CoSpaDi: Compressing LLMs via Calibration-Guided Sparse Dictionary Learning

Anonymous ACL submission

Abstract

Post-training LLM compression often relies on low-rank approximations, which force all columns of a projection matrix to share a single low-dimensional subspace. We propose CoSpaDi, a training-free compression framework that replaces this single-subspace assumption with a union-of-subspaces model via sparse dictionary learning. CoSpaDi factorizes each weight matrix into a dense dictionary and column-sparse coefficients, allowing different columns to select different subsets of dictionary atoms at the same storage budget. To preserve model behavior, we use calibration activations to transform functional reconstruction into a standard dictionary learning problem. Across Llama and Qwen models, CoSpaDi improves accuracy–compression and perplexity–compression trade-offs over SVD-based and structured pruning baselines at 20–40% compression ratios, while naturally supporting sparse–dense computation and post-training quantization of sparse coefficients.

1 Introduction

Large language models (LLMs) achieve strong performance across diverse tasks, from dialogue and instruction following (Brown et al., 2020; Achiam et al., 2023) to general-purpose reasoning (Touvron et al., 2023; Anil et al., 2023). Their effectiveness stems in part from transformer architectures that model long-range dependencies with attention (Vaswani et al., 2017; Devlin et al., 2019). At the same time, the scale that enables these capabilities makes LLMs expensive to store and run, creating a practical barrier to deployment on memory- and compute-constrained hardware.

A broad literature addresses post-training LLM compression and acceleration, spanning pruning, quantization, knowledge distillation, and matrix factorization (Frankle and Carbin, 2019; Dettmers et al., 2022; Hinton et al., 2015; Denton et al., 2014). Among training-free approaches, matrix factorization is particularly attractive because it yields explicit low-parameter surrogates for large projection matrices. In practice, the predominant choice is truncated SVD and its activation-aware variants, which use a small calibration set to guide which components to keep and how to scale them (Chen et al., 2021). Cross-layer extensions further reduce overhead by sharing a common low-dimensional subspace across groups of layers (Wang et al., 2025a). Despite strong results, these methods ultimately approximate each matrix within a *single* shared low-dimensional subspace; for heterogeneous transformer projections, this constraint can be unnecessarily restrictive and motivates richer factorizations.

In this work, we study an alternative factorization family that relaxes the shared-subspace constraint. Instead of approximating a matrix with one global low-dimensional basis, we model it using a *union of subspaces* via sparse dictionary learning (Aharon et al., 2006; Elad, 2010): a learned dictionary of atoms is combined with column-sparse coefficients, allowing different columns to be reconstructed from different subsets of atoms. This representation is more flexible than a single shared subspace at the same parameter budget, and it is well aligned with the intuition that different output channels may depend on different latent features.

This sparse parameterization is also timely from a systems perspective. Recent LLM pruning methods show that substantial sparsity can be induced post-training while preserving accuracy (Frantar and Alistarh, 2023; Sun et al., 2024), inference systems increasingly exploit activation and neuron sparsity for efficient serving (Song et al., 2024), and specialized sparse kernels have begun to target the moderate-sparsity regime relevant to LLM inference (Macko and Boža, 2025). In parallel, modern accelerator and software stacks expose increasingly mature support for sparse matrix computation, in-

cluding semi-structured sparsity and sparse tensor abstractions (Mishra et al., 2021; NVIDIA, 2020; PyTorch, 2024). CoSpaDi therefore targets not only a richer approximation class, but also a representation whose coefficient sparsity can be exploited by sparse–dense computation.

Building on this idea, we propose **CoSpaDi** (**Co**mpression via **Spa**rse **Di**ctionary Learning), a training-free compression framework for transformer projections. CoSpaDi learns dictionaries and sparse codes to approximate pretrained weight matrices, and is *data-aware*: from a small calibration set, we construct an activation-derived Gram orthonormalization that reformulates functional output reconstruction into a standard dictionary learning problem on transformed activation-weighted weights. The resulting factorization yields structured sparsity that can be paired with post-training quantization of the sparse coefficients, and naturally supports *cross-layer* dictionary sharing for groups of related projections.

Overall our **contributions are three-fold**. *(i)* We introduce sparse dictionary learning as a compression paradigm for LLM weight matrices, replacing the single-subspace constraint of SVD-based factorization with a union-of-subspaces representation. *(ii)* We integrate sparse dictionary learning with a data-aware objective via activation-derived Gram orthonormalization, yielding a tractable transformed problem that can be optimized with alternating sparse coding and dictionary updates, without gradient-based fine-tuning. *(iii)* Across multiple Llama and Qwen models and a range of compression ratios, CoSpaDi improves the quality–compression trade-off over strong activation-aware SVD baselines in both per-layer and grouped scenarios, and is competitive with recent structured pruning methods.

2 Related Work

Low-rank factorization for LLM compression. Post-training matrix factorization compresses large neural weights by replacing them with low-parameter surrogates (Denton et al., 2014). In LLMs, this direction is largely dominated by truncated SVD, which approximates each projection matrix with a rank-r surrogate. Since direct weight-space reconstruction can poorly reflect model behavior, methods such as DRONE (Chen et al., 2021), Fisher-weighted reconstruction (FWSVD) (Hsu et al., 2022), and ASVD (Yuan et al., 2023) use calibration data or sensitivity estimates to better preserve layer outputs. More recent SVD-based methods improve this pipeline through truncation-aware whitening and dynamic budget allocation across layers (Wang et al., 2025d,c,b). Orthogonally, cross-layer approaches such as Basis Sharing reduce storage by reusing low-rank factors across groups of layers (Wang et al., 2025a). Despite these advances, SVD-based compression retains a common modeling assumption: each matrix is approximated within a *single* shared low-dimensional subspace, possibly shared across layers. CoSpaDi targets the same post-training setting, but replaces this single-subspace model with a union-of-subspaces representation.

Sparse dictionary learning and structured factorizations. Dictionary learning offers a classical alternative to low-rank subspace models (Engan et al., 1999; Aharon et al., 2006; Mairal et al., 2009; Gregor and LeCun, 2010; Elad, 2010). Instead of expressing all vectors in one shared basis, it learns a dictionary and represents each vector using only a small subset of atoms, yielding a union-of-subspaces model. Beyond sparse coding itself, neural compression has also explored other structured factorization schemes, including block-wise low-rank approximation (Chen et al., 2018) and tensorized transformer parameterizations (Ma et al., 2019). Recent transformer applications also use learned dictionaries for other components, such as KV-cache compression via sparse decoding (Kim et al., 2025) and cross-layer sharing specialized to attention weights (Zhussip et al., 2025). In contrast, CoSpaDi develops a training-free, calibration-guided dictionary learning framework for LLM *weight* compression, with both per-layer compression and cross-layer dictionary sharing.

