CoSpaDi: Compressing LLMs via Calibration-Guided Sparse Dictionary Learning

Anonymous authors
Paper under double-blind review

Abstract

Post-training compression of large language models (LLMs) largely relies on low-rank weight approximation, which represents each column of a weight matrix in a shared low-dimensional subspace. While this is a computationally efficient strategy, the imposed structural constraint is rigid and can lead to a noticeable model accuracy drop. In this work, we propose **CoSpaDi** (**Co**mpression via **Spa**rse **Di**ctionary Learning), a novel training-free compression framework that replaces low-rank decomposition with a more flexible structured sparse factorization in which each weight matrix is represented with a dense dictionary and a column-sparse coefficient matrix. This formulation enables a union-of-subspaces representation: different columns of the original weight matrix are approximated in distinct subspaces spanned by adaptively selected dictionary atoms, offering greater expressiveness than a single invariant basis. Crucially, **CoSpaDi** leverages a small calibration dataset to optimize the factorization such that the output activations of compressed projection layers closely match those of the original ones, thereby minimizing functional reconstruction error rather than mere weight approximation. This data-aware strategy preserves better model fidelity without any fine-tuning under reasonable compression ratios. Moreover, the resulting structured sparsity allows efficient sparse–dense matrix multiplication and is compatible with post-training quantization for further memory and latency gains. We evaluate **CoSpaDi** across multiple Llama and Qwen models under per-layer and per-group settings at $20 - 50\%$ compression ratios, demonstrating consistent superiority over state-of-the-art data-aware low-rank methods both in accuracy and perplexity. Our results establish structured sparse dictionary learning as a powerful alternative to conventional low-rank approaches for efficient LLM deployment.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable performance across a wide range of applications, from conversational agents Brown et al. (2020); OpenAI (2023) to general-purpose reasoning systems Touvron et al. (2023a); Anil et al. (2023), owing to their ability to capture long-range dependencies through attention mechanisms Vaswani et al. (2017); Devlin et al. (2019). However, this success entails substantial memory and computational demands during both training and inference, which severely limits their deployment in resource-constrained environments.

Various compression and acceleration techniques have been proposed over recent years to mitigate these challenges, including pruning Frankle & Carbin (2019); Sanh et al. (2020), quantization Dettmers et al. (2022); Xiao et al. (2023); Yao et al. (2022); Dettmers et al. (2023), and weight decomposition/factorization Denton et al. (2014); Hu et al. (2022); Liu et al. (2024); Zhang et al. (2023); Ma et al. (2019); Chen et al. (2018); Hsu et al. (2022); Yu & Wu (2023); Wang et al. (2025a).

Recently, there has been growing interest in effective compression of large models without retraining or fine-tuning, since this is often computationally prohibitive. A prominent line of work leverages low-rank approximations of weight matrices via Singular Value Decomposition (SVD). Early approaches such as DRONE Chen et al. (2021), FWSVD Hsu et al. (2022) and ASVD Yuan et al. (2023) introduced activation-aware truncation of singular values, while SVD-LLM Wang et al. (2025a) and its variant SVD-LLMv2 Wang et al. (2025b) proposed truncation-aware and optimized singular value selection strategies. In parallel, Basis Sharing Wang et al. (2024) explored cross-

layer sharing of a common basis to further enhance compression efficiency by exploiting inter-layer redundancy. Despite their effectiveness, these methods are inherently constrained by the use of a single shared low-dimensional subspace, which may limit representational flexibility and prevent full exploitation of redundancies in LLM parameters.

In this work, we argue that moving beyond the low-rank weight decomposition is a promising new direction. Specifically, we propose to adopt dictionary learning, a well-established paradigm in signal and image processing Aharon et al. (2006); Elad (2010), and apply it to LLM compression. Unlike low-rank approximations that confine all columns of a weight matrix to a shared linear subspace, dictionary learning enables a richer union-of-subspaces representation: each column is approximated using only a sparse subset of atoms from a learned dictionary, allowing different columns to reside in distinct subspaces spanned by different atom combinations. Such a representation better accommodates heterogeneous features and introduces additional flexibility to reduce the overall approximation error.

To this end, we introduce **CoSpaDi** (**Co**mpression via **Spa**rse **Di**ctionary Learning), a training-free framework that applies dictionary learning to jointly estimate dictionaries and sparse coefficients used for compressing LLM weight matrices. Our contributions are as follows:

- We introduce dictionary learning with sparse coding as a new paradigm for LLM compression, addressing the limitation of using a single invariant basis for the weight approximation which is inherent in SVD-based methods.
- We demonstrate that **CoSpaDi** is effectively integrated with data-aware optimization, yielding state-of-the-art compression performance for a wide range of compression ratios, while being compatible with post-training quantization for further memory and latency gains.
- Through extensive experiments, we show that our approach is effective in both per-layer and cross-layer (shared dictionary) compression scenarios, consistently outperforming regular SVD, ASVD Yuan et al. (2023), SVD-LLM Wang et al. (2025a), and Basis Sharing Wang et al. (2024) strategies as well as state-of-the-art structure-pruning methods Ma et al. (2023); Shopkhoev et al. (2025).

2 Related Work

Research on Transformer compression covers pruning, quantization, distillation, and matrix factorization. Below we provide a brief overview of the recent progress in these directions.

Early work on **pruning** showed substantial redundancy in deep nets Han et al. (2015); Frankle & Carbin (2019); Sanh et al. (2020). For LLMs, SparseGPT performs one-shot post-training pruning and achieves high sparsity with small quality drop Frantar & Alistarh (2023). Wanda reduces overhead further with simple activation-based rules Sun et al. (2023). LLM-Pruner removes blocks using gradient-guided criteria and recovers accuracy with brief adapter tuning Ma et al. (2023), while ReplaceMe substitutes multiple transformer blocks with a single linear transformation Shopkhoev et al. (2025).

Quantization reduces precision to reduce memory consumption and accelerate inference. For weight-only post-training quantization, LLM.int8 enables INT8 inference via vector-wise quantization with a path for outliers Dettmers et al. (2022); GPTQ/OPTQ minimize output error using Hessian to reach 3–4 bit weights Frantar et al. (2022; 2023); AWQ derives scales from activations and protects a small set of salient weights Lin et al. (2024), and SpQR stores outliers at higher precision to maintain quality at 3–4 bits Dettmers et al. (2024). For quantizing both weights and activations, SmoothQuant rebalances channel ranges to achieve accurate W8A8 across matrix multiplications Xiao et al. (2023), while QuaRot applies orthogonal rotations to suppress outliers, enabling end-to-end 4-bit inference including the KV cache Ashkboos et al. (2024).

The general framework on **knowledge distillation** by Hinton et al. and sequence-level distillation by Kim and Rush established strong baselines Hinton et al. (2015); Kim & Rush (2016). DistilBERT demonstrates task-agnostic compression for Transformers Sanh et al. (2019); TinyBERT uses a two-stage distillation procedure Jiao et al. (2020); MobileBERT introduces a teacher-guided compact architecture Sun et al. (2020); Patient-KD leverages multi-layer hints Sun et al. (2019); MiniLM and MiniLMv2 distill self-attention relations Wang et al. (2020; 2021). BERT-of-Theseus compresses models via progressive module replacement Xu et al. (2020). Orca and Orca 2 show that richer teacher signals such as explanations can improve reasoning in smaller students Mukherjee et al. (2023); Mitra et al. (2023).

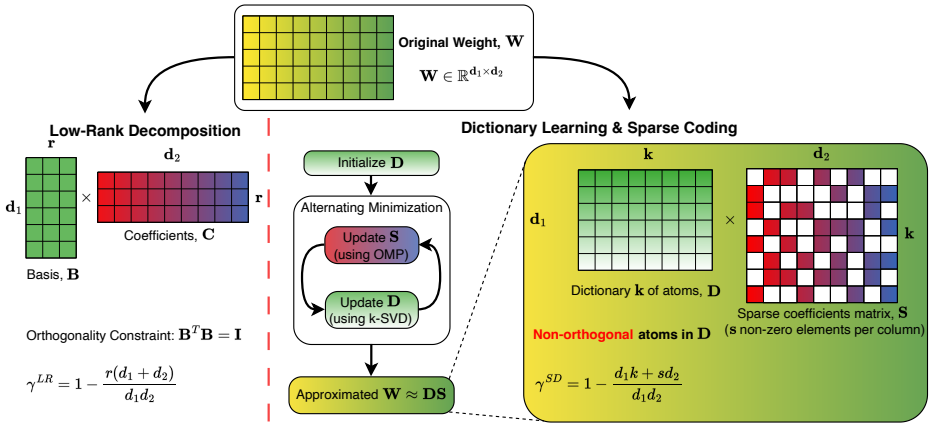


Figure 1: *Left side:* weight factorization methods using low-rank decomposition. Low-rank approximation decomposes a matrix into two dense matrices of lower rank. *Right side:* proposed CoSpaDi. A dictionary of k atoms and a column-sparse coefficient matrix are employed. No restrictions on size of k (undercomplete : $k < d_1$, complete: $k = d_1$ or overcomplete : $k > d_1$ dictionaries are possible), while sparsity is defined by s non-zero elements per column of the coefficient matrix.

Classic **low-rank approximations** reduce parameters and FLOPs with limited loss Denton et al. (2014). DRONE considers an objective related to the output activation for the weight approximation, Chen et al. (2021). FWSVD introduces Fisher-weighted reconstruction to emphasize important directions Hsu et al. (2022), while ASVD adapts truncation using activation-aware transforms and layer sensitivity Yuan et al. (2023). SVD-LLM utilizes truncation-aware whitening , while SVD-LLM v2 optimizes budget allocation across layers Wang et al. (2025a;b). Basis Sharing reuses one part of low-rank factorization across layers and learns layer-specific coefficients to exploit inter-layer redundancy Wang et al. (2024).

Dictionary learning and sparse coding. Dictionary learning factorizes original weight into dictionary and sparse codes, yielding a union-of-subspaces model with strong results in vision and image compression Engan et al. (1999); Aharon et al. (2006); Mairal et al. (2010); Gregor & LeCun (2010); Elad (2010). In NLP, GroupReduce explores block-wise low-rank/dictionary-style compression Chen et al. (2018), and tensorized Transformers factorize attention and embeddings Ma et al. (2019). Recent work also targets the KV-cache by learning universal dictionaries and decoding with OMP during inference Kim et al. (2024). Complementary to these directions, a recent work proposes cross-layer weight sharing via a matrix-based dictionary learning formulation for Transformer attention, directly exploiting inter-layer redundancy Zhussip et al. (2025). Cross-layer/shared-basis schemes in attention are thus closely related Wang et al. (2024). A comprehensive, training-free approach to weight-space dictionary learning for LLMs — at both per-layer and cross-layer levels — remains underexplored and is the focus of this work.

3 Proposed Method

In this section, we first provide a conceptual link between low-rank weight approximation and dictionary learning. Then, we introduce the proposed Sparse Dictionary Learning (SDL) strategy for LLM compression covering all practical aspects including data-aware SDL, dictionary-sharing among layers, compression ratios and inference complexity.

