AA-SVD: Anchored and Adaptive SVD for Large Language Model Compression

Anonymous authors
Paper under double-blind review

Abstract

We introduce a fast low-rank factorization-based framework for compressing large language models that enables rapid compression of billion-parameter models without retraining. Unlike existing factorization-based approaches that optimize only on the original inputs, ignoring distribution shifts from upstream compression and thus propagating errors forward, or those that rely only on shifted inputs and risk drifting away from the original outputs, our approach accounts for both. Beyond individual layer compression, we further refine each transformer block end-to-end, minimizing block-level output distortion and allowing compressed layers to jointly compensate for accumulated errors. By anchoring each compressed layer to the original outputs while explicitly modeling input distribution shifts, our method finds a low-rank approximation that maintains functional equivalence with the original model. Experiments on large language models show that our method consistently outperforms existing SVD-based baselines across compression ratios, with the advantage becoming increasingly pronounced at aggressive compression budgets, where competing methods degrade substantially or collapse entirely, offering a practical solution for efficient, large-scale model deployment.[1]

1 Introduction

The rapid progress of large-scale pretrained models has fundamentally transformed natural language processing. Modern large language models (LLMs) (Brown et al., 2020; Touvron et al., 2023; Zhang et al., 2022; Achiam et al., 2023) now routinely contain tens to hundreds of billions of parameters, enabling remarkable generalization across a wide range of downstream tasks (Kaplan et al., 2020). However, this scaling has come at steep computational cost: training, fine-tuning, and inference with such models require clusters of high-memory GPUs, making them prohibitively expensive to deploy in resource-constrained or latency-sensitive settings (Patterson et al., 2021).

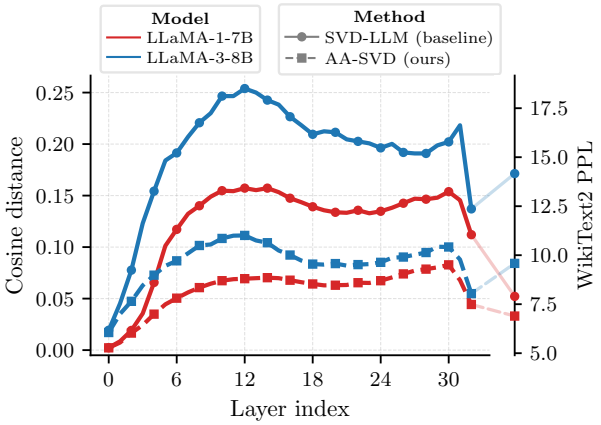


Figure 1: Distortion between *intermediate* features of the original and compressed model, measured as avg. cosine distance on 64 WikiText2 test samples. Diagonal lines link each method's final-layer distortion to its WikiText2 perplexity (right axis). AA-SVD suppresses compression error consistently across depth.

One promising direction is to move beyond ever-larger models toward smaller, more efficient ones. Compact models can be trained from scratch for specialized tasks, but this approach sacrifices the broad generalization ability of large pretrained networks. Alternatively, smaller models can be obtained by *distilling* large networks into student models trained to mimic their behavior (Hinton et al., 2015; Xu et al.,

[1]Code to reproduce all experiments will be released upon acceptance.

2024), or by applying *post-training compression* techniques such as pruning, quantization, or low-rank factorization (Cheng et al., 2017; Zhu et al., 2024). While both approaches reduce memory footprint and inference cost, distillation typically requires substantial retraining data and compute (Hinton et al., 2015; Jiao et al., 2020; Sanh et al., 2019), whereas post-training compression can often be applied more rapidly to pretrained networks (Frantar et al., 2022; Dettmers et al., 2022; Wang et al., 2025c), thereby offering a practical path towards democratizing deployment.

A wide range of model compression techniques have been proposed: *Pruning* removes redundant weights or structures from neural networks, with early work on unstructured sparsification (Han et al., 2015) and the lottery ticket hypothesis (Frankle & Carbin, 2019) showing that smaller subnetworks can be retrained to match dense counterparts. While effective, pruning often requires iterative retraining and specialized sparsity-aware hardware to fully realize efficiency gains, though recent advances such as SparseGPT and its variants (Frantar & Alistarh, 2023; Ma et al., 2023; Ashkboos et al., 2024; An et al., 2024) have enabled post-training pruning of large language models. *Quantization* reduces numerical precision of weights and activations; modern methods like LLM.int8() (Dettmers et al., 2022), QLoRA (Dettmers et al., 2023), and AWQ (Lin et al., 2024) allow near-lossless compression of transformers, though very low-bit settings may require careful calibration. Another line of work leverages the inherent low-rank structure of network weights: *low-rank factorization* decomposes large matrices into compact representations, reducing both parameters and computation. Early applications in CNNs (Denton et al., 2014; Tai et al., 2015) demonstrated significant speedups, but naïve SVD truncation is known to degrade accuracy. More recent activation-aware approaches for LLMs (Yuan et al., 2023; Wang et al., 2025c; Li et al., 2025; Wang et al., 2025a) explicitly account for input activations, mitigating this limitation at the cost of additional computation.

These methods differ in their retraining requirements, their dependence on large datasets versus small calibration samples, the efficiency with which compression can be applied to pretrained networks, the degree to which downstream accuracy is preserved, and the extent to which the resulting compressed structure aligns with modern accelerators (Cheng et al., 2018). *SVD-based methods* are especially appealing: they exploit the inherent low-rank structure of neural network weights, yielding compressed models without the need for expensive retraining (Denton et al., 2014; Jaderberg et al., 2014). A straightforward approach is to directly truncate weight matrices by retaining only the top singular components, but this often leads to severe degradation because it treats all input directions equally and discards information that is important for the actual distribution of activations (Denil et al., 2013; Chen et al., 2021; Wang et al., 2025c). This limitation has been repeatedly observed in large-scale networks, where naïve low-rank truncation fails to preserve task accuracy and generalization. To address this, activation-aware approaches have been developed that tailor the factorization to the input distribution, thereby retaining the directions most relevant to the network's operation. However, existing activation-aware SVD methods often optimize low-rank approximations using only the original input distribution (Yuan et al., 2023; Wang et al., 2025c; Li et al., 2025; Wang et al., 2025a), ignoring the shift introduced by upstream compression, which can propagate errors and degrade downstream performance. Conversely, methods that rely exclusively on shifted inputs, such as Dobi-SVD (Wang et al., 2025a), risk deviating from the original network behavior, introducing instability and loss of fidelity.