Pruning and quantization. Pruning and quantization are complementary post-training compression axes. One-shot pruning methods such as SparseGPT (Frantar and Alistarh, 2023) and Wanda (Sun et al., 2024) use calibration data to induce sparsity while preserving layer outputs, while structured pruning removes or replaces higher-level components such as channels, heads, or blocks (Ma et al., 2023; Shopkhoev et al., 2025). At the systems level, the practical benefit of sparsity depends on metadata overhead, memory access patterns, sparsity structure, and kernel or hardware support (Han et al., 2015; Wang, 2020; Mishra et al., 2021; NVIDIA, 2020). CoSpaDi is related to this line in that it introduces sparsity in the co-

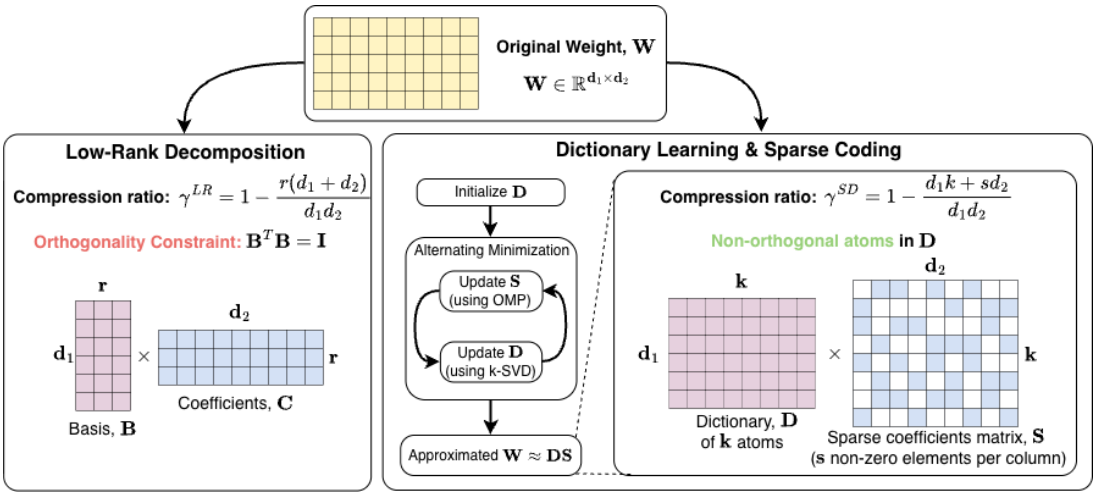

Figure 1: *Left:* low-rank factorization represents $\mathbf{W}$ in a single r-dimensional subspace via two dense factors. *Right:* CoSpaDi represents $\mathbf{W}$ as $\mathbf{DS}$ where $\mathbf{D}$ is a dictionary of k atoms and $\mathbf{S}$ is column-sparse (at most s nonzeros per column), yielding a union-of-subspaces model. Dictionaries may be undercomplete ($k < d_1$), complete ($k = d_1$), or overcomplete ($k > d_1$).

efficient matrix of a learned factorization, so its runtime benefits are subject to similar implementation considerations. Quantization reduces numerical precision and addresses outliers or activation imbalance through Hessian-aware reconstruction, activation-aware scaling, sparse outlier handling, or rotations (Dettmers et al., 2022; Frantar et al., 2023a,b; Lin et al., 2024; Dettmers et al., 2024; Xiao et al., 2023; Ashkboos et al., 2024). These techniques are orthogonal to CoSpaDi, and can be applied post hoc to the sparse coefficients matrix.

3 Method

CoSpaDi compresses pretrained transformer projections by replacing each dense weight matrix with a dense dictionary and sparse column-wise coefficients. We first formulate the activation-space objective used for calibration-guided compression, then reinterpret SVD as a basis–coefficient model, and finally introduce sparse dictionary learning as a union-of-subspaces alternative.

3.1 Activation-space reconstruction

Consider a pretrained projection $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ with calibration activations $\mathbf{X} \in \mathbb{R}^{N \times d_1}$ and outputs $\mathbf{XW}$. Post-training compression replaces $\mathbf{W}$ by a structured approximation $\tilde{\mathbf{W}} \in \mathcal{C}$, where $\mathcal{C}$ denotes the chosen compressed family. Instead of minimizing weight-space error $\|\mathbf{W} - \tilde{\mathbf{W}}\|_F^2$, we preserve the induced layer outputs:

$$\tilde{\mathbf{W}}^\star = \arg\min_{\tilde{\mathbf{W}} \in \mathcal{C}} \|\mathbf{XW} - \mathbf{X}\tilde{\mathbf{W}}\|_F^2. \tag{1}$$

This objective is used implicitly or explicitly in many calibration-guided post-training methods and motivates the activation-aware design of CoSpaDi.

To remove explicit activations from the optimization, let $\mathbf{\Delta} = \mathbf{W} - \tilde{\mathbf{W}}$ and $\mathbf{G} = \mathbf{X}^\mathsf{T}\mathbf{X}$ be the uncentered activation Gram matrix. We compute a numerically stabilized Gram factor $\mathbf{L}$ such that $\mathbf{L}^\top\mathbf{L}$ approximates $\mathbf{G}$ (See A.1 for details). Then

$$\begin{aligned} \|\mathbf{X\Delta}\|_F^2 &= \operatorname{tr}\left(\mathbf{\Delta}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X\Delta}\right) \\ &= \operatorname{tr}\left(\mathbf{\Delta}^\mathsf{T}\mathbf{G\Delta}\right) \approx \|\mathbf{L\Delta}\|_F^2. \end{aligned} \tag{2}$$

Thus, Eq. (1) reduces to Frobenius reconstruction of the transformed weights $\mathbf{W}_L = \mathbf{LW}$.

