD-based compression of Qwen-2.5-7B on three language modeling tasks and seven commonsense reasoning benchmarks (zero-shot evaluation). Best performance is marked in bold.

Ratio	Method	PPL (↓)			Accuracy (↑)								
		Wiki2	PTB	C4	Openb.	ARC_e	ARC_c	WinoG.	PIQA	MathQA	HellaS.	Avg.	Drop (%)
1.0	Dense	6.84	11.37	11.85	0.34	0.80	0.50	0.73	0.79	0.43	0.60	0.60	–
0.8	SVD-LLM	10.69	39.10	38.53	0.25	0.67	0.31	0.64	0.68	0.32	0.41	0.47	21.7%
	AA-SVD	**8.53**	**22.90**	**22.05**	**0.31**	**0.74**	**0.41**	**0.69**	**0.73**	**0.37**	**0.49**	**0.53**	**10.7%**
0.6	SVD-LLM	28.67	193.31	161.22	0.15	0.36	0.20	0.53	0.56	0.22	0.29	0.33	44.9%
	AA-SVD	**11.00**	**49.10**	**40.85**	**0.25**	**0.59**	**0.28**	**0.61**	**0.65**	**0.28**	**0.39**	**0.44**	**27.2%**
0.4	SVD-LLM	136.74	963.37	647.59	0.12	0.28	0.20	0.49	0.54	0.21	0.27	0.30	49.6%
	AA-SVD	**15.67**	**86.23**	**62.81**	**0.20**	**0.44**	**0.23**	**0.57**	**0.60**	**0.23**	**0.33**	**0.37**	**37.9%**

Table 11. Comparison of AA-SVD with SOTA methods for SVD-based compression of LLaMA-2-13B on three language modeling tasks and seven commonsense reasoning benchmarks (zero-shot evaluation).

Ratio	Method	PPL (↓)			Accuracy (↑)								
		Wiki2	PTB	C4	Openb.	ARC_e	ARC_c	WinoG.	PIQA	MathQA	HellaS.	Avg.	Drop (%)
1.0	Dense	4.88	34.40	6.73	0.35	0.79	0.48	0.72	0.79	0.32	0.60	0.58	–
0.8	SVD-LLM	6.65	84.17	14.99	0.29	0.67	0.33	0.68	0.71	0.26	0.44	0.48	16.5%
	AA-SVD	**5.95**	**46.43**	**11.6**	**0.33**	**0.73**	**0.40**	**0.69**	**0.74**	**0.29**	**0.52**	**0.53**	**8.9%**
0.6	SVD-LLM	10.79	243.85	46.47	0.22	0.47	0.22	0.61	0.60	0.23	0.33	0.38	33.8%
	AA-SVD	**7.44**	**79.01**	**19.32**	**0.25**	**0.64**	**0.32**	**0.63**	**0.68**	**0.26**	**0.41**	**0.46**	**21.2%**
0.4	SVD-LLM	44.28	1296.01	295.21	0.14	0.31	0.20	0.52	0.54	0.21	0.27	0.31	45.9%
	AA-SVD	**11.77**	**154.96**	**42.50**	**0.23**	**0.45**	**0.24**	**0.58**	**0.60**	**0.24**	**0.35**	**0.38**	**33.6%**