In this work, we present AA-SVD, a *fast low-rank factorization-based framework* for compressing pretrained networks. Our approach accounts for both the original outputs and the distribution shifts caused by upstream compression. This design yields compressed layers that more faithfully preserve the functional behavior of the uncompressed model, enabling post-training compression of billion-parameter networks without retraining. Additionally, AA-SVD refines all compressed layers within a block jointly, minimizing the block-output error and allowing layers to compensate for each other's residual errors. Figure 1 illustrates how AA-SVD suppresses compression error consistently across depth compared to prior methods.

2 Related Work

Low-rank factorization, e.g., via singular value decomposition (SVD), has emerged as a promising direction for compressing large pretrained models. Unlike pruning (irregular sparsity) or quantization (specialized kernels), factorization yields dense, structured factors—enabling the commutation $(\boldsymbol{U}\boldsymbol{V}^\top)\boldsymbol{X} = \boldsymbol{U}(\boldsymbol{V}^\top\boldsymbol{X})$—that integrate seamlessly with standard linear algebra libraries and reduce both parameters and FLOPs. Cru-

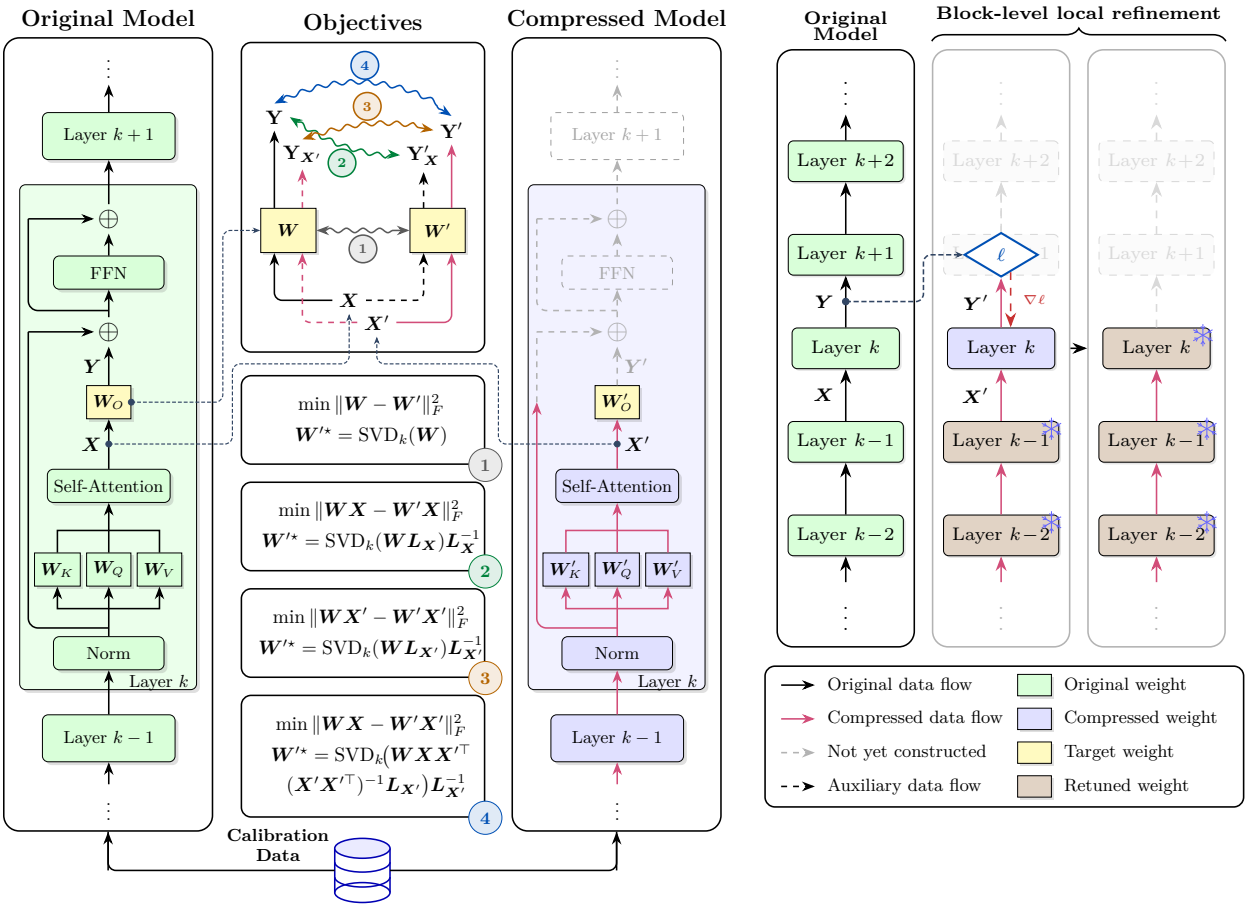


Figure 2: Overview of the two-stage compression pipeline. **Left:** Four layer-wise compression objectives, differing in which inputs and outputs are compared. (1) *Input-agnostic*: $\|\boldsymbol{W} - \boldsymbol{W}'\|_F^2$ — ignores activations entirely. (2) *Input-aware*: $\|\boldsymbol{W}\boldsymbol{X} - \boldsymbol{W}'\boldsymbol{X}\|_F^2$ — matches outputs on original inputs $\boldsymbol{X}$. (3) *Shift-aware*: $\|\boldsymbol{W}\boldsymbol{X}' - \boldsymbol{W}'\boldsymbol{X}'\|_F^2$ — matches outputs on the shifted inputs $\boldsymbol{X}'$ seen after upstream compression. (4) *Anchored adaptive* (ours): $\|\boldsymbol{W}\boldsymbol{X} - \boldsymbol{W}'\boldsymbol{X}'\|_F^2$ — anchors the target to the original output while conditioning on the shifted input, combining an uncorrupted reference with distribution-shift awareness. **Right:** Block-level local refinement. Stage 1 factorizes all linear layers in the block independently via any layer-wise objective. Stage 2 then jointly optimizes all factorized weights to minimize the block-output error $\ell = \|\mathcal{L}(\boldsymbol{X}) - \mathcal{L}'(\boldsymbol{X}')\|_F^2$, keeping upstream blocks frozen — the same anchored adaptive spirit as (4) but applied at block granularity. This lets the compressed layers within a block compensate for each other's residual errors, substantially recovering block-output fidelity.

cially, they can be applied post-training with only a small[2] calibration set, making them attractive when retraining is infeasible. Recent methods such as ASVD (Yuan et al., 2023), SVD-LLM (Wang et al., 2025c), AdaSVD (Li et al., 2025), SVD-LLM V2 (Wang et al., 2025b), Dobi-SVD (Wang et al., 2025a), DipSVD (Ding et al., 2025), and SAES-SVD (Hu et al., 2026) have demonstrated the viability of this approach at scale in large language models. Based on the optimization objective, compression methods can be broadly grouped into the following categories (Figure 2 (left) gives a visual overview):