3.1 Low-Rank Weight Approximation

A widely adopted strategy to reduce the parameter count in LLMs is to approximate weight matrices $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ — representing the parameters of network layers — with matrices of reduced rank $r < \min(d_1, d_2)$. Such a low-rank approximation can be derived as follows:

$$\tilde{\mathbf{W}}^{\star} = \underset{\text{rank}(\tilde{\mathbf{W}})=r}{\arg\min} \left\| \mathbf{W} - \tilde{\mathbf{W}} \right\|_F . \tag{1}$$

According to the Eckart–Young–Mirsky theorem Eckart & Young (1936), the orthogonal projection to the space of r-rank matrices admits an analytical solution. Specifically, if $\mathbf{W}$ admits the singular value decomposition $\mathbf{W} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$, with $\mathbf{U} \in \mathbb{R}^{d_1 \times k}$, $\mathbf{\Sigma} \in \mathbb{R}^{k \times k}$ and $\mathbf{V} \in \mathbb{R}^{d_2 \times k}$ with $k = \min(d_1, d_2)$, then the minimizer of Eq. (1) can be expressed as: $\tilde{\mathbf{W}}^\star = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^\mathsf{T}$, where $\mathbf{U}_r \in \mathbb{R}^{d_1 \times r}$ contains the first r left singular vectors, $\mathbf{\Sigma}_r \in \mathbb{R}^{r \times r}$ holds the top-r singular values, and $\mathbf{V}_r \in \mathbb{R}^{d_2 \times r}$ contains the first r right singular vectors of $\mathbf{W}$, respectively.

There is a deep connection between this problem and the principal component analysis (PCA) Bishop & Nasrabadi (2006). Specifically, consider the weight matrix $\mathbf{W}$ as a collection of d_1-dimensional vectors $\mathbf{W} = [\mathbf{w}_1, \quad \ldots, \mathbf{w}_{d_2}]$, where $\boldsymbol{w}_j \in \mathbb{R}^{d_1}$, with $j = 1, \ldots, d_2$. We seek to approximate each vector $\mathbf{w}_j$ as a linear combination of basis vectors spanning a lower-dimensional subspace of $\mathbb{R}^{d_1}$. The optimal basis and coefficients can be found by minimizing the total approximation error:

$$\mathcal{J} = \sum_{j=1}^{d_2} \left\| \mathbf{w}_j - \sum_{i=1}^{r} c_{i,j}\boldsymbol{b}_i \right\|_2^2 \equiv \|\mathbf{W} - \mathbf{B}\mathbf{C}\|_F^2 , \tag{2}$$

subject to the orthogonality constraint $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}$, where $\mathbf{B} = [\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r] \in \mathbb{R}^{d_1 \times r}$ is the basis matrix, $\mathbf{C} \in \mathbb{R}^{r \times d_2}$ is the coefficient matrix with entries $c_{i,j}$, and $\mathbf{I} \in \mathbb{R}^{r \times r}$ is the identity matrix.

The optimal pair $(\mathbf{B}^*, \mathbf{C}^*)$ is obtained by solving:

$$\mathbf{B}^*, \mathbf{C}^* = \arg\min_{\mathbf{B},\mathbf{C}} \mathcal{J} \text{ s.t } \mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}. \tag{3}$$

The solution is given by $\mathbf{B}^* = \mathbf{U}_r$ and $\mathbf{C}^* = \mathbf{\Sigma}_r\mathbf{V}_r^\mathsf{T}$ (we refer to Appendix A.1 for the proof). This result highlights that in the low-rank factorization $\mathbf{W} \approx \mathbf{B}\mathbf{C}$, the matrix $\mathbf{B}$ corresponds to the basis of the shared r-dimensional subspace in which the columns of $\mathbf{W}$ approximately lie, and $\mathbf{C}$ contains their coordinate vectors (coefficients) with respect to that basis.

3.2 Sparse Dictionary Learning

Motivated by this interpretation, we propose an alternative strategy for compressing the weights $\mathbf{W}$, based on the sparse dictionary learning methodology. Specifically, rather than modeling each column of the weight matrix as a linear combination of basis vectors $\boldsymbol{b}_i$, we aim to learn a dictionary $\mathbf{D} \in \mathbb{R}^{d_1 \times k}$ and approximate each column $\mathbf{w}_j$ as a linear combination of dictionary atoms $\mathbf{d}_i \in \mathbb{R}^{d_1}$ with $i = 1, \ldots, k$.

Although the two strategies appear similar, they differ in two key respects. First, in dictionary learning, unlike low-rank approximation, only a *subset* of atoms is used to represent each weight vector $\mathbf{w}_j$. This principle, known in the literature as *sparse coding* Lee et al. (2006), is motivated by the observation that learned dictionaries typically yield sparse representations: each signal (column) activates only a few atoms. In other words, instead of approximating the weight matrix $\mathbf{W}$ as the product of two dense matrices, the orthogonal basis $\mathbf{B}$ and the coefficient matrix $\mathbf{C}$, we approximate it as the product of a dense dictionary matrix $\mathbf{D}$ and a *column-wise sparse* coefficient matrix $\mathbf{S}$, where each column $\mathbf{s}_j$ indicates which atoms contribute to the representation of $\mathbf{w}_j$, and with what weights. Second, and of significant practical consequence, we do not impose orthogonality constraints on the dictionary atoms. This grants greater representational flexibility, allowing the model to adapt more effectively to the intrinsic structure of the weight matrix.

Finally, under the proposed strategy, rather than enforcing a *shared* low-dimensional subspace for all columns, each weight vector $\mathbf{w}_j$ is represented within its own *low-dimensional subspace* spanned by the atoms it activates. Consequently, the entire matrix $\mathbf{W}$ is modeled as a *union of low-dimensional subspaces*, introducing additional representational capacity and enabling lower approximation error compared to a single shared subspace.

Formally, the dictionary learning problem can be expressed as:

$$\mathbf{D}^\star, \mathbf{S}^\star = \arg\min_{\mathbf{D},\mathbf{S}} \|\mathbf{W} - \mathbf{D}\mathbf{S}\|_F^2 \quad \text{s.t.} \quad \|\mathbf{s}_j\|_0 \leq s \quad \forall\, j = 1, \ldots, d_2, \tag{4}$$

where $\mathbf{s}_j$ denotes the j-th column of the sparse coefficient matrix $\mathbf{S}$, and $\|\cdot\|_0$ denotes the ℓ_0 pseudo-norm, which counts the number of nonzero entries in a vector.

The optimization problem in Eq. (4) is NP-hard in its original form and admits no closed-form solution. Nevertheless, it has been extensively studied, and several efficient algorithms have been proposed to obtain high-quality approximate solutions. Among these, K-SVD Aharon et al. (2006) is a well-established algorithm widely adopted across diverse applications.

K-SVD alternates between two main steps: (a) **Sparse coding**: Each signal $\mathbf{w}_j$ is approximated as a sparse linear combination of dictionary atoms (typically via orthogonal matching pursuit or LASSO). (b) **Dictionary update**: Each atom $\mathbf{d}_i$ is refined to better fit the data while preserving the sparsity pattern in $\mathbf{S}$. This is performed sequentially per atom: the algorithm identifies the subset of signals that activate $\mathbf{d}_i$, computes the residual error over those signals, and applies a rank-1 SVD to jointly update the atom and its associated coefficients.

3.3 CoSpaDi: Activation-Aware SDL

In several recent works, including Chen et al. (2021), it has been observed that LLM weight matrices generally do not exhibit intrinsic low-rank structure. Consequently, direct low-rank approximation of the weights often leads to significant performance degradation. However, under the hypothesis that the input latent features to a given layer reside in a low-dimensional subspace, it remains feasible and often effective to approximate the corresponding layer weights as low-rank. This approach is well-motivated: the goal is not to preserve the weights in isolation, but rather to maintain the fidelity of the *activations* they produce when applied to low-dimensional inputs.

Within this framework, an effective approximation of $\mathbf{W}$ can be obtained by minimizing the output error:

$$\tilde{\mathbf{W}}^* = \arg\min_{\tilde{\mathbf{W}}} \left\| \mathbf{X}\mathbf{W} - \mathbf{X}\tilde{\mathbf{W}} \right\|_F \quad \text{s.t.} \quad \operatorname{rank}(\mathbf{X}\tilde{\mathbf{W}}) = r, \tag{5}$$

where $\mathbf{X} \in \mathbb{R}^{N \times d_1}$ is a matrix of N calibration input vectors $\left\{\mathbf{x}_i \in \mathbb{R}^{d_1}\right\}_{i=1}^{N}$. This minimization admits the analytical solution: $\tilde{\mathbf{W}} = \mathbf{L}^{-1}\mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^{\mathsf{T}} = \mathbf{B}\mathbf{C}$, where $\mathbf{L} \in \mathbb{R}^{d_1 \times d_1}$ is a non-singular transformation matrix derived from the calibration data (see Appendix A.2 for the derivation).

Motivated by this data-aware perspective, we propose adapting our sparse dictionary learning framework accordingly. Rather than directly approximating $\mathbf{W}$ as $\tilde{\mathbf{W}} = \mathbf{D}\mathbf{S}$, we instead minimize the reconstruction error of the *output activations* $\mathbf{Z} = \mathbf{X}\mathbf{W}$:

$$\mathbf{D}^\star, \mathbf{S}^\star = \arg\min_{\mathbf{D},\mathbf{S}} \|\mathbf{X}\mathbf{W} - \mathbf{X}\mathbf{D}\mathbf{S}\|_F^2 \quad \text{s.t.} \quad \|\mathbf{s}_j\|_0 \le s \quad \forall\, j = 1, \ldots, d_2. \tag{6}$$

To simplify this optimization, let $\mathbf{Y} = \mathbf{X}\mathbf{L}^{-1} \in \mathbb{R}^{N \times d_1}$ be a column-orthogonal matrix, i.e., $\mathbf{Y}^{\mathsf{T}}\mathbf{Y} = \mathbf{I}_{d_1}$, obtained by applying a linear transformation $\mathbf{L} \in \mathbb{R}^{d_1 \times d_1}$ to $\mathbf{X}$. We assume $N \ge d_1$ and that $\mathbf{X}$ has full column rank — conditions typically satisfied with sufficient calibration data. The matrix $\mathbf{L}$ can be computed via QR decomposition, SVD of $\mathbf{X}$, or Cholesky/eigen-decomposition of $\mathbf{X}^{\mathsf{T}}\mathbf{X}$.

Introducing the auxiliary variables $\mathbf{W}_L = \mathbf{L}\mathbf{W}$ and $\mathbf{D}_L = \mathbf{L}\mathbf{D}$, and noting that the Frobenius norm is invariant under left-multiplication by a column-orthogonal matrix, we can reformulate Eq. (6) as:

$$\mathbf{D}_L^\star, \mathbf{S}^\star = \arg\min_{\mathbf{D}_L,\mathbf{S}} \|\mathbf{W}_L - \mathbf{D}_L\mathbf{S}\|_F^2 \quad \text{s.t.} \quad \|\mathbf{s}_j\|_0 \le s \quad \forall\, j = 1, \ldots, d_2. \tag{7}$$

Since $\mathbf{L}$ is invertible, the final structured approximation of the original weights is given by:

$$\tilde{\mathbf{W}} = \mathbf{D}_a\mathbf{S}, \quad \text{where} \quad \mathbf{D}_a = \mathbf{L}^{-1}\mathbf{D}_L \tag{8}$$

is the *activation-aware learned dictionary*. Pseudocode for the full procedure is provided in Appendix A.3.