3.2 From SVD to sparse dictionary learning

Low-rank compression restricts the approximation to matrices of rank at most r:

$$\tilde{\mathbf{W}}_L^{\mathrm{LR}} = \arg\min_{\operatorname{rank}(\tilde{\mathbf{W}}_L) \le r} \|\mathbf{W}_L - \tilde{\mathbf{W}}_L\|_F^2. \tag{3}$$

By the Eckart–Young–Mirsky theorem (Eckart and Young, 1936), the solution is the truncated SVD $\tilde{\mathbf{W}}_L^{\mathrm{LR}} = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^\mathsf{T}$, where $(\mathbf{U}_r, \mathbf{\Sigma}_r, \mathbf{V}_r)$ are the top-r singular components of $\mathbf{W}_L$.

This solution also admits the standard PCA projection interpretation (Bishop and Nasrabadi, 2006). Viewing the columns of $\mathbf{W}_L$ as vectors in $\mathbb{R}^{d_1}$, SVD searches for an r-dimensional subspace that minimizes their reconstruction error. Equivalently, it solves the basis–coefficient problem

$$\min_{\mathbf{B},\mathbf{C}} \|\mathbf{W}_L - \mathbf{BC}\|_F^2 \quad \text{s.t.} \quad \mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}, \tag{4}$$

with optimum $\mathbf{B}^\star = \mathbf{U}_r$ and $\mathbf{C}^\star = \mathbf{\Sigma}_r\mathbf{V}_r^\mathsf{T}$; see Appendix A.2. Thus, truncated SVD is a basis–

coefficient model in which every column is reconstructed in the same shared subspace $\mathrm{span}(\mathbf{B})$, while only its dense coordinates vary.

Sparse dictionary learning keeps the basis–coefficient form but changes both modeling constraints. The orthonormal basis $\mathbf{B}$ is replaced by a generally non-orthogonal dictionary $\mathbf{D}_L \in \mathbb{R}^{d_1 \times k}$, and the dense coefficient matrix $\mathbf{C}$ is replaced by a column-sparse coefficient matrix $\mathbf{S} \in \mathbb{R}^{k \times d_2}$:

$$\min_{\mathbf{D}_L, \mathbf{S}} \|\mathbf{W}_L - \mathbf{D}_L \mathbf{S}\|_F^2 \quad \text{s.t.} \quad \|\mathbf{s}_j\|_0 \le s, \quad \forall j. \tag{5}$$

For column j, the support $\mathcal{T}_j = \mathrm{supp}(\mathbf{s}_j)$ selects the active atoms (columns), so the weight column is reconstructed in $\mathrm{span}(\mathbf{D}_{L,\mathcal{T}_j})$. Since different columns may choose different supports, CoSpaDi replaces the single shared SVD subspace with a union of at most s-dimensional subspaces. After solving Eq. (5), we map back to the original parameterization:

$$\tilde{\mathbf{W}} = \mathbf{D}_a \mathbf{S}, \qquad \mathbf{D}_a = \mathbf{L}^{-1} \mathbf{D}_L. \tag{6}$$

3.3 Optimization and grouped sharing

The objective in Eq. (5) is non-convex due to the bilinear factorization and sparsity constraints. We optimize it by alternating between sparse coding and dictionary update. With $\mathbf{D}_L$ fixed, each column is encoded independently:

$$\mathbf{s}_j \leftarrow \arg\min_{\|\mathbf{s}\|_0 \le s} \|\mathbf{w}_{L,j} - \mathbf{D}_L \mathbf{s}\|_2^2, \tag{7}$$

which we approximate with batched orthogonal matching pursuit (OMP) (Tropp and Gilbert, 2007; Rubinstein et al., 2008). With $\mathbf{S}$ fixed, the dictionary is updated by

$$\mathbf{D}_L \leftarrow \arg\min_{\mathbf{D}} \|\mathbf{W}_L - \mathbf{D}\mathbf{S}\|_F^2. \tag{8}$$

This step can be performed by MOD, which updates all atoms jointly (Engan et al., 1999), or by K-SVD, which updates atoms sequentially via rank-one residual approximations (Aharon et al., 2006). The full procedure is summarized in A.3.

CoSpaDi also supports sharing one dictionary across related projections (see A.4). For a group $\mathcal{G} = \{\ell_1, \ldots, \ell_m\}$, we concatenate weights as $\mathbf{W}_\mathcal{G} = [\mathbf{W}_{\ell_1}, \ldots, \mathbf{W}_{\ell_m}]$ and vertically stack calibration inputs to form $\mathbf{X}_\mathcal{G}$. We compute the Gram factor from $\mathbf{X}_\mathcal{G}$, solve Eq. (5) on $\mathbf{W}_{\mathcal{G},L} = \mathbf{L}\mathbf{W}_\mathcal{G}$, and then slice the learned coefficient matrix into layer-specific blocks. This amortizes the dictionary cost across layers while preserving layer-specific sparse coefficients, following the motivation of cross-layer basis sharing (Wang et al., 2025a).

3.4 Storage and inference cost

We report the compression ratio as storage reduction relative to the dense matrix. For low-rank factorization, storing two dense factors costs $r(d_1 + d_2)$ values:

$$\gamma^{\mathrm{LR}} = 1 - \frac{r(d_1 + d_2)}{d_1 d_2}. \tag{9}$$

For CoSpaDi, the dictionary costs $d_1 k$ values and the sparse coefficients contain sd_2 nonzero values. With $\rho = k/s$, a packed binary mask for the coefficient support costs $kd_2 = \rho s d_2$ bits, or $\rho s d_2/16$ bfloat16 words:

$$\gamma^{\mathrm{SD}}_{\mathrm{mask}} = 1 - \frac{d_1 k + sd_2 + \frac{\rho s d_2}{16}}{d_1 d_2}. \tag{10}$$

In the main experiments, we absorb this mask cost by storing each nonzero coefficient with ρ fewer bits: saving ρ bits for each of the sd_2 coefficients exactly offsets the $\rho s d_2$ mask bits. This gives the effective accounting

$$\gamma^{\mathrm{SD}} = 1 - \frac{d_1 k + sd_2}{d_1 d_2}. \tag{11}$$

Thus, once the target compression ratio γ^{SD} and allocation ratio $\rho = k/s$ are fixed, both the dictionary size k and sparsity level s are uniquely determined by the storage budget.