Input-agnostic compression. The simplest approach compresses a sub-module f without reference to its inputs, optimizing over the module's parameters alone: $\min_{f' \in \mathcal{F}} \; d\big(f, f'\big)$, where $\mathcal{F}$ denotes the family of admissible compressed sub-modules (e.g., rank-constrained matrices for factorization, sparse masks for pruning), and d is a distance defined purely on the parameters of f and f', with no dependence on any input data. For pruning, it corresponds to magnitude-based removal of weights or neurons (Han et al., 2015); and

[2]usually 64-1024 samples

in quantization, to rounding without calibration. With low-rank factorization for a linear layer $f(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}$, this takes the form

$$\min_{\boldsymbol{W}':\mathrm{rank}(\boldsymbol{W}')=k} \|\boldsymbol{W} - \boldsymbol{W}'\|_F,$$

which is solved in closed form by the truncated SVD of $\boldsymbol{W}$ using the Eckart–Young theorem, replacing it by a rank-k approximation $\boldsymbol{W}'$ constructed from its top singular components (Halko et al., 2011; Sainath et al., 2013). These methods require no data and are fully order-independent: each sub-module is compressed in isolation with no coupling to the others. However, they treat all parameter directions uniformly, ignoring the fact that in deep networks the actual input activations lie in a highly anisotropic subspace (Ortiz-Jiménez et al., 2020): directions preserved by parameter-space approximations may not align with those that matter for downstream performance (Chen et al., 2021; Idelbayev & Carreira-Perpinán, 2020).

Input-aware compression. A natural refinement is to account for the geometry of the intermediate features that the sub-module encounters during inference: $\min_{f'\in\mathcal{F}} \; d\big(f(\boldsymbol{X}),\, f'(\boldsymbol{X})\big)$, where $\boldsymbol{X} \in \mathbb{R}^{n\times l}$ collects intermediate activations at the input of f from the *original, uncompressed* network on calibration samples, and d measures the output discrepancy. For a linear layer $f(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}$, taking d to be the squared Frobenius norm specializes this to

$$\min_{\boldsymbol{W}':\mathrm{rank}(\boldsymbol{W}')=k} \|\boldsymbol{W}\boldsymbol{X} - \boldsymbol{W}'\boldsymbol{X}\|_F^2,$$

a formulation adopted in DRONE (Chen et al., 2021), ASVD, SVD-LLM, AdaSVD, SVD-LLM V2 and DipSVD. Pruning methods like FLAP (An et al., 2024) use a related objective, leveraging activation statistics from the original network to guide structured pruning decisions. By preserving the action of $\boldsymbol{W}$ on the occupied feature subspace, this is far more faithful to downstream behavior than the input-agnostic objective, and because $\boldsymbol{X}$ is fixed, sub-module objectives are fully decoupled and can be compressed in any order. However, as layers are compressed sequentially, the actual inputs received by each sub-module increasingly diverge from $\boldsymbol{X}$ — and since input-aware methods do not account for this error accumulation, the compressed model's behavior can diverge substantially from the original.

Shift-aware compression. A key limitation of input-aware methods is that $\boldsymbol{X}$ is produced by the *original* network, whereas in a sequentially compressed pipeline the sub-module actually receives different intermediate features — those produced by the upstream compressed layers. Shift-aware methods address this by instead minimizing $\min_{f'\in\mathcal{F}} \; d\big(f(\boldsymbol{X}'),\, f'(\boldsymbol{X}')\big)$, where $\boldsymbol{X}' \in \mathbb{R}^{n\times l}$ collects the intermediate features at the input of f, produced by running the *partially compressed* network on the same calibration samples, and d measures the output discrepancy on those shifted features. For a linear layer $f(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}$, taking d to be the squared Frobenius norm gives

$$\min_{\boldsymbol{W}':\mathrm{rank}(\boldsymbol{W}')=k} \|\boldsymbol{W}\boldsymbol{X}' - \boldsymbol{W}'\boldsymbol{X}'\|_F^2,$$

as adopted in Dobi-SVD (Wang et al., 2025a), with related ideas in earlier CNN methods (Denton et al., 2014; Jaderberg et al., 2014) and layer-wise distillation (Jiao et al., 2020). The same principle underlies quantization methods such as GPTQ (Frantar et al., 2022) and pruning methods such as SparseGPT (Frantar & Alistarh, 2023), which process weights in a fixed sequential order, conditioning each update on the outputs of already-compressed predecessors. By anchoring to the intermediate features the sub-module truly receives, shift-aware methods can mitigate error propagation through the stack. In stark contrast to input-agnostic and input-aware methods, ordering is not a matter of convenience but a hard requirement: since $\boldsymbol{X}'$ depends on all upstream compressed layers, shift-aware compression must follow a valid topological order—compressing out of order yields features $\boldsymbol{X}'$ inconsistent with any valid partial compression state. Their drawback is that when upstream compression has degraded representations, anchoring solely to $\boldsymbol{X}'$ risks amplifying divergence from the original network's behavior; moreover, $\boldsymbol{X}'$ is estimated from a finite calibration batch and may be noisy, introducing instability. Thus shift-aware objectives alone provide only a partial solution.

Beyond the choice of approximation objective, the effectiveness of low-rank factorization depends critically on how ranks are distributed across layers. Uniform allocation ignores heterogeneity in both compressibility and functional importance. ASVD (Yuan et al., 2023) proposed Sensitivity-based Truncation Rank Searching

(STRS), which evaluates the sensitivity of each linear module to truncation at different rank levels in isolation, measuring sensitivity as the change in perplexity on the calibration dataset; this requires repeated full-model evaluations across modules and rank levels, making it expensive. SVD-LLM V2 (Wang et al., 2025b) takes a different heuristic approach, reallocating rank based on the truncation loss $\|\boldsymbol{W}\boldsymbol{X} - \boldsymbol{W}'\boldsymbol{X}\|_F^2$ observed after an initial uniform compression. 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). More principled methods include analytical formulations (Solgi et al., 2025; Abbasi et al., 2026) and differentiable relaxations (Rausch et al., 2025; Wang et al., 2025a) that optimize rank allocation end-to-end, and learned mask approaches (Gao et al., 2024; Sundrani et al., 2025; Xv et al., 2025) that select singular components via gradient descent.