Cross-Layer SDL Although most compression techniques focus on intra-layer optimizations, the repetitive layered structure of LLMs suggests the presence of significant *inter-layer redundancy*. To exploit this, we propose sharing a single dictionary across weight matrices of the same type from multiple layers. Specifically, let $\mathcal{G}$ denote a group of layer indices (e.g., all attention projection layers or all FFN layers). We form a grouped weight matrix by horizontally concatenating the corresponding layer weights: $\mathbf{W}_G = [\mathbf{W}_{\ell_1}, \quad \mathbf{W}_{\ell_2}, \quad \ldots, \quad \mathbf{W}_{\ell_L}] \in \mathbb{R}^{d_1 \times (d_2 \cdot L)}$, where each $\mathbf{W}_\ell \in \mathbb{R}^{d_1 \times d_2}$,

and $L = |\mathcal{G}|$ is the number of layers in the group. We then apply the same activation-aware dictionary learning procedure as before, but now to $\mathbf{W}_G$. To ensure the approximation remains data-aware across all layers in the group, we construct a grouped calibration matrix by vertically stacking the corresponding input batches: $\mathbf{X}_G = [\mathbf{X}_{\ell_1}; \quad \mathbf{X}_{\ell_2}; \quad \ldots; \quad \mathbf{X}_{\ell_L}] \in \mathbb{R}^{(N \cdot L) \times d_1}$ where each $\mathbf{X}_\ell \in \mathbb{R}^{N \times d_1}$ is the calibration input for layer ℓ. The transformation matrix $\mathbf{L} \in \mathbb{R}^{d_1 \times d_1}$ is then computed from $\mathbf{X}_G$ (e.g., via QR or Cholesky decomposition), ensuring that $\mathbf{Y}_G = \mathbf{X}_G \mathbf{L}^{-1}$ is column-orthogonal. The optimization proceeds by solving Eq. (7) with $\mathbf{W}_L = \mathbf{L}\mathbf{W}_G$, yielding a shared dictionary $\mathbf{D}_a$ and a sparse coefficient matrix $\mathbf{S}_G \in \mathbb{R}^{k \times (d_2 \cdot L)}$.

This approach, inspired by Wang et al. (2024), reduces memory overhead by amortizing the dictionary cost across multiple layers, while preserving activation fidelity through data-aware calibration. Each layer's compressed weights are then recovered by slicing the corresponding block from $\tilde{\mathbf{W}}_G = \mathbf{D}_a \mathbf{S}_G$.

Compression ratio. For the low-rank setting, the compression ratio is $\gamma^{\mathrm{LR}} \coloneqq 1 - \frac{r(d_1+d_2)}{d_1 d_2}$, where r is the retained rank. For CoSpaDi, we store an overcomplete dictionary and sparse codes plus a binary mask (details are provided in appendix A.4), giving

$$\gamma^{\mathrm{SD}} \coloneqq 1 - \frac{\overbrace{d_1 k}^{\text{dict. (bf16)}} + \overbrace{s d_2}^{\text{codes (bf16)}} + \overbrace{(k d_2)/16}^{\text{mask (1 bit/entry)}}}{d_1 d_2}. \tag{9}$$

Unlike the low-rank case, γ^{SD} depends on two knobs — the number of atoms k and sparsity s — allowing us to trade off model capacity and storage at fixed budget. We parameterize this trade-off by the ratio $\rho \coloneqq k/s$, which uniquely determines (k, s) at a target γ^{SD}:

$$k = \frac{(1 - \gamma^{\mathrm{SD}})\, d_1 d_2}{d_1 + \frac{d_2}{\rho} + \frac{d_2}{16}}, \qquad s = \tfrac{k}{\rho}. \tag{10}$$

Inference Complexity In terms of computational efficiency, low-rank and dictionary learning exhibit distinct inference-time complexity profiles: the former has a cost of $Tr(d_1 + d_2)$, whereas the latter – when exploiting sparsity and reusing inner products – achieves $Td_1K_{\text{active}} + Tsd_2$ (see appendix A.5), potentially yielding superior efficiency under favorable sparsity patterns. Although both methods share identical theoretical complexity under matched compression ratios, practical inference latency varies significantly due to factors such as the number of active atoms, indexing overhead, and hardware-specific kernel efficiency. Further details regarding complexity derivations are provided in Appendix A.5.

4 Experiments

In this section, we present our experimental setup. We first compare CoSpaDi with low-rank baselines in the *per-layer* setting, where each linear weight matrix is compressed independently. We then investigate cross-layer sharing, analyze the impact of coefficient matrix quantization, and conduct ablation studies on the ks-ratio.

4.1 Experimental Setup

We evaluate our method in both per-layer and cross-layer settings. For per-layer evaluations, we consider LLaMA-3.2 1B, Qwen-3 0.6B, LLaMA-3 8B, and Qwen-3 8B. For the cross-layer study, we follow the Basis Sharing Wang et al. (2024) setup and conduct experiments on LLaMA-2 7B. All models are evaluated in a zero-shot setting on PIQA Bisk et al. (2019), HellaSwag Zellers et al. (2019), OpenAI LAMBADA Paperno et al. (2016), ARC-easy and ARC-challenge Clark et al. (2018), SciQ Welbl et al. (2017), Race Lai et al. (2017), and MMLU Hendrycks et al. (2021), while perplexity is additionally reported on WikiText Merity et al. (2016) and LAMBADA-OpenAI. Compression is applied with target ratios of 0.2-0.5 with 0.1 step. For methods requiring calibration data, we randomly sample 256 sequences from the RefinedWeb dataset Penedo et al. (2023), each consisting of 1024 tokens. Unless specified otherwise, we compress all dense linear projections in self-attention (Q/K/V/O) and feed-forward network (gate/up/down). Embeddings and `lm_head` are left intact.

For low-rank methods the rank is uniquely defined with γ^{LR} and we floored it to the nearest integer. For CoSpaDi we fixed $\rho = 2$ (k/s ratio) for all experiments, so k and s were obtained according to Eq. (10) and then were floored to the nearest integers.

In CoSpaDi we employ K-SVD using power iterations instead of full-svd for the dictionary updates. Specifically, we use 60 K-SVD iterations and 8 power iterations to ensure stable convergence. Further implementation details and convergence analysis are provided in Appendix A.8.

4.2 Ablation Study

Influence of k/s-ratio. In the proposed CoSpaDi we can redistribute capacity across both k and s while preserving predefined compression ratio by varying the ρ value (k/s). We verified its influence on the Llama 3.2 1B and report the metrics in Fig. 2.

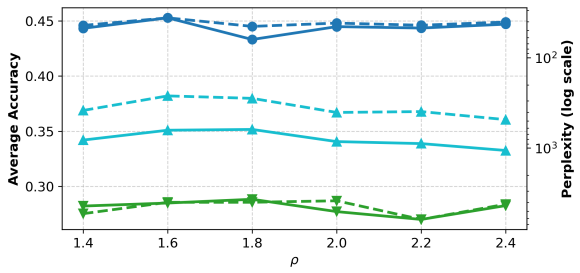


Figure 2: Dual-axis plot showing average accuracy (—— solid lines, left axis) and perplexity (– – – dashed lines, right axis, logarithmic scale with inverted direction) as functions of ρ for Llama3.2-1B under three compression levels: 0.2, 0.3 and 0.4. Perplexity decreases upward due to axis inversion.

CR	Bitwidth	Avg. Acc.	PPL
0.1686	bFP16	0.6198	1.94E+01
0.1843	bFP15	0.6195	1.95E+01
0.2001	bFP14	0.6176	1.97E+01
0.2726	bFP16	0.5408	4.34E+01
0.2864	bFP15	0.5394	4.37E+01
0.3002	bFP14	0.5373	4.46E+01
0.3765	bFP16	0.4096	1.80E+02
0.3883	bFP15	0.4086	1.82E+02
0.4002	bFP14	0.4069	1.84E+02
0.4804	bFP16	0.2846	8.23E+02
0.4902	bFP15	0.2838	8.15E+02
0.5001	bFP14	0.2837	8.10E+02

Table 1: Results on truncation of bfloat16 mantissa bits of coefficient matrix with reported average accuracy (Avg. Acc.) and Wiki Text word perplexity (PPL) for Llama3-8B resulting in different compression ratios (CR).

From the provided plots we can observe that depending on the compression ratio the optimal ks-ratio differs, thus, for simplicity in further experiments we select in all cases $\rho = 2$.

Data-Free and Data-Aware Scenarios We argue that our proposed CoSpaDi allows for a more flexible representation compared to low-rank approximation, regardless of whether data-free or data-aware scenarios are considered. To prove this claim we performed an ablation study on Llama3-1B for different compression levels. We selected SVD for the data-free scenario and SVD-LLM for the data-aware case, while varying the compression ratio from 0.2 to 0.5. In Table 2 we report the results only for 0.2-0.5 compression. These results clearly indicate that in both data-free and data-aware settings the SDL-based methods outperform low-rank baselines by a wide margin, while CoSpaDi outperforms all methods.

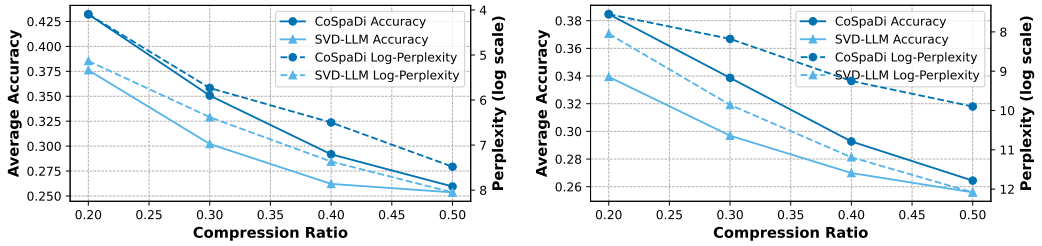
Quantization of Sparse Coefficient Matrix We investigated the quantization of the coefficient matrix $\mathbf{S}$ in the post-training regime to leverage the memory add-on due to the requirement of storing the indices of the nonzero values. We utilized a naive mantissa bit truncation of the original bfloat16 coefficient values. In Table 1 we report how truncation affects the performance at different compression levels. According to the obtained results, truncation of 2 bits leads to negligible drop. Thus, in all our experiments CoSpaDi consists of bfloat16 dictionary and coefficient matrices but is evaluated with the truncated version of the sparse coefficient matrix using 14-bits.

4.3 Main Results

Per-Layer Scenario: In this section we apply CoSpaDi for each projection layer independently and compare it against current sota training-free low-rank method – SVD-LLM Wang et al. (2025a).

Table 2: SDL-based methods comparison vs low-rank counterparts in data-free and data-aware scenarios on Llama3.2-1B at different compression ratios (CR). We denote CoSpaDi† as the proposed method without using calibration data. Best results are provided in **bold**.