For N input tokens, dense inference costs Nd_1d_2 multiplications, while low-rank inference costs $Nr(d_1 + d_2)$. CoSpaDi evaluates $\mathbf{X}\tilde{\mathbf{W}} = (\mathbf{X}\mathbf{D}_a)\mathbf{S}$; with reuse of inner products over the active dictionary atoms, its sparse cost is $Nd_1K_{\mathrm{active}} + Nsd_2$, where K_{active} is the number of dictionary atoms used by at least one column. Actual latency depends on the sparsity pattern, mask handling, memory traffic, and sparse–dense kernel efficiency. We provide the full storage accounting, theoretical complexity derivations, and empirical inference measurements in A.5 and A.6.

4 Experiments

We evaluate CoSpaDi in two regimes: **per-layer** compression, where each projection matrix is compressed independently, and **grouped (cross-layer)** compression, where a dictionary is shared across a set of layers of the same type. We first present

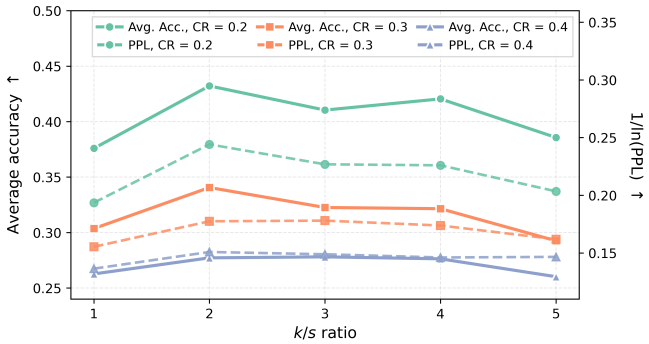

Figure 2: Dual-axis plot showing average accuracy (—— solid lines, left axis) and WikiText perplexity (- - - dashed lines, right axis, inverted logarithmic scale) as functions of ρ for Llama3.2-1B under three CRs: 0.2, 0.3 and 0.4.

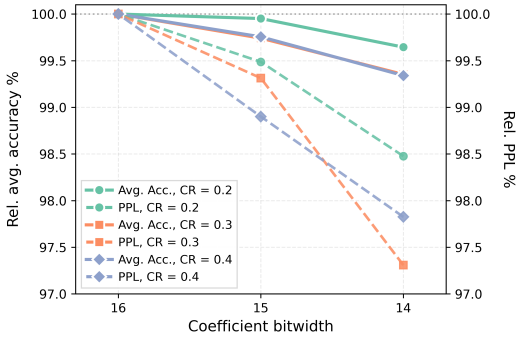

Figure 3: Dual-axis plot showing relative average accuracy (—— solid lines, left axis) and relative WikiText perplexity (- - - dashed lines, right axis) as functions of **S** bitwidth for Llama3-8B under three CRs: 0.2, 0.3 and 0.4.

Method	Data	CR	Avg. Acc.↑	Perplexity↓	
				Wiki Text	LAMBADA
Llama3.2 1B	–	–	57.6	11.6E+00	5.7E+00
SVD	✗	0.2	23.9	2.9E+06	4.6E+06
CoSpaDi†	✗		**24.5**	**3.3E+05**	**2.2E+06**
SVD-LLM	✓		37.6	1.7E+02	1.7E+02
CoSpaDi	✓		**42.7**	**6.4E+01**	**3.5E+01**
SVD	✗	0.3	24.3	1.1E+06	**3.9E+06**
CoSpaDi†	✗		**24.5**	**2.1E+05**	4.3E+06
SVD-LLM	✓		30.2	5.9E+02	2.5E+03
CoSpaDi	✓		**33.7**	**2.9E+02**	**6.6E+02**
SVD	✗	0.4	24.1	**1.2E+06**	**4.2E+06**
CoSpaDi†	✗		**24.7**	3.1E+06	3.7E+07
SVD-LLM	✓		26.2	1.6E+03	3.3E+04
CoSpaDi	✓		**27.5**	**8.0E+02**	**9.2E+03**

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

Method	Wall-clock time	Avg. Acc.↑	Perplexity↓	
			Wiki Text	LAMBADA
Llama3.2 1B	–	57.6	11.6E+00	5.7E+00
MOD	**222.2**	38.9	1.3E+02	1.1E+02
K-SVD (PCA)	902.1	41.5	9.2E+01	5.4E+01
K-SVD (power)	646.8	**42.7**	**6.4E+01**	**3.5E+01**

Table 2: Solver comparison for dictionary update on LLaMA-3.2-1B at 0.2 CR and fixed $\rho = 2$, 60 alternating minimization iterations. We report wall-clock compression time in minutes using a single A100. Best results are provided in **bold**.

ablations that isolate key design choices (dictionary capacity allocation, data-awareness, packing/quantization, and solver variants), and then report main results for both regimes.

4.1 Experimental Setup

Models and layers. We evaluate per-layer compression on `LLaMA-3.2-1B`, `Qwen-3-0.6B`, `LLaMA-3-8B`, `Qwen-3-8B` and `Qwen-3-14B`. For *SVD-LLM* we used the original code-base[1] with only first step on compression without extra fine-tuning. For grouped compression, we follow the *Basis Sharing*[2] protocol and report results on `LLaMA-2-7B`. Unless stated otherwise, we compress all dense linear projections in self-attention (Q/K/V/O) and gated MLP (up/down/gate), while leaving embeddings and the LM head intact.

Calibration data and metrics. For calibration we randomly sample 256 sequences of length 1024 from RefinedWeb (Penedo et al., 2023). We report standard zero-shot accuracy (normalized when available) on a suite of benchmarks (PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), OpenAI LAMBADA (Paperno et al., 2016), ARC-easy/ARC-challenge (Clark et al., 2018), SciQ (Welbl et al., 2017), Race (Lai et al., 2017), MMLU (Hendrycks et al., 2021a)) and perplexity on WikiText (Merity et al., 2017) and OpenAI LAMBADA. We used lm-evaluation-harness 0.4.8 (Gao et al., 2024) to ensure reproducibility.

Baselines. In the per-layer setting, we compare primarily to SVD-LLM (Wang et al., 2025d), a strong activation-aware SVD baseline, and, where relevant, to plain truncated SVD as its data-free counterpart. This isolates the effect of replacing SVD's single-subspace model with CoSpaDi's union-of-subspaces representation, while treating complementary ideas from SVD-LLM V2 (Wang et al., 2025c) and Dobi-SVD (Wang et al., 2025b), such as dynamic allocation and quantization-specific remapping, as orthogonal (see A.7). We also include comparisons to training-free structured pruning baselines.