3 AA-SVD

As established in Section 2, existing SVD-based compression methods fall into three broad categories that each capture only a partial view of the compression problem: input-agnostic methods ignore the input distribution entirely; input-aware methods account for the original activations but are blind to shifts introduced by upstream compression; and shift-aware methods adapt to the modified inputs but risk drifting from the original network's behavior. We present AA-SVD (Anchored and Adaptive SVD), a compression framework that bridges these perspectives. The central insight is that a faithfully compressed layer must simultaneously satisfy two constraints: its outputs should remain close to those of the uncompressed model, and it must operate correctly on the inputs it will actually receive at inference time—which, after upstream layers have been compressed, may differ substantially from the original activations. A second insight is that minimizing the error of each linear layer independently is not sufficient: errors across the multiple linear layers within a transformer block can interact, so that even small per-layer errors compound into a larger distortion at the block output. We therefore introduce a block-level refinement step that minimizes the output error of the entire block after its linear layers have been compressed—allowing the compressed layers to jointly compensate for one another's errors regardless of which layer-wise objective was used.

3.1 Preliminaries

We consider a pretrained model $\mathcal{M}$ comprising a sequence of B transformer blocks $\{\mathcal{L}_i\}_{i=1}^{B}$, applied sequentially. Each block $\mathcal{L}_i$ is composed of multiple linear layers—parameterized by weight matrices—together with non-linear operations such as normalization and activations. Our compression procedure operates at two granularities: at the *linear-layer level*, where each weight matrix $\boldsymbol{W}$ within a block is individually approximated by a low-rank matrix; and at the *block level*, where the compressed linear layers within $\mathcal{L}_i$ are jointly refined.

We collect a calibration set of N samples and, for any component $f \in \{\boldsymbol{W}, \mathcal{L}_i\}$, denote by $\boldsymbol{X}$ the matrix of its input activations on the calibration set (stacked column-wise) and by $f(\boldsymbol{X})$ its corresponding outputs. For a linear layer, $\boldsymbol{X} \in \mathbb{R}^{n\times l}$ and $f(\boldsymbol{X}) = \boldsymbol{W}\boldsymbol{X} \in \mathbb{R}^{m\times l}$; for a block, $\boldsymbol{X} \in \mathbb{R}^{d\times l}$ and $f(\boldsymbol{X}) = \mathcal{L}_i(\boldsymbol{X}) \in \mathbb{R}^{d\times l}$.

When components are compressed sequentially, each receives *shifted* intermediate features produced by upstream compressed components rather than the original network. We denote by $\boldsymbol{X}'$ the corresponding shifted activations—collected by running the same calibration samples through the *partially compressed* network up to (but not including) the current component. For a linear layer, $f' = \boldsymbol{W}'$ is a low-rank matrix with $\mathrm{rank}(\boldsymbol{W}') = k \leq \min(m, n)$, decomposed as $\boldsymbol{W}' = \boldsymbol{U}\boldsymbol{V}^\top$ with $\boldsymbol{U} \in \mathbb{R}^{m\times k}$ and $\boldsymbol{V} \in \mathbb{R}^{n\times k}$. For a block, $f' = \mathcal{L}_i'$ denotes the block with its linear layers replaced by their low-rank approximations and subsequently refined with a block-level objective.

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 $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ with thin SVD $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$. Then*

$$\min_{\mathrm{rank}(\boldsymbol{W}')\leq k} \|\boldsymbol{W} - \boldsymbol{W}'\|_F^2 = \sum_{i>k} \sigma_i(\boldsymbol{W})^2,$$

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

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

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

Suppose $\boldsymbol{B}\boldsymbol{B}^\top$ *is invertible, and let* $\boldsymbol{L}_{\boldsymbol{B}}$ *be any invertible matrix satisfying* $\boldsymbol{B}\boldsymbol{B}^\top = \boldsymbol{L}_{\boldsymbol{B}}\boldsymbol{L}_{\boldsymbol{B}}^\top$[3]*. Then an optimal solution to equation 1 is*

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

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

Proof. See Appendix A. □

Corollary 3.3 (No distribution shift)**.** *If* $\boldsymbol{B} = \boldsymbol{A}$*, then* $\boldsymbol{A}\boldsymbol{B}^\top = \boldsymbol{B}\boldsymbol{B}^\top = \boldsymbol{L}_{\boldsymbol{B}}\boldsymbol{L}_{\boldsymbol{B}}^\top$*, so* $\boldsymbol{M} = \boldsymbol{W}\boldsymbol{L}_{\boldsymbol{B}}^\top$*. The solution reduces to* $\boldsymbol{W}'^{\star} = \mathrm{SVD}_k\big(\boldsymbol{W}\boldsymbol{L}_{\boldsymbol{B}}\big)\,\boldsymbol{L}_{\boldsymbol{B}}^{-1}$*, the standard whitening-based low-rank regression solution.*

3.2 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 $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ with original inputs $\boldsymbol{X} \in \mathbb{R}^{n\times l}$ and shifted inputs $\boldsymbol{X}' \in \mathbb{R}^{n\times l}$ (after upstream compression), we seek a low-rank approximation $\boldsymbol{W}' \in \mathbb{R}^{m\times n}$ that solves

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

This objective enforces that the compressed outputs $\boldsymbol{W}'\boldsymbol{X}'$ stay close to the original outputs $\boldsymbol{W}\boldsymbol{X}$, anchoring the compressed network to the behavior of the uncompressed one while simultaneously adapting to the shifted input distribution. By explicitly constraining $\mathrm{rank}(\boldsymbol{W}') = k$, the problem is well-posed as a low-rank regression: we seek the best rank–k approximation of the mapping from $\boldsymbol{X}'$ to $\boldsymbol{W}\boldsymbol{X}$. 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 $\boldsymbol{X}\boldsymbol{X}'^\top$ and $\boldsymbol{X}'\boldsymbol{X}'^\top$, 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.