Method	Data-Aware	CR	Accuracy↑									Perplexity↓	
			PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	Wiki Text	LAMBADA
Llama3.2 1B			0.7453	0.6366	0.6295	0.6047	0.3620	0.8830	0.3779	0.3700	0.5761	11.57	5.73
SVD	x	0.2	0.5201	0.2566	0.0003	0.2471	0.2210	0.2000	0.2144	0.2537	0.2391	2.93E+06	4.56E+06
CoSpaDi†	x		0.5174	0.2635	0.0002	0.2538	0.2534	0.2100	0.2182	0.2413	0.2447	3.33E+05	2.16E+06
SVD-LLM	v		0.6213	0.3643	0.2443	0.3603	0.2509	0.6490	0.2900	0.2298	0.3762	1.69E+02	1.70E+02
CoSpaDi	v		0.6605	0.4288	0.3839	0.3994	0.2602	0.7160	0.3167	0.2483	**0.4267**	**6.37E+01**	**3.51E+01**
SVD	x	0.3	0.5234	0.2564	0.0002	0.2416	0.2346	0.1950	0.2191	0.2698	0.2425	1.09E+06	3.92E+06
CoSpaDi†	x		0.5049	0.2626	0.0002	0.2449	0.2611	0.2180	0.2153	0.2552	0.2453	2.06E+05	4.34E+06
SVD-LLM	v		0.5565	0.3006	0.0910	0.3047	0.2150	0.4590	0.2583	0.2323	0.3022	5.90E+02	2.47E+03
CoSpaDi	v		0.5691	0.3241	0.1822	0.3194	0.2210	0.5670	0.2804	0.2308	**0.3368**	**2.89E+02**	**6.59E+02**
SVD	x	0.4	0.5277	0.2590	0.0001	0.2386	0.2133	0.1990	0.2220	0.2693	0.2411	1.23E+06	4.17E+06
CoSpaDi†	x		0.5103	0.2626	0.0001	0.2546	0.2688	0.2130	0.2115	0.2549	0.2470	3.05E+06	3.67E+07
SVD-LLM	v		0.5180	0.2725	0.0126	0.2685	0.2287	0.3230	0.2440	0.2297	0.2621	1.58E+03	3.30E+04
CoSpaDi	v		0.5348	0.2824	0.0380	0.2778	0.2295	0.3690	0.2402	0.2310	**0.2753**	**7.96E+02**	**9.23E+03**
SVD	x	0.5	0.5392	0.2566	0.0001	0.2441	0.2039	0.2040	0.2077	0.2690	0.2406	1.22E+06	1.88E+07
CoSpaDi†	x		0.5120	0.2607	0.0001	0.2572	0.2654	0.2280	0.2211	0.2704	0.2519	6.09E+05	1.08E+07
SVD-LLM	v		0.5109	0.2663	0.0004	0.2609	0.2594	0.2610	0.2392	0.2303	0.2536	3.13E+03	1.03E+05
CoSpaDi	v		0.5169	0.2703	0.0031	0.2626	0.2398	0.2950	0.2421	0.2326	**0.2578**	**1.80E+03**	**7.26E+04**



(a) Accuracy & Perplexity for LLaMA3.2-1B

(b) Accuracy & Perplexity for Qwen3-0.6B

Figure 3: Average benchmark accuracy and WikiText perplexity for (a) LLaMA-3.2-1B and (b) Qwen-3 0.6B using SVD-LLM and CoSpaDiwith respect to compression ratio.

From the provided plots CoSpaDi significantly outperforms SVD-LLM in both avg. accuracy and perplexity for small Llama3 and Qwen3 models. We further investigate how our method scales to larger models of the same families with 8B parameters. The related results are provided in Table 3.

We have also performed comparisons on the same models and compression levels against other training-free compression strategies. In particular we compare CoSpaDi against ASVD Yuan et al. (2023), ReplaceMe Shopkhoev et al. (2025) and LLMPruner Ma et al. (2023), with the last two being sota pruning methods. These results are provided in Appendix A.6

Cross-Layer Scenario In this section, we validate CoSpaDi on the cross-layer compression scenario, where a common dictionary is shared across different layers. To ensure a fair comparison, we adopt the identical layer selection and grouping procedure introduced in Basis Sharing Wang et al. (2024) (for a detailed description we refer to Appendix A.7) and compare the performance of the two methods on the Llama2-7B model Touvron et al. (2023b). As shown in Table 4, CoSpaDi consistently outperforms Basis Sharing by a large margin across all benchmarks and compression rates (0.2–0.5). This performance gap suggests that CoSpaDi's sparsity-aware, dictionary-based decomposition better preserves task-critical features under aggressive compression.

We believe that a more sophisticated layer grouping strategy, based on feature map similarity or end-to-end differentiable search, could work better with CoSpaDi and further widen this performance margin, but we leave this as a future research direction. We provide details for k (dictionary size) and s (sparsity) parameters in the Appendix A.7.

Table 3: Performance comparison of CoSpaDi vs sota SVD-LLM on Llama3-8B and Qwen3-8B at different compression levels on different benchmarks. Best results are highlighted with **bold**.

Method	CR	Accuracy↑									Perplexity↓	
		PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	Wiki Text	LAMBADA
Llama3 8B		0.8069	0.7913	0.7557	0.7769	0.5350	0.9390	0.4029	0.6215	0.7036	7.26E+00	3.09E+00
SVD-LLM	0.2	0.7106	0.5837	0.5925	0.5551	0.3396	0.8640	0.3550	0.3262	0.5408	4.07E+01	1.09E+01
CoSpaDi		0.7519	0.6649	0.7380	0.6654	0.4155	0.8950	0.3818	0.4282	**0.6176**	**1.97E+01**	**4.27E+00**
SVD-LLM	0.3	0.6578	0.4640	0.3806	0.4188	0.2765	0.7000	0.3177	0.2715	0.4358	1.47E+02	6.09E+01
CoSpaDi		0.7051	0.5620	0.6128	0.5421	0.3353	0.8570	0.3617	0.3224	**0.5373**	**4.46E+01**	**9.18E+00**
SVD-LLM	0.4	0.6028	0.3450	0.1143	0.3237	0.2449	0.4420	0.2574	0.2305	0.3201	5.49E+02	1.30E+03
CoSpaDi		0.6371	0.4138	0.3029	0.3914	0.2662	0.6850	0.3053	0.2538	**0.4069**	**1.84E+02**	**1.19E+02**
SVD-LLM	0.5	0.5381	0.2790	0.0041	0.2731	0.2389	0.2660	0.2230	0.2297	0.2565	1.37E+03	2.80E+04
CoSpaDi		0.5642	0.3102	0.0390	0.2988	0.2218	0.3770	0.2287	0.2301	**0.2837**	**8.10E+02**	**7.64E+03**
Qwen3 8B		0.7769	0.7494	0.6408	0.8068	0.5674	0.9570	0.4086	0.7295	0.7045	1.22E+01	4.58E+00
SVD-LLM	0.2	0.7383	0.6391	0.6218	0.6869	0.4573	0.9010	0.4048	0.5473	0.6246	2.06E+01	6.40E+00
CoSpaDi		0.7650	0.6804	0.6559	0.7218	0.4889	0.9320	0.4067	0.6075	**0.6573**	**1.81E+01**	**4.88E+00**
SVD-LLM	0.3	0.7035	0.5522	0.5377	0.5926	0.3712	0.8720	0.3837	0.4482	0.5576	2.74E+01	1.07E+01
CoSpaDi		0.7242	0.6050	0.6255	0.6385	0.4121	0.8840	0.3952	0.5127	**0.5997**	**2.29E+01**	**6.29E+00**
SVD-LLM	0.4	0.6627	0.4458	0.3790	0.4503	0.2807	0.7730	0.3531	0.2901	0.4544	4.30E+01	3.59E+01
CoSpaDi		0.6888	0.4898	0.4991	0.4941	0.2986	0.8200	0.3675	0.3660	**0.5030**	**3.59E+01**	**1.47E+01**
SVD-LLM	0.5	0.6023	0.3543	0.1956	0.3363	0.2278	0.6670	0.2909	0.2305	0.3631	8.72E+01	2.69E+02
CoSpaDi		0.6159	0.3798	0.2791	0.3565	0.2295	0.6860	0.3167	0.2305	**0.3868**	**6.82E+01**	**1.10E+02**

Table 4: Performance comparison of CoSpaDi vs Basis Sharing Wang et al. (2024) on Llama2-7B under different compression levels on various benchmarks. Best results are highlighted with **bold**.

Method	CR	Accuracy↑									Perplexity↓	
		PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	Wiki Text	LAMBADA
Llama2 7B		0.7797	0.5709	0.7367	0.7609	0.4326	0.9390	0.3923	0.4087	0.6276	8.71	3.40
Basis Sharing	0.2	0.7008	0.4337	0.6297	0.6646	0.3370	0.9100	0.3493	0.2476	0.5341	15.18	7.24
CoSpaDi		0.7459	0.6627	0.7132	0.6738	0.3933	0.8880	0.3828	0.2720	**0.5915**	**11.72**	**4.32**
Basis Sharing	0.3	0.6556	0.3757	0.5354	0.5880	0.2730	0.8680	0.3263	0.2317	0.4817	22.23	13.57
CoSpaDi		0.7084	0.5850	0.6447	0.6191	0.3430	0.8760	0.3589	0.2404	**0.5469**	**15.42**	**6.31**
Basis Sharing	0.4	0.6061	0.3310	0.4114	0.4945	0.2355	0.8300	0.2967	0.2305	0.4295	39.51	37.67
CoSpaDi		0.6431	0.4801	0.5180	0.5231	0.2986	0.8150	0.3321	0.2300	**0.4800**	**25.24**	**14.58**
Basis Sharing	0.5	0.5631	0.2975	0.2377	0.3729	0.2048	0.7450	0.2536	0.2282	0.3628	98.56	225.09
CoSpaDi		0.5751	0.3656	0.3264	0.4171	0.2491	0.7470	0.2632	0.2288	**0.3965**	**57.31**	**72.61**

Limitations and Future Work In this work, we employ k-SVD as a representative instantiation of our framework, though other dictionary learning algorithms could be similarly applied. A key limitation of k-SVD lies in its reliance on Orthogonal Matching Pursuit (OMP) for sparse coding and its sequential atom updates, which can be computationally slow. However, more efficient variants do exist that accelerate convergence while maintaining the same performance and we plan to explore them in the future. As an additional direction for future work, we plan to investigate adaptive capacity allocation across the model by dynamically distributing sparsity and dictionary size according to layer-specific demands, while leveraging cross-layer dictionary sharing in a structured and scalable manner.

5 Conclusions

In summary, we introduced **CoSpaDi**, a novel training-free compression framework for LLMs that challenges the dominance of low-rank approximation in post-training model compression. By introducing dictionary learning with sparse coding as a more expressive alternative, we shift from the rigid constraint of a shared linear subspace to a flexible union-of-subspaces representation, where each column of a weight matrix is approximated using a sparse combination of learned dictionary atoms. This work demonstrates that moving beyond SVD-driven paradigms, long considered the default for matrix compression in LLMs, can yield significant gains in model fidelity under aggressive compression. We hope **CoSpaDi** serves as a conceptual stepping stone toward richer, data-aware factorizations inspired by classical signal processing, yet tailored for the complexity of modern language models.

REFERENCES

Michal Aharon, Michael Elad, and Alfred Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11): 4311–4322, 2006. doi: 10.1109/TSP.2006.881199.

Rohan Anil, Sebastian Borgeaud, Jiecao Chen, Aakanksha Chowdhery, Jonathan Clark, et al. Palm 2 technical report. In *arXiv preprint arXiv:2305.10403*, 2023.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. In *Advances in Neural Information Processing Systems*, 2024.

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 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.

Patrick H. Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. In *Advances in Neural Information Processing Systems*, pp. 11011–11021, 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, 2018.

Emily 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 (NeurIPS)*, 2014.

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

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression. In *International Conference on Learning Representations*, 2024.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

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

Michael Elad. *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*. Springer, 2010.

Kjersti Engan, Sven Ole Aase, and J. Håkon Husø y. Method of optimal directions for frame design. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 5, pp. 2443–2446, 1999. doi: 10.1109/ICASSP.1999.760624.

Ky Fan. On a theorem of Weyl concerning eigenvalues of linear transformations I. *Proceedings of the National Academy of Sciences of the United States of America*, 35(11):652–655, 1949.