Reproducibility details. For reproducibility, we provide a consolidated summary of the implementation details used in the main experiments in A.8,

[1] `https://github.com/AIoT-MLSys-Lab/SVD-LLM`
[2] `https://github.com/TUDa-HWAI/Basis_Sharing`

Method	CR	Accuracy↑									Perplexity↓	
		PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	Wiki Text	LAMBADA
Llama3 8B	–	80.7	79.1	75.6	77.7	53.5	93.9	40.3	62.2	70.4	7.3E+00	3.1E+00
SVD-LLM	0.2	71.1	58.4	59.3	55.5	34.0	86.4	35.5	32.6	54.1	4.1E+01	1.1E+01
CoSpaDi		75.2	66.5	73.8	66.5	41.6	89.5	38.2	42.8	**61.8**	**2.0E+01**	**4.3E+00**
SVD-LLM	0.3	65.8	46.4	38.1	41.9	27.7	70.0	31.8	27.2	43.6	1.5E+02	6.1E+01
CoSpaDi		70.5	56.2	61.3	54.2	33.5	85.7	36.2	32.2	**53.7**	**4.5E+01**	**9.2E+00**
SVD-LLM	0.4	60.3	34.5	11.4	32.4	24.5	44.2	25.7	23.1	32.0	5.5E+02	1.3E+03
CoSpaDi		63.7	41.4	30.3	39.1	26.6	68.5	30.5	25.4	**40.7**	**1.8E+02**	**1.2E+02**
Qwen3 8B	–	77.7	74.9	64.1	80.7	56.7	95.7	40.9	73.0	70.5	1.2E+01	4.6E+00
SVD-LLM	0.2	73.8	63.9	62.2	68.7	45.7	90.1	40.5	54.7	62.5	2.1E+01	6.4E+00
CoSpaDi		76.5	68.0	65.6	72.2	48.9	93.2	40.7	60.8	**65.7**	**1.8E+01**	**4.9E+00**
SVD-LLM	0.3	70.4	55.2	53.8	59.3	37.1	87.2	38.4	44.8	55.8	2.7E+01	1.1E+01
CoSpaDi		72.4	60.5	62.6	63.9	41.2	88.4	39.5	51.3	**59.97**	**2.3E+01**	**6.3E+00**
SVD-LLM	0.4	66.3	44.6	37.9	45.0	28.1	77.3	35.3	29.1	45.4	4.3E+01	3.6E+01
CoSpaDi		68.9	49.0	49.9	49.4	29.9	82.0	36.8	36.6	**50.3**	**3.6E+01**	**1.5E+01**

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

Method	CR	Accuracy↑									Perplexity↓	
		PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	Wiki Text	LAMBADA
Llama3 8B		80.7	79.1	75.6	77.7	53.5	93.9	40.3	62.2	70.4	7.3E+00	3.1E+00
ReplaceMe	0.22	73.1	65.7	42.1	65.9	43.7	86.4	35.4	51.7	58.0	3.4E+01	2.0E+01
LLM-Pruner	0.2	75.5	67.5	51.0	62.1	36.6	87.8	35.1	25.0	55.1	**1.6E+01**	1.1E+01
CoSpaDi		75.2	66.5	73.8	66.5	41.6	89.5	38.2	42.8	**61.8**	2.0E+01	**4.3E+00**
ReplaceMe	0.31	66.6	53.8	24.0	50.7	37.9	77.3	34.0	30.6	46.9	6.7E+01	1.3E+02
LLM-Pruner	0.3	67.3	45.1	20.9	45.4	28.8	63.4	30.1	22.9	40.5	**3.8E+01**	2.2E+02
CoSpaDi		70.5	56.2	61.3	54.2	33.5	85.7	36.2	32.2	**53.7**	4.5E+01	**9.2E+00**
ReplaceMe	0.41	61.7	44.3	9.8	37.4	27.5	60.4	31.6	26.4	37.4	2.3E+02	1.8E+03
LLM-Pruner	0.4	50.3	25.8	1.5	26.4	25.8	28.1	21.8	23.2	25.4	∞	5.7E+05
CoSpaDi		63.7	41.4	30.3	39.1	26.6	68.5	30.5	25.4	**40.7**	**1.8E+02**	**1.2E+02**

Table 4: Comparison of the proposed CoSpaDi method with 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 in **bold**.

including the solver configuration, dictionary initialization, sparse-coding setup, calibration protocol and a table of the main hyperparameters.

4.2 Ablation Studies

Capacity Allocation: the k/s Ratio. At a fixed compression ratio (CR), CoSpaDi admits a family of factorizations parameterized by $\rho := k/s$. Smaller ρ increases per-column expressiveness (larger s) but reduces the dictionary size k; larger ρ increases k but restricts each column to fewer active atoms. We perform a coarse sweep over $\rho \in [1; 5]$. Figure 2 (LLaMA-3.2-1B) shows that $\rho = 2$ consistently provides the best trade-off across all considered CRs in terms of both average accuracy and perplexity, so we fix $\rho = 2$ in all subsequent experiments.

Data-Free vs. Data-Aware Compression A core motivation for dictionary learning is that its union-of-subspaces form can be more expressive than a single low-dimensional subspace *regardless of whether calibration is used*. We therefore compare: (i) **data-free** truncated SVD vs. CoSpaDi† (dictionary learning in weight space, without whitening), and (ii) **data-aware** SVD-LLM vs. CoSpaDi (activation-aware via whitening). Table 1 summarizes results on LLaMA-3.2-1B across CR 0.2–0.4 reporting average accuracy and perplexities. In both regimes, the dictionary-based factorization is consistently stronger than the corresponding low-rank baseline, and activation-aware CoSpaDi yields the best overall trade-off.

Dictionary Learning Solver Variants. The main computational bottleneck in CoSpaDi is the dictionary update: K-SVD updates atoms sequentially, which can be expensive for large projection matrices. We therefore compare several dictionary-update variants, including exact rank-1 updates via truncated PCA, approximate rank-1 updates via power iterations, and MOD, which updates the whole dictionary in a single least-squares step.