3.3 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 4). 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 $\boldsymbol{X}$ and shifted inputs $\boldsymbol{X}'$ (received after upstream blocks are compressed), we minimize

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

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 $\boldsymbol{W}_j$ replaced by its factorized approximation $\boldsymbol{U}_j\boldsymbol{V}_j^\top$, 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,

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

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

Set $\boldsymbol{A} = \boldsymbol{X}$, $\boldsymbol{B} = \boldsymbol{X'}$ {shift-aware: $\boldsymbol{A}=\boldsymbol{B}=\boldsymbol{X'}$; input-aware: $\boldsymbol{A}=\boldsymbol{B}=\boldsymbol{X}$}

Compute $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}^\top$ and $\boldsymbol{S} = \boldsymbol{B}\boldsymbol{B}^\top$

Factorize: $\boldsymbol{S} = \boldsymbol{R}\boldsymbol{R}^\top$ {e.g. Cholesky or EVD}

Compute $\boldsymbol{M} = \boldsymbol{W}\boldsymbol{C}\boldsymbol{S}^{-1}\boldsymbol{R}$

Truncated SVD: $[\boldsymbol{U}_k, \boldsymbol{\Sigma}_k, \boldsymbol{V}_k] = \mathrm{SVD}_k(\boldsymbol{M})$

return factorized weight $\boldsymbol{U} = \boldsymbol{U}_k\boldsymbol{\Sigma}_k$, $\boldsymbol{V} = \boldsymbol{R}^{-\top}\boldsymbol{V}_k$, so that $\boldsymbol{W'} = \boldsymbol{U}\boldsymbol{V}^\top$

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 $\boldsymbol{X} \leftarrow \boldsymbol{X'} \leftarrow \mathcal{E}$(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 $\boldsymbol{W}_j$ in $\mathcal{L}'_i$ **do**

    Collect $\boldsymbol{X}_j$ from $\mathcal{L}_i$ and $\boldsymbol{X}'_j$ from $\mathcal{L}'_i$ by forward pass up to layer j

    $[\boldsymbol{U}_j, \boldsymbol{V}_j] \leftarrow$ CompressLayer($\boldsymbol{W}_j$, $\boldsymbol{X}_j$, $\boldsymbol{X}'_j$, k)

    Update $\boldsymbol{W}_j \leftarrow \boldsymbol{U}_j\boldsymbol{V}_j^\top$ in $\mathcal{L}'_i$

  end for

  Block-level refinement: optimize $\{\boldsymbol{U}_j, \boldsymbol{V}_j\}$ and block-local parameters $\boldsymbol{\theta}_i$ (e.g. norms, biases) jointly to minimize $\mathrm{MSE}(\mathcal{L}_i(\boldsymbol{X}), \mathcal{L}'_i(\boldsymbol{X'}))$

  Update inputs for next block: $\boldsymbol{X} \leftarrow \mathcal{L}_i(\boldsymbol{X})$, $\boldsymbol{X'} \leftarrow \mathcal{L}'_i(\boldsymbol{X'})$

end for

return compressed model $\mathcal{M}'$ with blocks $\{\mathcal{L}'_i\}_{i=1}^{B}$

it adds negligible overhead while substantially recovering block-output fidelity. Figure 2 (right) illustrates the block-level refinement stage.

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 are provided in Appendix B.2.

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 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. Compressed models are then evaluated along two axes: *language modeling perplexity* on WikiText2, C4 (Raffel et al., 2020), and PTB (Marcinkiewicz, 1994), which measures 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 against five SVD-based baselines—ASVD, SVD-LLM, Dobi-SVD, Dip-SVD, and SAES-SVD—across three perplexity benchmarks and seven zero-shot common-

[3]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 on three language modeling tasks and seven commonsense reasoning benchmarks (zero-shot evaluation) under varying compression ratios. Best performance is marked in bold. (†) uses LoRA fine-tuning, (‡) uses dynamic or non-uniform capacity/rank allocation, and (^{q}) indicates results with Dobi-SVD-style remapping enabled. Results for baseline methods are taken from the original papers or prior work 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	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%
	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%
	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%
	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%**
	Dobi-SVD^{‡,q}	6.08	15.39	**7.83**	0.27	0.65	0.37	0.68	0.77	**0.27**	**0.54**	0.51	7.3%
	AA-SVD^{q}	**6.01**	**8.97**	8.37	**0.30**	**0.74**	**0.41**	**0.69**	**0.77**	0.26	0.53	**0.53**	**3.4%**
0.6	ASVD	1407	3292	1109	0.13	0.28	0.22	0.48	0.55	0.19	0.26	0.30	44.9%
	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%
	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%
	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%**
	Dobi-SVD^{‡,q}	8.12	43.85	12.63	0.28	0.65	0.32	0.62	0.72	0.25	0.45	0.47	14.1%
	AA-SVD^{q}	**7.09**	**11.07**	**11.25**	**0.28**	**0.71**	**0.37**	**0.65**	**0.73**	**0.26**	**0.49**	**0.50**	**8.9%**
0.4	ASVD	57057	45218	43036	0.12	0.26	0.21	0.49	0.53	0.18	0.26	0.29	46.5%
	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%
	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%
	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%**
	Dobi-SVD^{‡,q}	9.95	67.62	**17.94**	0.23	0.52	0.24	0.56	**0.65**	0.23	0.38	0.40	26.6%
	AA-SVD^{q}	**8.61**	**24.44**	19.69	**0.26**	**0.58**	**0.31**	**0.62**	0.64	**0.23**	**0.41**	**0.44**	**20.4%**

sense reasoning tasks at compression ratios of 0.8, 0.6, and 0.4. Table 2 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 C. We also include results with Dobi-SVD-style remapping enabled for both Dobi-SVD and AA-SVD for a fair comparison; more details on remapping are provided in Appendix B.4.

At ratio 0.8, AA-SVD achieves the best perplexity and average accuracy among all methods without weight remapping, with the nearest competitor (SAES-SVD) incurring a notably larger accuracy drop; enabling weight remapping (AA-SVD^{q}) further reduces the accuracy gap to only 3.4%, outperforming Dobi-SVD^{‡,q} on both metrics despite Dobi-SVD employing dynamic rank allocation. As compression becomes more aggressive the margin widens: at ratio 0.6, AA-SVD reduces perplexity substantially across all three benchmarks while matching or exceeding SAES-SVD on every reasoning task, and with remapping, out-of-domain perplexity (PTB) improves by a particularly large factor over Dobi-SVD^{‡,q}. At ratio 0.4, ASVD and SVD-LLM become essentially degenerate, while AA-SVD continues to produce functional compressed models, reducing perplexity by nearly 40% relative to SAES-SVD and cutting the accuracy drop by roughly six points.

The gains generalize broadly across architectures (Table 2). 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 degrades by a factor of three—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—underscoring the importance of minimizing block-level output error rather than compressing each layer in isolation.

Table 2: 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**

Table 3: Comparison of AA-SVD with structured pruning methods on compression performance of LLaMA-2-7B across five commonsense 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%

4.2 Comparison with pruning methods

Table 3 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 4 reports WikiText2 perplexity on LLaMA-7B under fixed GPU memory budgets and compares AA-SVD against LLM-Pruner, SliceGPT and Block-Pruner (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, 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.