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

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10323–10337. PMLR, 2023.

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

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Optq: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations*, 2023.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pp. 399–406, 2010.

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 (NeurIPS)*, 2015.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. Language model compression with weighted low-rank factorization. In *International Conference on Learning Representations (Workshop Track)*, 2022. ICLR 2022 Workshop.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

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. doi: 10.18653/v1/2020.findings-emnlp.372.

Junhyuck Kim, Jongho Park, Jaewoong Cho, and Dimitris Papailiopoulos. Lexico: Extreme kv cache compression via sparse coding over universal dictionaries, 2024.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1317–1327, 2016. doi: 10.18653/v1/D16-1139.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations, 2017.

Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Ng. Efficient sparse coding algorithms. *Advances in neural information processing systems*, 19, 2006.

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. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 6, 2024.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *International Conference on Machine Learning*, 2024. Oral.

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Dawei Song, and Ming Zhou. A tensorized transformer for language modeling. In *Advances in Neural Information Processing Systems*, 2019.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, 2023.

Julien Mairal, Francis R. Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11:19–60, 2010.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, Ahmed Awadallah, and Subhabrata Mukherjee. Orca 2: Teaching small language models how to reason. *arXiv preprint arXiv:2311.11045*, 2023.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707*, 2023.

OpenAI. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context, 2016.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023.

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

Victor Sanh, Thomas Wolf, and Alexander M. Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Dmitriy Shopkhoev, Ammar Ali, Magauiya Zhussip, Valentin Malykh, Stamatios Lefkimmiatis, Nikos Komodakis, and Sergey Zagoruyko. Replaceme: Network simplification via depth pruning and transformer block linearization, 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.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for BERT model compression. In *Proceedings of EMNLP-IJCNLP 2019*, pp. 4323–4332, 2019. doi: 10.18653/v1/D19-1441.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: A compact task-agnostic bert for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 2158–2170, 2020. doi: 10.18653/v1/2020.acl-main.195.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. In *International Conference on Learning Representations (ICLR)*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Jingcun Wang, Yu-Guang Chen, Ing-Chao Lin, Bing Li, and Grace Li Zhang. Basis sharing: Cross-layer parameter sharing for large language model compression. *arXiv preprint arXiv:2410.03765*, 2024.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020.

Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. Minilmv2: Multi-head self-attention relation distillation for compressing pretrained transformers. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 2140–2151, 2021. doi: 10.18653/v1/2021.findings-acl.188.

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*, 2025a.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. Svd-llm v2: Optimizing singular value truncation for large language model compression. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2025b.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions, 2017.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning (ICML)*, 2023.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. Bert-of-theseus: Compressing bert by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7859–7869, 2020.

Zhewei Yao, Jiarui Zhao, Shiyang Zhang, Boxiao Wang, Ye Zhang, George Biros, Dan Alistarh, Kurt Keutzer, Michael W. Mahoney, and Joseph E. Gonzalez. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

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

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?, 2019.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*, 2023.

Magauiya Zhussip, Dmitriy Shopkhoev, Ammar Ali, and Stamatios Lefkimmiatis. Share your attention: Transformer weight sharing via matrix-based dictionary learning. *arXiv preprint arXiv:2508.04581*, 2025.

A APPENDIX

A.1 DERIVATION OF THE OPTIMAL PAIR OF BASIS AND COEFFICIENT MATRICES

We are seeking the optimal pair $(\mathbf{B}^\star, \mathbf{C}^\star)$ that minimizes the constrained problem in Eq. (3). First, we rewrite the objective $\mathcal{J}$ in its equivalent form:

$$\mathcal{J} = \operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{W}\right) - 2\operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{B}\mathbf{C}\right) + \operatorname{tr}\left(\mathbf{C}^\mathsf{T}\mathbf{B}^\mathsf{T}\mathbf{B}\mathbf{C}\right). \tag{11}$$

Next, we consider the basis matrix $\mathbf{B}$ as fixed and compute the gradient of the objective w.r.t the matrix coefficient $\mathbf{C}$ as

$$\nabla_\mathbf{C}\mathcal{J} = 2\mathbf{B}^\mathsf{T}\mathbf{B}\mathbf{C} - 2\mathbf{B}^\mathsf{T}\mathbf{W}. \tag{12}$$

Setting the gradient to zero and taking into account the constraint $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}$, we can recover the optimum matrix coefficient as: $\mathbf{C} = \mathbf{B}^\mathsf{T}\mathbf{W}$. Now, putting back $\mathbf{C}$ in Eq. (11) we get:

$$\begin{aligned}\mathcal{J} &= \operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{W}\right) - 2\operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{B}\mathbf{B}^\mathsf{T}\mathbf{W}\right) + \operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{B}\mathbf{B}^\mathsf{T}\mathbf{B}\mathbf{B}^\mathsf{T}\mathbf{W}\right) \\ &= \operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{W}\right) - \operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{B}\mathbf{B}^\mathsf{T}\mathbf{W}\right) \text{ (from the constraint } \mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}\text{)} \\ &= \operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{W}\right) - \operatorname{tr}\left(\mathbf{B}^\mathsf{T}\mathbf{W}\mathbf{W}^\mathsf{T}\mathbf{B}\right). \end{aligned} \tag{13}$$

Based on this we can recover the optimum basis matrix $\mathbf{B}$ as the maximizer of the constrained problem:

$$\mathbf{B}^\star = \arg\max_{\mathbf{B}} \operatorname{tr}\left(\mathbf{B}^\mathsf{T}\mathbf{W}\mathbf{W}^\mathsf{T}\mathbf{B}\right) \text{ s.t } \mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}. \tag{14}$$

The above maximization problem enjoys a closed-form solution Fan (1949), which is fully defined by the eigenvalues of the matrix $\mathbf{P} = \mathbf{W}\mathbf{W}^\mathsf{T}$. Specifically, the matrix $\mathbf{P} \in \mathbb{R}^{d_1 \times d_1}$, which is symmetric and positive semi-definite, admits the eigenvalue decomposition $\mathbf{P} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\mathsf{T}$, with $\mathbf{U} \in \mathbb{R}^{d_1 \times d_1}$ holding the eigenvectors of $\mathbf{P}$ in its columns. Then, the maximizer of Eq. (14) is recovered as $\mathbf{B}^\star = \mathbf{U}_r$ where $\mathbf{U}_r \in \mathbb{R}^{d_1 \times r}$ is a reduced version of $\mathbf{U}$ formed with the r eigenvectors corresponding to the largest eigenvalues of $\mathbf{P}$. A useful observation is that the eigenvectors of $\mathbf{P}$ exactly match the left-singular vectors of $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$. Indeed, if $\mathbf{W}$ admits the singular value decomposition $\mathbf{W} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$, then we have that: $\mathbf{P} = \mathbf{W}\mathbf{W}^\mathsf{T} = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\mathsf{T} \equiv \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\mathsf{T}$, with $\mathbf{\Lambda} = \mathbf{\Sigma}^2$. Therefore, instead of performing the eigenvalue decomposition on $\mathbf{P}$ we can recover $\mathbf{U}$ and consequently $\mathbf{U}_r$ by computing the SVD of $\mathbf{W}$.

Finally, we can compute the optimum coefficient matrix as:

$$\begin{aligned}\mathbf{C}^\star &= \left(\mathbf{B}^\star\right)^\mathsf{T}\mathbf{W} = \mathbf{U}_r^\mathsf{T}\mathbf{W} \\ &= \mathbf{U}_r^\mathsf{T}\overbrace{\begin{bmatrix}\mathbf{U}_r & \mathbf{U}_{d-r}\end{bmatrix}}^{\mathbf{U}}\overbrace{\begin{bmatrix}\mathbf{\Sigma}_r & \boldsymbol{O}_{r\times(d-r)} \\ \mathbf{O}_{(d-r)\times r} & \mathbf{\Sigma}_{d-r}\end{bmatrix}}^{\mathbf{\Sigma}}\overbrace{\begin{bmatrix}\mathbf{V}_r^\mathsf{T} \\ \mathbf{V}_{d-r}^\mathsf{T}\end{bmatrix}}^{\mathbf{V}^\mathsf{T}} \\ &= \begin{bmatrix}\mathbf{I}_{r\times r} & \mathbf{O}_{r\times(d-r)}\end{bmatrix}\begin{bmatrix}\mathbf{\Sigma}_r\boldsymbol{V}_r^\mathsf{T} \\ \mathbf{\Sigma}_{d-r}\mathbf{V}_{d-r}^\mathsf{T}\end{bmatrix} \\ &= \mathbf{\Sigma}_r\boldsymbol{V}_r^\mathsf{T}, \end{aligned} \tag{15}$$

where $\boldsymbol{V}_r \in \mathbb{R}^{d_2 \times r}$ is a reduced version of $\mathbf{V}$ formed with the r right singular vectors of $\mathbf{W}$ that correspond to its top-r singular values, which are kept in the diagonal matrix $\mathbf{\Sigma}_r \in \mathbb{R}^{r\times r}$.

A.2 DATA-AWARE LOW-RANK WEIGHT APPROXIMATION

While the low-rank approximation of the weights $\mathbf{W}$ has been extensively used for compression tasks, in practice is not well suited to LLMs and it can lead to a severe drop of their performance . Several recent works have suggested that instead of approximating the weights $\mathbf{W}$ with a low-rank matrix, a more efficient strategy is to model the weight activations, $\mathbf{Z} = \mathbf{X}\mathbf{W}$, as low-rank. Here, the matrix $\mathbf{X} = \begin{bmatrix}\mathbf{x}_1 & \dots & \mathbf{x}_N\end{bmatrix}^\mathsf{T} \in \mathbb{R}^{N\times d}$ holds in its rows the d-dimensional input vectors $\mathbf{x}_n$

with $n = 1 \ldots, N$, which play the role of calibration data. Under this modeling framework, we can approximate the matrix weights $\mathbf{W}$ as the minimizer of the following problem:

$$\tilde{\mathbf{W}}^* = \arg\min_{\tilde{\mathbf{W}}} \left\| \mathbf{X}\mathbf{W} - \mathbf{X}\tilde{\mathbf{W}} \right\|_F \text{ s.t. } \operatorname{rank}\left(\mathbf{X}\tilde{\mathbf{W}}\right) = r. \tag{16}$$

Let us now consider $\mathbf{Y} = \mathbf{X}\mathbf{L}^{-1} \in \mathbb{R}^{N \times d_1}$ to be a semi-orthogonal matrix (column-orthogonal matrix), that is $\mathbf{Y}^\mathsf{T}\mathbf{Y} = \mathbf{I}_{d_1}$, which is obtained by linearly transforming the matrix $\mathbf{X}$ using a non-singular matrix $\mathbf{L} \in \mathbb{R}^{d_1 \times d_1}$. Here we assume that $N \geq d$ and the matrix $\mathbf{X}$ is of full rank. We note that there are different ways we can achieve this column-orthogonalization of $\mathbf{X}$. Among them we can employ the QR/SVD decomposition on $\mathbf{X}$ and the Cholesky/Eigen-value decomposition on $\mathbf{X}^\mathsf{T}\mathbf{X}$ to compute a proper linear transformation $\mathbf{L}$. By using the representation $\mathbf{X} = \mathbf{Y}\mathbf{L}$ we can rewrite the problem of Eq. (16) as:

$$\tilde{\mathbf{W}}^* = \arg\min_{\tilde{\mathbf{W}}} \left\| \mathbf{Y}\mathbf{L}\mathbf{W} - \mathbf{Y}\mathbf{L}\tilde{\mathbf{W}} \right\|_F \text{ s.t. } \operatorname{rank}\left(\mathbf{Y}\mathbf{L}\tilde{\mathbf{W}}\right) = r. \tag{17}$$