Table 2 reports quality and wall-clock compression time on LLaMA-3.2-1B at 0.2 CR, using the same hardware, batching, stopping criteria, and fixed factorization shape (k, s). This isolates the solver-level speed–quality trade-off. We use power-

iteration K-SVD in the main experiments, as it provides the best overall balance between accuracy, perplexity, and compression time. Additional ablations over the number of alternating steps and power iterations are provided in A.9, and A.10 further breaks down the wall-clock cost into sparse coding, dictionary updates, and miscellaneous overhead, confirming that the sequential dictionary-update stage dominates runtime.

4.2.1 Coefficient Quantization

Sparse factorization requires storing both nonzero coefficient values and their locations. We therefore study post-training coefficient quantization by truncating bf16 mantissa bits. As shown in Figure 3, with our default $\rho = k/s = 2$, truncating 2 mantissa bits yields almost negligible degradation across tested CRs. This truncation offsets the packed mask overhead, matching the effective storage accounting in Eq. 11. Unless stated otherwise, we evaluate CoSpaDi with bf16 dictionaries and 14-bit sparse coefficients. Appendix A.11 provides a component-wise memory breakdown.

4.3 Main Results

4.3.1 Per-Layer Compression

We first compress each projection matrix independently and compare CoSpaDi to SVD-LLM over a range of CRs. Table 3 reports results on LLaMA-3-8B and Qwen-3-8B at CR 0.2–0.4, while additional results for smaller and larger models are provided in Appendix A.12. Across model families and compression budgets, CoSpaDi consistently improves both average benchmark accuracy and perplexity at matched CR. This suggests that the union-of-subspaces factorization preserves task-relevant directions more effectively than a single shared low-rank basis. We further report results on modern instruction-following and reasoning benchmarks in Appendix A.13, including IFEval (Zhou et al., 2023), BBH (Suzgun et al., 2022), MATH (Hendrycks et al., 2021b), GPQA(Rein et al., 2024), MUSR (Sprague et al., 2024), and MMLU-Pro (Wang et al., 2024).

To position CoSpaDi beyond structured weight factorization, we also compare against recent training-free structured pruning methods. Specifically, Table 4 includes ReplaceMe (Shopkhoev et al., 2025) and LLM-Pruner (Ma et al., 2023), which reduce parameters by removing or replacing structured components such as blocks or layers. These methods provide a complementary reference point under standard parameter-count compression, since their compressed models can still be evaluated within conventional dense inference pipelines.

Unstructured pruning methods involve additional deployment choices, such as sparse storage format, metadata representation, sparsity pattern, and kernel support. Since these choices can substantially affect the realized memory and latency at a fixed nominal sparsity level, we treat sparse pruning as a separate comparison rather than mixing storage conventions in the main table. For completeness, Appendix A.14 reports a comparison with Wanda (Sun et al., 2024) in a 2:4 semi-structured setting, which provides a hardware-friendly sparse baseline.

4.3.2 Cross-Layer Dictionary Sharing

Next, we evaluate grouped compression where a single dictionary is shared across multiple layers of the same projection type. For a fair and controlled comparison, we adopt the *exact* layer grouping and evaluation protocol of Basis Sharing (Wang et al., 2025a), i.e., we share parameters across the same layer sets and report results under the same compression budgets. Table 5 reports results on LLaMA-2-7B across CR 0.2–0.4. We additionally included per-layer counterparts to show the benefits of grouping strategy and exploiting inter-layer redundancies. To facilitate direct comparison with Basis Sharing, we report unnormalized accuracies in parentheses alongside the normalized results.

At all budgets, CoSpaDi substantially outperforms Basis Sharing as well as per-layer baselines, suggesting that cross-layer sharing is most effective when paired with sparse codes that allow each column (and each layer) to select its own subset of atoms. We note that CoSpaDi does not rely on this particular grouping and could benefit from more adaptive sharing strategies (e.g., data-driven grouping or partially shared dictionaries), which we leave for future work.

4.3.3 End-to-end Throughput

Beyond parameter reduction, practical compression depends on whether the induced computation can be executed efficiently. We therefore evaluate a hardware-aware setting on LLaMA-3-8B, where only the wide MLP expansion projections (`up` and `gate`) are replaced by compressed modules. This setting targets the projection types where CoSpaDi's dense-times-sparse structure is most favorable; A.15 provides full details on the

Method	CR	Accuracy↑ (% (unnorm))									Perplexity↓	
		PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	Wiki Text	LAMBADA
Llama2 7B	–	78.9 (78.0)	76.1 (57.3)	73.8	74.2 (76.0)	45.8 (42.7)	91.4 (93.9)	39.7	40.8	65.1 (62.8)	8.70	3.38
SVD-LLM	0.2	73.1 (72.1)	60.3 (44.4)	63.9	55.0 (62.0)	32.0 (29.1)	79.5 (89.2)	36.6	25.5	53.2 (52.9)	20.18	6.33
CoSpaDi (per-layer)		74.5 (73.6)	64.5 (47.7)	70.0	61.5 (67.2)	35.2 (34.0)	85.4 (91.3)	39.2	28.0	57.3 (56.4)	15.97	4.58
Basis Sharing		71.1 (70.2)	60.0 (43.4)	62.8	60.5 (65.5)	37.8 (33.5)	85.0 (90.8)	34.7	25.0	54.6 (53.2)	15.17	7.03
CoSpaDi (grouped)		75.3 (74.5)	66.3 (48.8)	71.1	68.1 (71.3)	39.3 (37.3)	88.5 (92.2)	38.5	27.0	**59.3 (57.6)**	**11.73**	**4.42**
SVD-LLM	0.3	68.2 (68.0)	52.2 (39.4)	51.6	47.3 (53.6)	28.1 (25.2)	75.9 (84.9)	34.6	23.5	47.7 (47.6)	33.99	13.54
CoSpaDi (per-layer)		71.4 (70.1)	57.1 (42.4)	62.7	53.2 (60.2)	31.1 (27.5)	81.7 (88.3)	36.3	26.7	52.5 (51.8)	23.90	6.75
Basis Sharing		66.5 (65.4)	50.3 (37.6)	53.6	54.2 (58.7)	29.3 (27.2)	81.4 (86.9)	32.4	23.3	48.9 (48.1)	22.21	13.18
CoSpaDi (grouped)		71.0 (68.8)	58.5 (42.3)	64.1	63.5 (67.4)	35.5 (33.4)	87.7 (91.3)	35.8	24.0	**55.0 (53.4)**	**15.43**	**6.51**
SVD-LLM	0.4	63.3 (62.9)	42.9 (33.4)	33.0	37.7 (41.9)	24.7 (21.3)	70.1 (77.0)	32.3	23.0	40.9 (40.6)	86.50	66.91
CoSpaDi (per-layer)		66.0 (65.1)	47.9 (36.9)	46.4	44.4 (48.9)	26.5 (22.7)	76.7 (84.0)	33.4	23.5	45.6 (45.1)	50.46	21.50
Basis Sharing		60.7 (60.3)	41.5 (33.0)	41.0	44.6 (49.1)	26.5 (23.5)	75.4 (82.8)	30.1	23.2	42.9 (42.9)	39.58	36.49
CoSpaDi (grouped)		64.8 (63.4)	48.0 (36.1)	51.8	51.9 (56.7)	29.1 (25.6)	80.5 (88.2)	32.7	23.0	**47.7 (47.2)**	**25.29**	**14.99**