4.3 Ablations and Analysis

Table 4: 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**

Impact of linear layer compression objective and refinement. Table 5 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 ($\|\boldsymbol{W}-\boldsymbol{W}'\|_F^2$) is completely degenerate at both ratios, confirming that minimizing weight distance in isolation is insufficient for preserving model behavior. The input-aware ($\|\boldsymbol{W}\boldsymbol{X}-\boldsymbol{W}'\boldsymbol{X}\|_F^2$) and the adaptive and anchored ($\|\boldsymbol{W}\boldsymbol{X}-\boldsymbol{W}'\boldsymbol{X}'\|_F^2$) objectives both recover reasonable performance without refinement, with the adaptive and anchored objective performing slightly better in that setting. The shift-aware objective ($\|\boldsymbol{W}\boldsymbol{X}'-\boldsymbol{W}'\boldsymbol{X}'\|_F^2$) 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 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.

Table 5: WikiText2 Perplexity and average accuracy over seven commonsense reasoning tasks for AA-SVD 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	$\|\boldsymbol{W}-\boldsymbol{W}'\|_F^2$	✗	2e4	0.31
		✓	7.35	0.50
	$\|\boldsymbol{W}\boldsymbol{X}-\boldsymbol{W}'\boldsymbol{X}\|_F^2$	✗	7.89	0.45
		✓	**6.89**	**0.50**
	$\|\boldsymbol{W}\boldsymbol{X}'-\boldsymbol{W}'\boldsymbol{X}'\|_F^2$	✗	8.22	0.45
		✓	7.28	0.45
	$\|\boldsymbol{W}\boldsymbol{X}-\boldsymbol{W}'\boldsymbol{X}'\|_F^2$	✗	**7.68**	**0.46**
		✓	7.08	0.48
0.6	$\|\boldsymbol{W}-\boldsymbol{W}'\|_F^2$	✗	5e5	0.30
		✓	10.93	**0.45**
	$\|\boldsymbol{W}\boldsymbol{X}-\boldsymbol{W}'\boldsymbol{X}\|_F^2$	✗	13.11	0.37
		✓	**8.35**	0.44
	$\|\boldsymbol{W}\boldsymbol{X}'-\boldsymbol{W}'\boldsymbol{X}'\|_F^2$	✗	14.87	0.36
		✓	8.54	0.44
	$\|\boldsymbol{W}\boldsymbol{X}-\boldsymbol{W}'\boldsymbol{X}'\|_F^2$	✗	**12.19**	**0.38**
		✓	8.52	0.43

Impact of Number of Calibration Samples. Figure 3 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. Perplexity and accuracy exhibit qualitatively different behaviors: perplexity drops sharply with the first ∼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.

Error Evolution Across Layers. To better understand how compression affects internal representations, we track the MSE and cosine distance between original and compressed outputs across depth, compressing LLaMA-7B at ratio 0.8 and evaluating on WikiText2 test split samples not used for calibration. Figure 4 reports results for attention output projections, MLP-down projections, and full block outputs, comparing

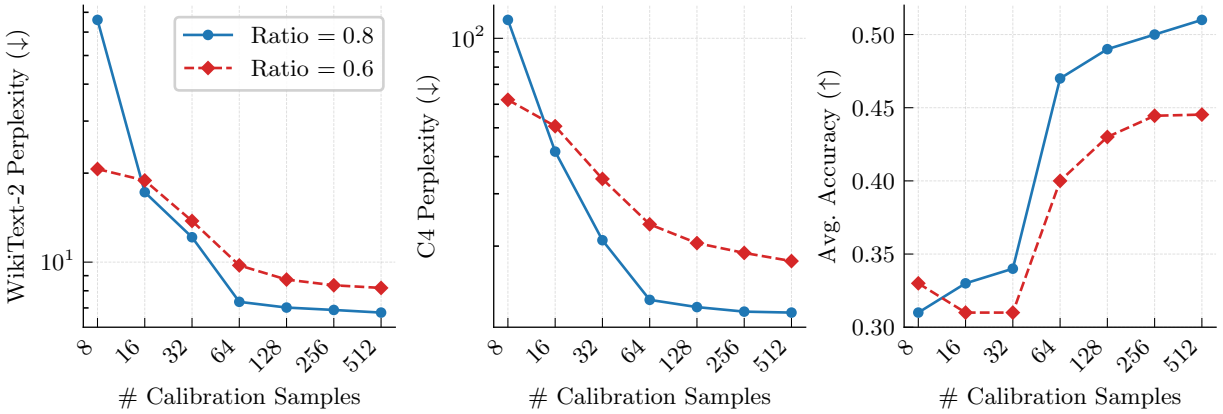


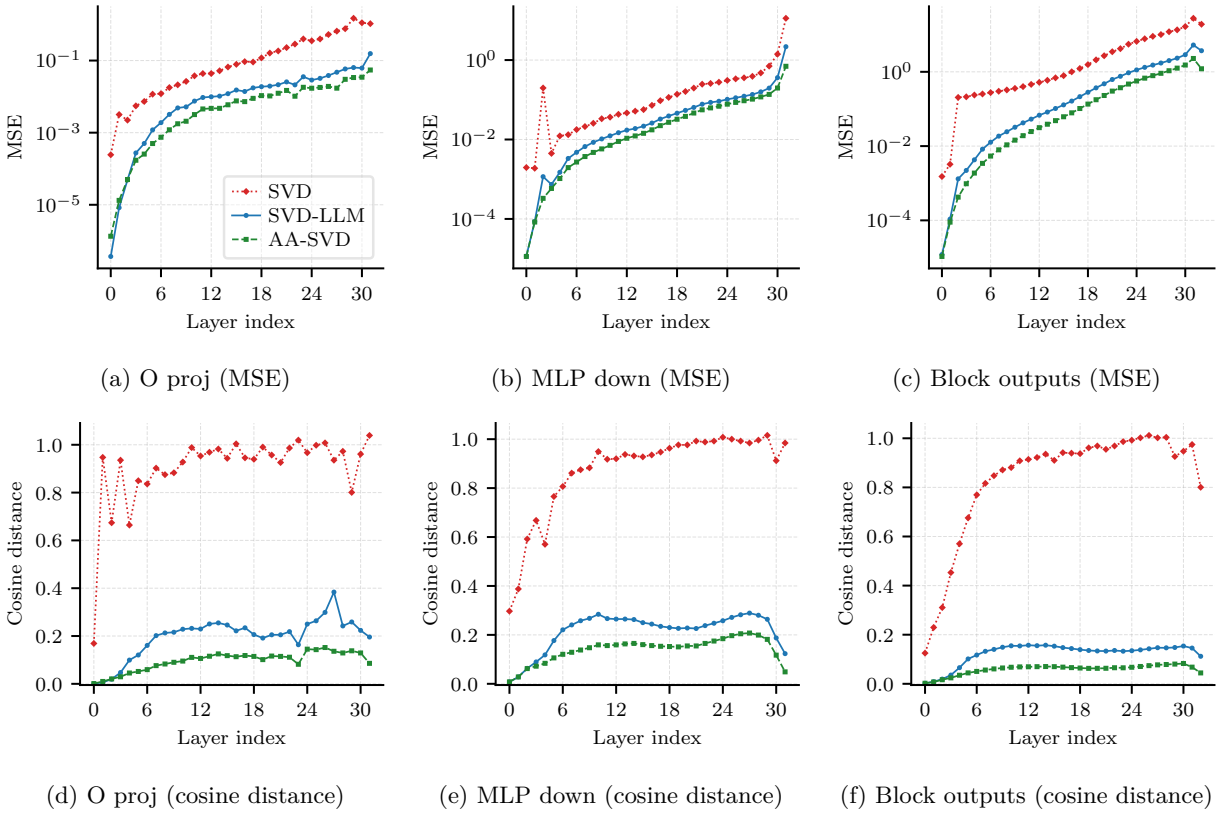
Figure 3: Impact of calibration set size on compression performance. Performance is measured by perplexity on WikiText2 (left) and C4 (middle), and average accuracy across seven zero-shot reasoning tasks (right).