To solve the above minimization problem we first note that due to the orthonormal columns of $\mathbf{Y}$, it can be expressed in the equivalent form:

$$\tilde{\mathbf{W}}^* = \arg\min_{\tilde{\mathbf{W}}} \left\| \mathbf{\Delta} - \mathbf{L}\tilde{\mathbf{W}} \right\|_F \text{ s.t. } \operatorname{rank}\left(\mathbf{L}\tilde{\mathbf{W}}\right) = r, \tag{18}$$

where $\mathbf{\Delta} = \mathbf{L}\mathbf{W}$. Next, we introduce the auxiliary matrix $\tilde{\mathbf{\Delta}} = \mathbf{L}\tilde{\mathbf{W}}$ and the problem in Eq. (18) becomes:

$$\tilde{\mathbf{\Delta}}^* = \arg\min_{\tilde{\mathbf{\Delta}}} \left\| \mathbf{\Delta} - \tilde{\mathbf{\Delta}} \right\|_F \text{ s.t. } \operatorname{rank}\left(\tilde{\mathbf{\Delta}}\right) = r, \tag{19}$$

which is the orthogonal projection of $\mathbf{\Delta}$ to the space of r-rank matrices. Given that $\tilde{\mathbf{\Delta}}^* = \mathbf{L}\tilde{\mathbf{W}}^*$ and $\mathbf{L}$ is invertible, we can now recover $\tilde{\mathbf{W}}^* = \mathbf{L}^{-1}\tilde{\mathbf{\Delta}}^*$.

To conclude, if $\mathbf{\Delta}$ admits the singular value decomposition $\mathbf{\Delta} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$, then the optimal r-rank approximation of $\mathbf{W}$ that minimizes the loss in Eq. (16) can be written in the form:

$$\tilde{\mathbf{W}} = \mathbf{B}\mathbf{C} = \underbrace{\mathbf{L}^{-1}\mathbf{U}_r}_{\mathbf{B}} \underbrace{\mathbf{\Sigma}_r\mathbf{V}_r^\mathsf{T}}_{\mathbf{C}}. \tag{20}$$

We note that in this case, unlike the direct weight low-rank approximation, the matrix $\mathbf{B} = \mathbf{L}^{-1}\mathbf{U}_r$ does not correspond to a basis of a subspace of $\mathbb{R}^{d_1}$, since its columns are no longer orthonormal, that is $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{U}_r^\mathsf{T}\left(\mathbf{L}\mathbf{L}^\mathsf{T}\right)^{-1}\mathbf{U}_r \neq \mathbf{I}$.

A.3 Pseudo Algorithm of the Proposed Method

Goal. Given a weight matrix $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ and a small calibration set $\mathbf{X} \in \mathbb{R}^{N \times d_1}$, compute an activation-aware sparse–dictionary factorization $\mathbf{W} \approx \widetilde{\mathbf{W}} = \mathbf{D}\mathbf{S}$ under a target compression ratio. The procedure consists of whitening the activation objective, alternating sparse coding and dictionary updates on the whitened weights, and a final de-whitening step.

(1) Calibration and whitening. Compute an invertible transform $\mathbf{L} \in \mathbb{R}^{d_1 \times d_1}$ (e.g., via QR/SVD of $\mathbf{X}$ or Cholesky of $\mathbf{X}^\top\mathbf{X}$) such that $\mathbf{Y} = \mathbf{X}\mathbf{L}^{-1}$ has orthonormal columns ($\mathbf{Y}^\top\mathbf{Y} = \mathbf{I}$). Left-multiply $\mathbf{W}$ to obtain the whitened weights $\mathbf{W}_L = \mathbf{L}\mathbf{W}$. Whitening converts the data-aware loss $\|\mathbf{X}\mathbf{W} - \mathbf{X}\widehat{\mathbf{W}}\|_F^2$ into a standard Frobenius objective $\|\mathbf{W}_L - \mathbf{D}_L\mathbf{S}\|_F^2$ that is amenable to dictionary learning.

(2) Initialization. Initialize the whitened dictionary $\mathbf{D}_\mathbf{L}^{(\mathbf{0})} \in \mathbb{R}^{d_1 \times k}$ (e.g., random permutation of columns of $\mathbf{W}$) and set $\mathbf{S}^{(0)} = 0$. The pair (k, s) is set from the target compression ratio via Eq. 10 optionally using the fixed ratio $\rho = k/s$.

(3) Alternating minimization. Repeat for $t = 1, \ldots, T$: *(a) Sparse coding.* For each column j, solve

$$\mathbf{s_j^{(t)}} \in \arg\min_{\|\mathbf{s}\|_0 \le s} \left\|(\mathbf{W}_L)_{:,j} - \mathbf{D_L}^{(t-1)}\mathbf{s}\right\|_2^2,$$

using OMP (greedy selection with orthogonal residual updates) to enforce exactly s nonzeros per column. *(b) Dictionary update.* For each atom i, collect its support $\Omega_i = \{j : s_{i,j}^{(t)} \neq 0\}$ and form the residual on those columns:

$$\mathbf{R}_i = \mathbf{W}_L[:, \Omega_i] - \sum_{\ell \neq i} \mathbf{D_L}_{,\ell}^{(t-1)}\, \mathbf{s}_{\ell,\Omega_i}^{(t)}.$$

Update $(\mathbf{D_L}_{,\mathbf{i}}^{(t)}, \mathbf{s}_{i,\Omega_i}^{(t)})$ by the best rank-1 approximation $\mathbf{R}_i \approx \mathbf{u}\,\sigma\,\mathbf{v}^\top$ (set $/mD_{L,i}^{(t)} \leftarrow \mathbf{u}$, $\mathbf{s}_{i,\Omega_i}^{(t)} \leftarrow \sigma\mathbf{v}^\top$). This preserves the current sparsity pattern while reducing the residual. Iterate atoms sequentially. Stop when the maximum iteration T is reached or when the relative improvement falls below a tolerance.

(4) De-whitening and packing. Map the dictionary back to activation space via $\mathbf{D_a} = \mathbf{L^{-1}}\mathbf{D_L^{(T)}}$ and set $\widetilde{\mathbf{W}} = \mathbf{D}_a\mathbf{S}^{(T)}$. For storage, keep $\mathbf{D}_a$ and the sd_2 nonzero entries of $\mathbf{S}$ in `bf16` along with a packed binary mask $\mathbf{M} \in \{0,1\}^{k \times d_2}$ for locations (one bit per entry; $\frac{kd_2}{16}$ words). This yields the compression ratio in Appendix A.4 and Eq. 9.

(5) Inference. At runtime, apply $\widetilde{\mathbf{W}}$ as $\mathrm{matmul}(\mathbf{x}, \mathbf{D}_a\mathbf{S})$ with sparse–dense kernels. Reuse inner products $\langle \mathbf{x}, \mathbf{D}_{\mathbf{a},:,\mathbf{i}} \rangle$ across columns to achieve the complexity in Appendix A.5; the number of active atoms controls the practical speedup.

(6) Cross-layer variant. For a group of layers $\mathbf{G}$, concatenate weights horizontally $\mathbf{W_G} = [\mathbf{W}_{\ell_1} \cdots \mathbf{W}_{\ell_L}]$ and stack calibration batches vertically to form $\mathbf{X}_G$. Compute $\mathbf{L}$ from $\mathbf{X}_G$, run the same alternating procedure on $\mathbf{W}_{L,G} = \mathbf{L}\mathbf{W}_G$ to obtain a single shared $\mathbf{D}_a$, and slice the corresponding blocks of $\mathbf{S}_G$ back to per-layer coefficients.

A.4 Derivation of the CoSpaDi Compression Ratio

We derive the expression for the compression ratio γ^{SD} of our sparse–dictionary (SD) parameterization. Let $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ be factorized as

$$\mathbf{W} \approx \mathbf{D}\mathbf{S}, \qquad \mathbf{D} \in \mathbb{R}^{d_1 \times k},\ \mathbf{S} \in \mathbb{R}^{k \times d_2},$$

where each column of $\mathbf{S}$ has exactly s nonzeros (*column-s-sparse*). Throughout, we store real values in `bfloat16` (16 bits) as is common in modern LLMs.

Dense baseline. A dense $\mathbf{W}$ requires $d_1 d_2$ `bf16` values.

Dictionary term. The dictionary $\mathbf{D}$ stores $d_1 k$ `bf16` values.

Sparse codes. Naively, $\mathbf{S}$ would need kd_2 values. Since $\mathbf{S}$ is column-s-sparse, only sd_2 values are stored. For locations, one option is COO: per nonzero we keep a row index and (redundantly) the column index. Because sparsity is fixed per column, column indices can be omitted; keeping only row indices yields sd_2 indices. With 16-bit indices, the total becomes sd_2 values + sd_2 indices $= 2sd_2$ 16-bit words. For typical $\rho := k/s = 2$, this equals kd_2 words—offering no savings over dense $\mathbf{S}$ storage.

Instead, we use a *bit mask* $\mathbf{M} \in \{0,1\}^{k \times d_2}$ to mark nonzero positions. This requires kd_2 bits, i.e., $kd_2/16$ 16-bit words after packing. We then store sd_2 `bf16` values for the nonzeros and the packed mask for their positions.

Total and ratio. The SD parameterization thus stores

$$\underbrace{d_1 k}_{\text{dictionary}} + \underbrace{sd_2}_{\text{values}} + \underbrace{\tfrac{kd_2}{16}}_{\text{mask}}$$

Algorithm 1: Pseudo algorithm of the proposed CoSpaDi which consists of two steps: (a) sparse coding to compute the coefficients and (b) sequential dictionary update step.