Table 5: Performance comparison of CoSpaDi vs Basis Sharing (Wang et al., 2025a) and per-layer counterparts on Llama2-7B under different CRs on various benchmarks. Best average accuracy and perplexities are highlighted with **bold**. Unnormalized accuracies are shown in parentheses for direct comparison with (Wang et al., 2025a).

Table 6: Selective up/gate compression on LLaMA-3-8B. Throughput is averaged over prompt lengths 1–256 in synchronous generation; higher is better.

Method	Global CR	UG CR	Avg.↑	Wiki.↓	LAMB.↓	Tok/s↑
Dense baseline	–	–	70.4	7.3	3.1	44.0
SVD-LLM^{UG}	0.22	0.40	53.5	56.2	8.5	**53.3**
CoSpaDi^{UG}	0.22	0.40	**61.9**	**25.0**	**2.8**	48.0
SVD-LLM^{UG}	0.27	0.50	43.8	213.0	54.5	**52.5**
CoSpaDi^{UG}	0.27	0.50	**54.8**	**52.0**	**5.8**	50.5

Table 7: Asynchronous serving throughput on LLaMA-3-8B for prompt length 1. SVD-LLM^{UG} and CoSpaDi^{UG} use UG CR $= 0.60$; higher is better.

Method	1 user	4 users	8 users	16 users
Dense baseline	77	290	589	1344
SVD-LLM^{UG}	**99**	**386**	**798**	1332
CoSpaDi^{UG}	96	376	758	**1515**

deployment setup, prompt lengths, and selective-compression protocol. We denote variants that compress only the up and gate projections by the superscript UG.

Table 6 summarizes the quality–throughput trade-off in synchronous generation. At matched global compression ratios, CoSpaDi substantially outperforms SVD-LLM in both average accuracy and perplexity. SVD-LLM remains faster in raw tokens/s under this implementation, but CoSpaDi maintains practical throughput while preserving much stronger model quality. For reference, we include the dense LLaMA-3-8B baseline measured under the same deployment setup.

We also evaluate asynchronous serving in Table 7, reporting prompt-length-1 throughput for the dense model, SVD-LLM^{UG}, and CoSpaDi^{UG}. Both compressed variants improve over the dense baseline at most concurrency levels, while CoSpaDi remains competitive with SVD-LLM and preserves substantially better quality. The full asynchronous breakdown is provided in Appendix A.15.

These results suggest that CoSpaDi's advantage comes from a stronger quality–compression trade-off that remains executable in an end-to-end setting. Since CoSpaDi evaluates projections as $(\mathbf{X}\mathbf{D}_a)\mathbf{S}$, its throughput is directly tied to sparse coefficient computation. As sparse kernels and inference engines continue to improve (Macko and Boža, 2025; Chen et al., 2025; Neural Magic, 2025, 2024), the measured throughput should be viewed as an initial deployment result rather than the systems limit of the proposed factorization.

5 Conclusions

CoSpaDi establishes activation-aware sparse dictionary learning as a new post-training compression paradigm beyond single-subspace low-rank approximation. Our current implementation intentionally uses a simple instantiation with fixed $\rho = k/s$, mostly uniform budgets, and a standard alternating solver. This leaves many complementary advances from low-rank compression—such as dynamic allocation, improved calibration objectives, cross-layer sharing, and quantization-aware storage—directly applicable to CoSpaDi without changing its core union-of-subspaces factorization.

From a systems perspective, CoSpaDi exposes a dense-times-sparse computation pattern whose practical efficiency depends on sparse runtimes and kernel support. Our deployment results in Appendix A.15 show measurable speedups over the dense baseline under hardware-aware choices, while ongoing progress in sparse kernels and inference engines suggests that these results are unlikely to be the final systems limit. Thus, CoSpaDi provides both a stronger compression model today and a flexible foundation for future advances in allocation, quantization, and sparse inference.

Limitations

Solver efficiency and scalability. CoSpaDi relies on alternating minimization with OMP-based sparse coding and K-SVD-style dictionary updates. While this procedure is simple and effective, it is more expensive than closed-form SVD because it repeatedly alternates between sparse coding and per-atom dictionary refinement. Our runtime breakdown shows that the dictionary-update stage is the main bottleneck, reflecting the sequential nature of K-SVD. This limitation is algorithmic rather than inherent to the CoSpaDi objective. Several established acceleration routes could reduce compression time, including joint dictionary updates such as MOD (Engan et al., 1999), efficient batched sparse coding such as Batch-OMP (Rubinstein et al., 2008), online or stochastic dictionary learning (Mairal et al., 2009), and approximate rank-one updates based on randomized SVD or power iterations (Halko et al., 2011; Golub and Van Loan, 2013). Our ablations already show that power-iteration updates provide a favorable accuracy-runtime trade-off.

Budget allocation and hyperparameter selection. In the main experiments, we use a fixed ratio $\rho = k/s$ and uniform compression budgets to isolate the effect of the proposed union-of-subspaces factorization. However, prior low-rank work shows that layer-wise and type-wise sensitivity can substantially improve quality at a fixed global budget. Adapting such allocation strategies to CoSpaDi, for example by choosing (k_ℓ, s_ℓ) per layer, projection type, or shared group is a natural extension. Similarly, more adaptive choices of ρ may improve the trade-off between dictionary capacity and per-column sparsity.