AA-SVD against naive SVD and SVD-LLM (see also Figure 1). Naive SVD fails catastrophically from the very first layers: cosine distance immediately saturates near 1.0, indicating that compressed outputs are nearly orthogonal to the originals, with MSE several orders of magnitude above both data-driven methods. SVD-LLM recovers substantially, but its MSE and cosine distance still grow monotonically with depth and exhibit notable spikes in the deeper layers. AA-SVD achieves the lowest error throughout the network on both metrics: cosine distance rises gradually to $\sim$0.1–0.15 and then plateaus in later layers, while MSE remains consistently below SVD-LLM across all layer indices. The block-output MSE tells a similar story at a larger scale, reflecting cumulative error across all compressed layers within each transformer block—AA-SVD grows more slowly and maintains a clear margin over SVD-LLM at every depth. These results confirm that anchoring compression to the original outputs while accounting for shifted inputs curbs error accumulation across depth, which directly underlies the stronger end-task performance observed in the main results.

5 Limitations and Future Work

Our current method applies a uniform compression ratio across all layers, with per-layer ranks derived from that ratio rather than optimized individually. Existing capacity allocation methods are not directly applicable to our setting: sensitivity-based approaches evaluate each layer in isolation, but our final compressed factors emerge from block-level joint refinement, making per-layer sensitivity a poor proxy for the true cost of reducing a layer's capacity. More broadly, because our sequential compression propagates compressed activations forward, the optimal ratio for any layer depends on how upstream layers were compressed — a dependency that per-layer allocation schemes do not account for. Developing capacity allocation strategies that are aware of block-level refinement is a promising direction for further improving compression performance at a fixed parameter budget.

Beyond capacity allocation, our method also focuses exclusively on low-rank factorization. Structured pruning removes entire heads or channels, quantization reduces numerical precision, and low-rank factorization compresses weight matrices — each targeting a distinct source of redundancy. These techniques are largely orthogonal and can in principle be composed: for instance, low-rank factors could be quantized post-compression, or pruning could be applied to reduce capacity before factorization. Exploring hybrid pipelines that combine these methods, potentially within a unified block-level optimization framework, is a natural and promising direction for future work.



(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 4: 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.

6 Conclusion

We introduced a post-training framework for compressing large language models based on low-rank matrix factorization, with two key contributions. First, we proposed an anchored and adaptive layer-wise compression objective with a closed-form solution, efficient to compute with a small calibration set. Second, and more critically, we introduced block-level refinement, which jointly optimizes all compressed layers within a transformer block to minimize the block output error—accounting for interactions between compressed layers that layer-wise objectives cannot capture. Extensive experiments across LLaMA and Qwen model families at multiple scales, evaluated on language modeling and commonsense reasoning benchmarks, show that our approach consistently outperforms prior SVD-based methods. At moderate compression ratios our method preserves accuracy with negligible loss, while under aggressive compression it widens the gap to baselines. Overall, our study demonstrates that block-level refinement is the dominant factor enabling effective compression, and that careful initialization via a good layer-wise objective further improves final performance. Together, these results establish block-level optimization as a powerful and underexplored paradigm for structured model compression. We hope it contributes toward practical deployment of large-scale pretrained models and inspires further work in this direction.

References

Ali Abbasi, Chayne Thrash, Haoran Qin, Shansita Sharma, Sepehr Seifi, and Soheil Kolouri. Zero sum svd: Balancing loss sensitivity for low rank llm compression. *arXiv preprint arXiv:2602.02848*, 2026.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. *arXiv preprint arXiv:1905.13319*, 2019.

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. Fluctuation-based adaptive structured pruning for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 10865–10873, 2024.

Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*, 2024.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

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

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

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

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136, 2018.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. In *Proceedings of EMNLP*, pp. 279–290, 2018.

Misha Denil, Babak Shakibi, Laurent Dinh, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in neural information processing systems*, volume 26, 2013.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, volume 27, 2014.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Xuan Ding, Rui Sun, Yunjian Zhang, Xiu Yan, Yueqi Zhou, Kaihao Huang, Suzhong Fu, Chuanlong Xie, and Yao Zhu. Dipsvd: Dual-importance protected svd for efficient llm compression. *arXiv preprint arXiv:2506.20353*, 2025.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

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

Elias Frantar, Saleh Ashkboos, Pierre Stock, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Shangqian Gao, Ting Hua, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. 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.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

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

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NeurIPS Deep Learning Workshop*, 2015.

Xing Hu, Dawei Yang, Yuan Cheng, Zhixuan Chen, and Zukang Xu. Saes-svd: Self-adaptive suppression of accumulated and local errors for svd-based llm compression. *arXiv preprint arXiv:2602.03051*, 2026.

Yerlan Idelbayev and Miguel A Carreira-Perpinán. 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.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference*, 2014.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4163–4174, 2020.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Steven Kolawole, Lucio Dery, Jean-François Kagy, Virginia Smith, Graham Neubig, and Ameet Talwalkar. Everybody prune now: Structured pruning of llms with only forward passes. *arXiv preprint arXiv:2402.05406*, 2024.