Input : $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$: weight matrix to compress
$\mathbf{X} \in \mathbb{R}^{N \times d_1}$: calibration input data (N samples)
k: dictionary size (number of atoms, $k \geq s$)
s: sparsity level (max nonzeros per column in $\mathbf{S}$)
T: number of K-SVD iterations

Output: $\mathbf{D}_a \in \mathbb{R}^{d_1 \times k}$: activation-aware dictionary
$\mathbf{S} \in \mathbb{R}^{k \times d_2}$: sparse coefficient matrix
$\tilde{\mathbf{W}} = \mathbf{D}_a \mathbf{S}$: compressed weight matrix

Compute $\mathbf{L} \in \mathbb{R}^{d_1 \times d_1}$ such that $\mathbf{Y} = \mathbf{X}\mathbf{L}^{-1}$ satisfies $\mathbf{Y}^\mathsf{T}\mathbf{Y} = \mathbf{I}_{d_1}$;
`% e.g., via QR:` $\mathbf{X} = \mathbf{Q}\mathbf{R} \Rightarrow \mathbf{L} = \mathbf{R}$
`% e.g., via Cholesky:` $\mathbf{X}^\mathsf{T}\mathbf{X} = \mathbf{C}^\mathsf{T}\mathbf{C} \Rightarrow \mathbf{L} = \mathbf{C}$
$\mathbf{W}_L \leftarrow \mathbf{L}\mathbf{W}$;
Initialize $\mathbf{D}_L^{(0)} \in \mathbb{R}^{d_1 \times k}$ with random Gaussian or SVD-based atoms;
Initialize $\mathbf{S}^{(0)} \in \mathbb{R}^{k \times d_2}$ as zero matrix;
for $t = 1$ **to** T **do**
  for $j = 1$ **to** d_2 **do**
    $\mathbf{s}_j^{(t)} \leftarrow \arg\min_{\|\mathbf{s}\|_0 \leq s} \left\| \mathbf{W}_{L,j} - \mathbf{D}_L^{(t-1)}\mathbf{s} \right\|_2^2$;
    `% Solve via OMP, LASSO, or thresholding`
  end
  for $i = 1$ **to** k **do**
    $\Omega_i \leftarrow \left\{ j \mid s_{i,j}^{(t)} \neq 0 \right\}$;
    if $\Omega_i \neq \emptyset$ **then**
      $\mathbf{R}_i \leftarrow \mathbf{W}_L[:, \Omega_i] - \sum_{l \neq i} \mathbf{d}_{L,l}^{(t-1)} \mathbf{s}_{l,\Omega_i}^{(t)}$;
      $[\mathbf{u}, \sigma, \mathbf{v}] \leftarrow$ rank-1 SVD of $\mathbf{R}_i$;
      $\mathbf{d}_{L,i}^{(t)} \leftarrow \mathbf{u}$;
      $\mathbf{s}_{i,\Omega_i}^{(t)} \leftarrow \sigma \cdot \mathbf{v}^\mathsf{T}$;
    end
  end
end
$\mathbf{D}_a \leftarrow \mathbf{L}^{-1}\mathbf{D}_L^{(T)}$;
$\tilde{\mathbf{W}} \leftarrow \mathbf{D}_a \mathbf{S}^{(T)}$;
return $\mathbf{D}_a$, $\mathbf{S}^{(T)}$, $\tilde{\mathbf{W}}$;

16-bit words. Relative to the dense baseline $d_1 d_2$, the resulting compression ratio is

$$\gamma^{\mathrm{SD}} := 1 - \frac{\overbrace{d_1 k}^{\text{dict. (bf16)}} + \overbrace{s d_2}^{\text{codes (bf16)}} + \overbrace{(k d_2)/16}^{\text{mask (1 bit/entry)}}}{d_1 d_2}.$$

This matches the expression used in the main text and makes explicit the dependence on the two design knobs (k, s).

A.5 Low-rank and CoSpaDi Inference Complexity

Here we derive the multiplication complexity for the original weight, SVD-compressed weight, and dictionary-learning (k-SVD) compression. We count multiplications only (additions are of the same order). Let $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ be a projection matrix in some layer and $\mathbf{X} \in \mathbb{R}^{N \times d_1}$ be an input feature map; then a dense product $\mathbf{Y} = \mathbf{X}\mathbf{W}$ costs

$$O_{baseline} = N d_1 d_2. \tag{21}$$

For low-rank, in particular, SVD compression with rank r the projection matrix is approximated with two matrices $\mathbf{W} \approx \mathbf{UV}$ with $\mathbf{U} \in \mathbb{R}^{d_1 \times r}$ and $\mathbf{V} \in \mathbb{R}^{r \times d_2}$, resulting in the following complexity:

$$O_{LR} = Nd_1 r + Nrd_2 = Nr(d_1 + d_2). \tag{22}$$

Sparse dictionary (SD) learning similarly represents $\mathbf{W} \approx \mathbf{DS}$ with dictionary $\mathbf{D} \in \mathbb{R}^{d_1 \times k}$ of k atoms and sparse coefficient matrix $\mathbf{S} \in \mathbb{R}^{k \times d_2}$. Omitting sparsity of $\mathbf{S}$ will result in:

$$O_{SD,dense} = Nd_1 k + Nkd_2 = Nk(d_1 + d_2). \tag{23}$$

Taking into account that each column $\mathbf{s}_j$ of $\mathbf{S}$ is s-sparse, the (i, j) element of $\mathbf{Y} = \mathbf{XDS}$ is

$$y_{i,j} = \sum_{k=1}^{K} \mathbf{S}_{k,j} \langle \mathbf{X}_{i,:}, \mathbf{D}_{:,k} \rangle = \sum_{k \in \mathbf{S}_j} \mathbf{S}_{k,j} \langle \mathbf{X}_{i,:}, \mathbf{D}_{:,k} \rangle, \tag{24}$$

where $S_j = \mathrm{supp}(s_j)$ and $|\mathbf{S}_j| = s$. The overall sparse complexity depends on whether the inner products $\langle \mathbf{X}_{i,:}, \mathbf{D}_{:,k} \rangle$ are reused across columns. With the most efficient way with reuse letting $\mathbf{U} = \bigcup_{j=1}^{d_2} S_j$ and $K_{active} = |U|$ we have:

$$O_{SD,sparse-reuse} = Nd_1 K_{active} + Nsd_2, \qquad s \leq K_{active} \leq \min(K, sd_2). \tag{25}$$

With proposed truncation of 2-bits in mantissa we can omit term for storing indices of nonzero elements in $\mathbf{S}$ resulting in corrected compression ratio:

$$\hat{\gamma}^{SD} = 1 - \frac{d_1 k + sd_2}{d_1 d_2}$$

The rank for the low-rank decomposition could be estimated from the compression ratio with the following equation:

$$r = \frac{(1 - \gamma^{LR}) d_1 d_2}{d_1 + d_2}$$

For sparse dictionary based method we defined $\rho = k/s$ and, thus we can estimate both k and s in the following way:

$$k = \frac{(1 - \hat{\gamma}^{\mathrm{SD}})\, d_1 d_2}{d_1 + \frac{d_2}{\rho}}, \qquad s = \tfrac{k}{\rho}.$$

Under the same compression ratio for both SVD and CoSpaDi we obtain exactly the same complexity:

$$O_{LR} = Nr(d_1 + d_2) = N(1 - \gamma^{LR}) d_1 d_2$$

$$O_{SD} = Nd_1 k + Nsd_2 = N(d_1 k + d_2 \frac{k}{\rho}) = Nk(d_1 + \frac{d_2}{\rho}) = N \frac{(1 - \hat{\gamma}^{\mathrm{SD}})\, d_1 d_2}{d_1 + \frac{d_2}{\rho}} (d_1 + \frac{d_2}{\rho}) = N(1 - \hat{\gamma}^{\mathrm{SD}})\, d_1 d_2$$

$$O_{LR} = O_{SD} = N(1 - \gamma) d_1 d_2. \tag{26}$$

While, theoretical complexity is the same, in practice inference time depends on the sparsity structure (K_{active}), indexing overhead, and kernel efficiency, as can be observed from Figs. 4, 5 and 6.

A.6 Comparison with Other Training-Free Methods

We also evaluate the performance of the proposed CoSpaDi relative to other training free methods, particularly structural pruning ones and ASVD Yuan et al. (2023) which is another data-aware SVD-based method. The results are provided in Table 5.

Across LLaMA-3 8B, our method consistently outperforms structural pruning baselines at moderate budgets. At CR ≈ 0.2, CoSpaDi attains the best average accuracy (0.618 vs. 0.580 for ReplaceMe and 0.551 for LLM-Pruner) while also delivering the lowest perplexities, indicating balanced generative and discriminative quality. At CR ≈ 0.3, CoSpaDi widens the margin in average accuracy (0.537 vs. 0.469/0.405) and avoids the severe perplexity blow-ups observed for pruning methods. Even under aggressive compression (CR ≈ 0.4–0.5), CoSpaDi remains competitive or superior on most tasks and preserves far more stable perplexity profiles, whereas ReplaceMe and LLM-Pruner exhibit task collapse (near-zero LAMBADA and extreme PPL spikes). Overall, the results support dictionary–sparsity as a more robust data-free alternative to structural pruning at matched or tighter budgets, especially when both accuracy and perplexity must be maintained.

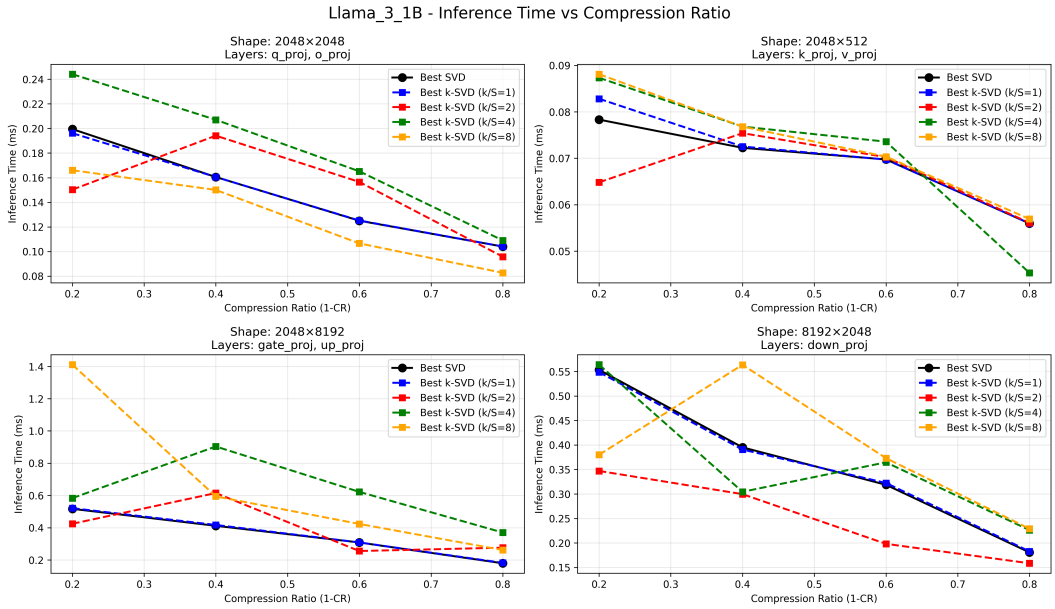


Figure 4: Inference time for different projection layers of Llama3.2 1B for different compression ratios and k/s ratios on A100

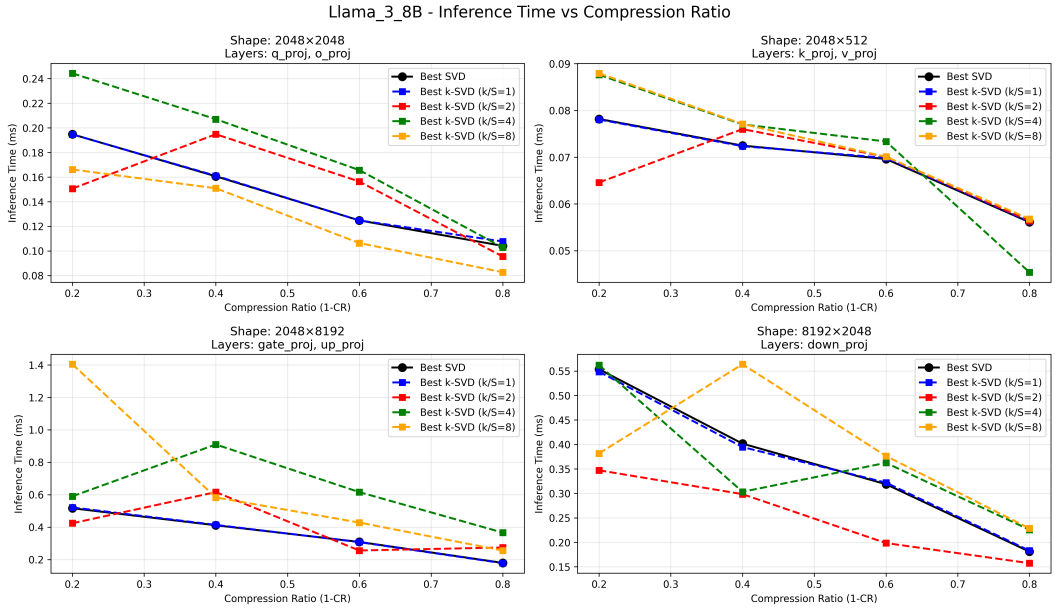


Figure 5: Inference time for different projection layers of Llama3 8B for different compression ratios and k/s ratios on A100

A.7 Cross-Layer Scenario

In Table 6, we report the dictionary size (k), sparsity level (s), and input/output weight dimensions for each linear transform under a fixed k/s ratio. This includes the query, key, value, and output projections within the attention block, as well as the up, down, and gated projections in the gated MLP block. The dictionary and sparsity dimensions are adjusted according to the target compression rate.

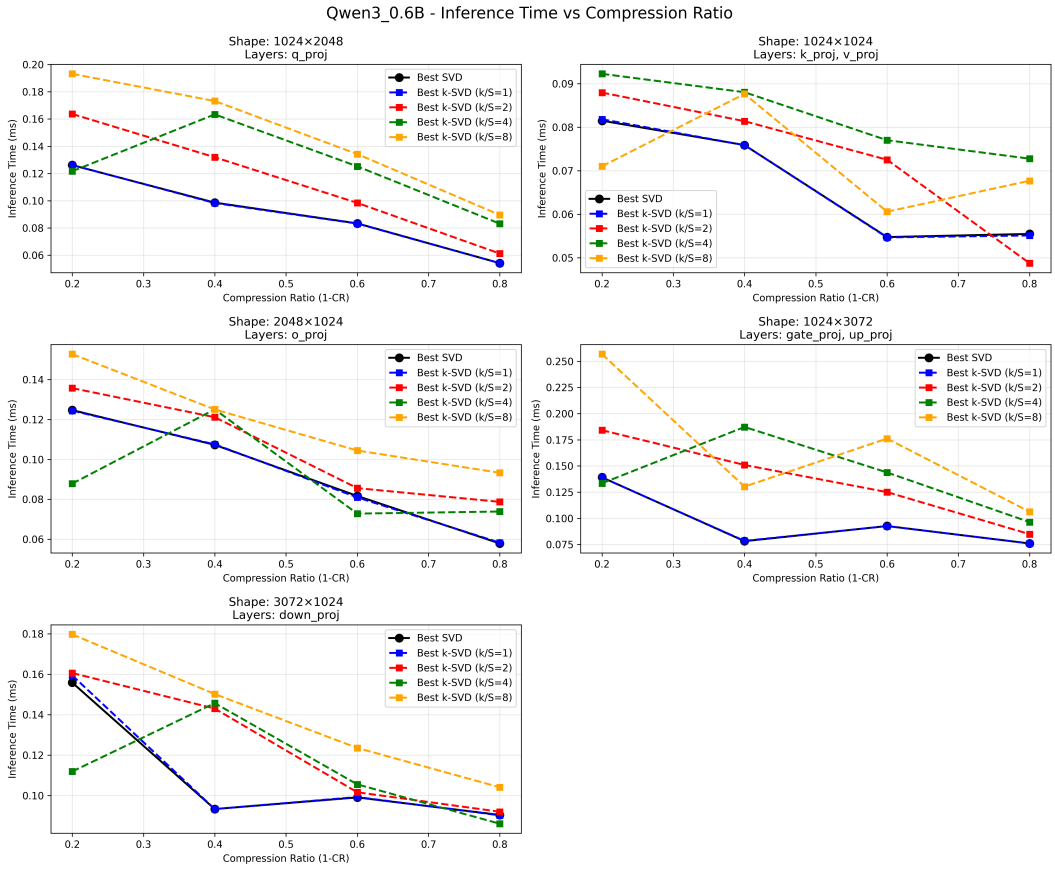


Figure 6: Inference time for different projection layers of Qwen3 0.6B for different compression ratios and k/s ratios on A100

A.8 K-SVD and Power Iteration Analysis

We conducted ablation studies to assess the effect of the number of K-SVD iterations and power iterations on performance using Llama3.2-1B with fixed $\rho = 2$. The left plot in Fig. 7 shows that average accuracy stabilizes after roughly 50 K-SVD iterations, while perplexity continues to decrease slightly before flattening out. The right plot of Fig. 7 indicates that very few power iterations are sufficient for stable convergence: performance improves sharply up to around 5 iterations, after which additional iterations yield minimal benefit. Based on these results, we fixed the number of K-SVD iterations to 60 and power iterations to 8 in our final experiments, which provides a good balance between accuracy, perplexity, and computational efficiency.

A.9 Use of Large Language Models

The authors acknowledge the use of a large language model (LLM) solely for language editing and grammatical refinement of the current manuscript. All scientific content, analysis, and interpretations presented herein are the sole responsibility of the authors.

Table 5: Comparison of the proposed CoSpaDi method with other training-free methods including ASVD Yuan et al. (2023) and state-of-the-art structured pruning methods ReplaceMe Shopkhoev et al. (2025) and LLM-Pruner Ma et al. (2023) on Llama3 8B under different compression ratios. We report accuracy on different benchmarks as well as its average and perplexity. Best results are highlighted with **bold**

Method	CR	Accuracy↑									Perplexity↓	
		PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	Wiki Text	LAMBADA
Llama3 8B		0.8069	0.7913	0.7557	0.7769	0.5350	0.9390	0.4029	0.6215	0.7036	7.26E+00	3.09E+00
ASVD	0.2	0.5577	0.2666	0.0070	0.3237	0.2150	0.3050	0.2182	0.2347	0.2660	9.38E+04	9.89E+05
ReplaceMe	0.22	0.7307	0.6572	0.4211	0.6587	0.4369	0.8640	0.3541	0.5167	0.5799	3.37E+01	2.01E+01
LLM-Pruner	0.2	0.7546	0.6746	0.5096	0.6208	0.3660	0.8780	0.3512	0.2495	0.5505	**1.60E+01**	1.05E+01
CoSpaDi		0.7519	0.6649	0.7380	0.6654	0.4155	0.8950	0.3818	0.4282	**0.6176**	1.97E+01	**4.27E+00**
ASVD	0.3	0.5359	0.2595	0.0000	0.2694	0.2142	0.2190	0.2115	0.2349	0.2431	3.43E+05	1.35E+07
ReplaceMe	0.31	0.6659	0.5382	0.2401	0.5067	0.3788	0.7730	0.3397	0.3058	0.4685	6.67E+01	1.33E+02
LLM-Pruner	0.3	0.6725	0.4509	0.2090	0.4541	0.2875	0.6340	0.3014	0.2292	0.4048	**3.79E+01**	2.21E+02
CoSpaDi		0.7051	0.5620	0.6128	0.5421	0.3353	0.8570	0.3617	0.3224	**0.5373**	4.46E+01	**9.18E+00**
ASVD	0.4	0.5365	0.2575	0.0000	0.2504	0.2287	0.2080	0.2201	0.2431	0.2430	2.05E+05	2.57E+07
ReplaceMe	0.41	0.6170	0.4430	0.0976	0.3742	0.2747	0.6040	0.3158	0.2638	0.3738	2.30E+02	1.76E+03
LLM-Pruner	0.4	0.5033	0.2580	0.0151	0.2643	0.2577	0.2810	0.2182	0.2324	0.2537	∞	5.67E+05
CoSpaDi		0.6371	0.4138	0.3029	0.3914	0.2662	0.6850	0.3053	0.2538	**0.4069**	**1.84E+02**	**1.19E+02**
ASVD	0.5	0.5196	0.2547	0.0000	0.2601	0.2261	0.1980	0.2134	0.2554	0.2409	1.72E+06	1.72E+07
ReplaceMe		0.5724	0.3635	0.0367	0.2976	0.2543	0.3810	0.2785	0.2300	**0.3017**	6.99E+02	5.03E+04
LLM-Pruner		0.5038	0.2623	0.0043	0.2630	0.2637	0.2560	0.2172	0.2371	0.2509	1.24E+03	1.23E+06
CoSpaDi		0.5642	0.3102	0.0390	0.2988	0.2218	0.3770	0.2287	0.2301	0.2837	**8.10E+02**	**7.64E+03**

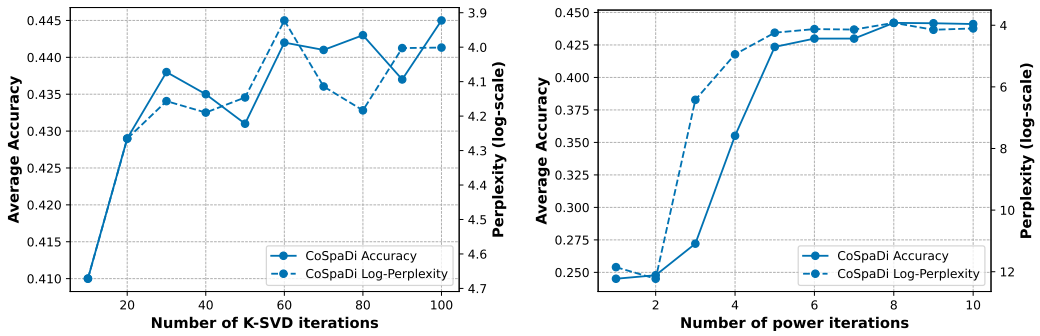


Figure 7: Average benchmark accuracy and WikiText perplexity with respect to the number of K-SVD iterations (left) and the number of power iterations (right) for Llama3.2-1B with $\rho = 2$

Table 6: Compression configurations for Llama2 7B weight matrices across varying compression rates (20%–50%). Each row specifies sparsity parameters (k, s, k/s ratio) and weight dimensions (d_1, d_2) for different weight types (Query, Key, Value, Up, Gate, Down, Out), grouped by compression rate and group size (1 or 2).

Weight Type	**Compression Rate**	**Group size**	**Dictionary,** k	**Sparsity,** s	k/s **ratio**	$\mathbf{d}_1$	$\mathbf{d}_2$
Query	20%	2	3276	1638	2	4096	8192
Key			3276	1638	2	4096	8192
Value			3276	1638	2	4096	8192
Up			4776	2388	2	4096	22016
Gate			4776	2388	2	4096	22016
Down		1	2762	1381	2	11008	4096
Out			2184	1092	2	4096	4096
Query	30%	2	2866	1433	2	4096	8192
Key			2866	1433	2	4096	8192
Value			2866	1433	2	4096	8192
Up			4178	2089	2	4096	22016
Gate			4178	2089	2	4096	22016
Down		1	2416	1208	2	11008	4096
Out			1910	955	2	4096	4096
Query	40%	2	2456	1228	2	4096	8192
Key			2456	1228	2	4096	8192
Value			2456	1228	2	4096	8192
Up			3582	1791	2	4096	22016
Gate			3582	1791	2	4096	22016
Down		1	2072	1036	2	11008	4096
Out			1638	819	2	4096	4096
Query	50%	2	2048	1024	2	4096	8192
Key			2048	1024	2	4096	8192
Value			2048	1024	2	4096	8192
Up			2984	1492	2	4096	22016
Gate			2984	1492	2	4096	22016
Down		1	1726	863	2	11008	4096
Out			1364	682	2	4096	4096