Zhiteng Li, Mingyuan Xia, Jingyuan Zhang, Zheng Hui, Haotong Qin, Linghe Kong, Yulun Zhang, and Xiaokang Yang. Adasvd: Adaptive singular value decomposition for large language models. *arXiv preprint arXiv:2502.01403*, 2025.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

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

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

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

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

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

Guillermo Ortiz-Jiménez, Apostolos Modas, Seyed-Mohsen Moosavi, and Pascal Frossard. Neural anisotropy directions. *Advances in Neural Information Processing Systems*, 33:17896–17906, 2020.

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluís Munguia, Daniel Rothchild, David R So, Maud Texier, and Jeffrey Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

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. In *Journal of Machine Learning Research*, volume 21, pp. 1–67, 2020.

Roman Rausch, David Jansen, Sukhbinder Singh, and Román Orús. Globally optimized svd compression of llms via fermi-function-based rank selection and gauge fixing. *arXiv preprint arXiv:2512.03062*, 2025.

Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 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.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *arXiv preprint arXiv:1910.01108*, 2019.

Ryan Solgi, Parsa Madinei, Jiayi Tian, Rupak Swaminathan, Jing Liu, Nathan Susanj, and Zheng Zhang. Activation-informed pareto-guided low-rank compression for efficient llm/vlm. *arXiv preprint arXiv:2510.05544*, 2025.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Sidhant Sundrani, Francesco Tudisco, and Pasquale Minervini. Low-rank compression of language models via differentiable rank selection. *arXiv preprint arXiv:2512.13733*, 2025.

Chengxi Ye Tai, Tong Xiao, Yi Zhang, and Xiaogang Wang. Convolutional neural networks with low-rank regularization. In *International Conference on Learning Representations*, 2015.

Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Qinsi Wang, Jinghan Ke, Masayoshi Tomizuka, Yiran Chen, Kurt Keutzer, and Chenfeng Xu. Dobi-svd: Differentiable svd for llm compression and some new perspectives. In *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. 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.

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`.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*, 2024.

Lin Xv, Jingsheng Gao, Xian Gao, Ting Liu, and Yuzhuo Fu. Ara: Adaptive rank allocation for efficient large language model svd compression. *arXiv preprint arXiv:2510.19389*, 2025.

Zhihang Yuan, Yuzhang Shang, Yue Song, Qiang Wu, Yan Yan, and Guangyu Sun. Asvd: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 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.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Longguang Zhong, Fanqi Wan, Ruijun Chen, Xiaojun Quan, and Liangzhi Li. Blockpruner: Fine-grained pruning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 5065–5080, 2025.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12:1556–1577, 2024.

A Proofs and Discussion

A.1 Proof of Theorem 3.2

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

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

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

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

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

Expanding the squared Frobenius norm gives

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

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

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

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

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

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

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

A.2 Discussion of Corollary 3.3

Corollary 3.3 applies whenever $\boldsymbol{A} = \boldsymbol{B}$, so that $\boldsymbol{A}\boldsymbol{B}^\top = \boldsymbol{B}\boldsymbol{B}^\top = \boldsymbol{L_B}\boldsymbol{L_B}^\top$ and the general solution reduces to the whitening-based form

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

Two natural instantiations arise in our setting: setting $\boldsymbol{B} = \boldsymbol{X}$ (original inputs) yields an input-aware solution, while setting $\boldsymbol{B} = \boldsymbol{X}'$ (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 $\boldsymbol{B} = \boldsymbol{X}$ case, differing only in their factorization of $\boldsymbol{X}\boldsymbol{X}^\top$: SVD-LLM uses the lower-triangular Cholesky factor $\boldsymbol{L_X}$, while SVD-LLM V2 uses the eigendecomposition $\boldsymbol{X}\boldsymbol{X}^\top = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$ with $\boldsymbol{L_X} = \boldsymbol{Q}\boldsymbol{\Lambda}^{1/2}$, giving

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

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 $\boldsymbol{X}'$ via their covariance, projecting the cross-term $\boldsymbol{W}\boldsymbol{X}$ into this whitened space, applying truncated SVD, and mapping back. This closed-form solution generalizes the classical whitening construction ($\boldsymbol{X}' = \boldsymbol{X}$) and can be implemented efficiently with a Cholesky factorization. Importantly, our formulation operates only on the covariance matrices $\boldsymbol{X}\boldsymbol{X}'^\top$ and $\boldsymbol{X}'\boldsymbol{X}'^\top$ 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 ($\boldsymbol{X}\boldsymbol{X}^\top$, $\boldsymbol{X}\boldsymbol{X}'^\top$ and $\boldsymbol{X}'\boldsymbol{X}'^\top$), 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 $\{\boldsymbol{U}_j, \boldsymbol{V}_j\}$ and block-local parameters $\boldsymbol{\theta}_i$ to minimize the MSE between the original and compressed block outputs, as described in the main text. 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 $\boldsymbol{X}$ and $\boldsymbol{X}'$ 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 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 $\boldsymbol{W} \in \mathbb{R}^{m \times n}$. 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× 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.

Table 6: 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%**

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.4 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 $k \in [0, \min(m, n)]$[4].

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 Compression performance on more models

Tables 6–10 provide full per-benchmark breakdowns for the five additional models summarized in Table 2 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.

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

Table 7: Comparison of AA-SVD with SOTA methods for SVD-based compression of LLaMA-2-7B on three 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} represents quantized parameters.

Ratio	Method	PPL (↓)			Accuracy (↑)								
		Wiki2	PTB	C4	Openb.	ARC_e	ARC_c	WinoG.	PIQA	MathQA	HellaS.	Avg.	Drop (%)
1.0	Dense	5.47	24.09	7.28	0.32	0.76	0.43	0.69	0.78	0.28	0.57	0.55	–
0.8	SVD-LLM	8.41	119.32	20.34	0.26	0.57	0.26	0.62	0.66	0.24	0.39	0.43	21.7%
	AA-SVD	**6.84**	**1486.20**	**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	30.78	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	571.51	73.12	0.18	0.39	0.21	0.53	0.58	0.22	0.31	0.35	36.8%
	SAES-SVD	11.35	**217.20**	40.57	–	0.43	–	0.58	0.59	–	0.32	–	–
	AA-SVD	**8.55**	2688.10	**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	100.25	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	1612.91	615.24	0.13	0.27	0.22	0.49	0.53	0.20	0.27	0.30	44.9%
	SAES-SVD	23.89	**334.67**	100.42	–	0.31	–	0.52	0.55	–	0.30	–	–
	AA-SVD	**14.58**	4342.20	**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	528.41	22.48	0.26	0.60	0.30	0.61	0.65	0.25	0.40	0.44	**19.8%**

Table 8: 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 9: 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 10: 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%**