Grouped sharing design. Our grouped variant follows a simple sharing scheme aligned with prior cross-layer factor sharing. Its performance can depend on how layers are clustered and which projections share a dictionary. More adaptive grouping based on activation similarity, weight similarity, projection type, or learned sharing patterns could further improve the grouped setting.

Overall, these limitations reflect the direct instantiation studied in this work rather than fundamental restrictions of the sparse dictionary factorization itself.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Michal Aharon, Michael Elad, and Alfred Bruckstein. 2006. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311–4322.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, and 1 others. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240.

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

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Haoxian Chen, Likang Wu, Ming He, Jianping Fan, and Limin Wang. 2025. Efficient low-rank and sparse approximation and adaptation for large language models.

Patrick Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. 2018. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. *Advances in Neural Information Processing Systems*, 31.

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

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *Preprint*, arXiv:1803.05457.

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

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35:30318–30332.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

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

Michael Elad. 2010. *Sparse and redundant representations: from theory to applications in signal and image processing*. Springer Science & Business Media.

Kjersti Engan, Sven Ole Aase, and John Hakon Husoy. 1999. Frame based signal compression using method of optimal directions (mod). In *1999 IEEE International symposium on circuits and systems (ISCAS)*, volume 4, pages 1–4. IEEE.

Ky Fan. 1949. 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.

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

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pages 10323–10337. PMLR.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023a. Gptq: Accurate post-training quantization for generative pre-trained transformers. *Preprint*, arXiv:2210.17323.

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

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The language model evaluation harness.

Gene H. Golub and Charles F. Van Loan. 2013. *Matrix Computations*, 4 edition. Johns Hopkins University Press.

Karol Gregor and Yann LeCun. 2010. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pages 399–406.

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

Song Han, Huizi Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

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

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

Junhyuck Kim, Jongho Park, Jaewoong Cho, and Dimitris Papailiopoulos. 2025. Lexico: Extreme KV cache compression via sparse coding over universal dictionaries. In *Forty-second International Conference on Machine Learning*.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, Copenhagen, Denmark. Association for Computational Linguistics.

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

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Ming Zhou, and Dawei Song. 2019. A tensorized transformer for language modeling. *Advances in neural information processing systems*, 32.

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

Vladimír Macko and Vladimír Boža. 2025. Macko: Sparse matrix-vector multiplication for low sparsity. *Preprint*, arXiv:2511.13061.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. 2009. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pages 689–696.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. 2021. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*.

Neural Magic. 2024. nm-vllm. https://github.com/neuralmagic/nm-vllm. GitHub repository.

Neural Magic. 2025. DeepSparse. https://github.com/neuralmagic/deepsparse. GitHub repository.

NVIDIA. 2020. Nvidia a100 tensor core gpu architecture. Technical report, NVIDIA. NVIDIA Ampere Architecture Whitepaper.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 1525–1534.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Hamza Alobeidli, Alessandro Cappelli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for falcon LLM: Outperforming curated corpora with web data only. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

PyTorch. 2024. Accelerating neural network training with semi-structured (2:4) sparsity. https://pytorch.org/blog/accelerating-neural-network-training/.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Ron Rubinstein, Michael Zibulevsky, and Michael Elad. 2008. Efficient implementation of the k-svd algorithm using batch orthogonal matching pursuit. *Cs Technion*, 40(8):1–15.

Dmitriy Shopkhoev, Ammar Ali, Magauiya Zhussip, Valentin Malykh, Stamatios Lefkimmiatis, Nikos Komodakis, and Sergey Zagoruyko. 2025. Replaceme: Network simplification via depth pruning and transformer block linearization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, SOSP '24, page 590–606, New York, NY, USA. Association for Computing Machinery.

Zayne Rea Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. 2024. MuSR: Testing the limits of chain-of-thought with multistep soft reasoning. In *The Twelfth International Conference on Learning Representations*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2024. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Joel A. Tropp and Anna C. Gilbert. 2007. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

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

Qinsi Wang, Jinghan Ke, Masayoshi Tomizuka, Yiran Chen, Kurt Keutzer, and Chenfeng Xu. 2025b. Dobi-SVD: Differentiable SVD for LLM compression and some new perspectives. In *The Thirteenth International Conference on Learning Representations*.

Xin Wang, Samiul Alam, Zhongwei Wan, Hui Shen, and Mi Zhang. 2025c. Svd-llm v2: Optimizing singular value truncation for large language model compression. *arXiv preprint arXiv:2503.12340*.

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

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. 2024. MMLU-pro: A more robust and challenging multi-task language understanding benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Ziheng Wang. 2020. Sparsert: Accelerating unstructured sparsity on gpus for deep learning inference. In *Proceedings of the ACM international conference on parallel architectures and compilation techniques*, pages 31–42.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 94–106, Copenhagen, Denmark. Association for Computational Linguistics.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pages 38087–38099. PMLR.

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

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *Preprint*, arXiv:2311.07911.

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

A Appendix

A.1 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 (Chen et al., 2021; Yuan et al., 2023; Wang et al., 2025d) 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 $n = 1\dots, N$, 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:

$$\begin{aligned}\tilde{\mathbf{W}}^* = \arg\min_{\tilde{\mathbf{W}}} \left\|\mathbf{X}\mathbf{W} - \mathbf{X}\tilde{\mathbf{W}}\right\|_F \\ \text{s.t.} \quad \operatorname{rank}\left(\mathbf{X}\tilde{\mathbf{W}}\right) = r.\end{aligned} \tag{12}$$

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. (12) as:

$$\begin{aligned}\tilde{\mathbf{W}}^* = \arg\min_{\tilde{\mathbf{W}}} \left\|\mathbf{Y}\mathbf{L}\mathbf{W} - \mathbf{Y}\mathbf{L}\tilde{\mathbf{W}}\right\|_F \\ \text{s.t.} \quad \operatorname{rank}\left(\mathbf{Y}\mathbf{L}\tilde{\mathbf{W}}\right) = r.\end{aligned} \tag{13}$$

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:

$$\tilde{\mathbf{W}}^* = \arg\min_{\tilde{\mathbf{W}}} \left\| \mathbf{\Gamma} - \mathbf{L}\tilde{\mathbf{W}} \right\|_F \quad \text{s.t.} \quad \operatorname{rank}\left(\mathbf{L}\tilde{\mathbf{W}}\right) = r. \tag{14}$$

where $\mathbf{\Gamma} = \mathbf{L}\mathbf{W}$. Next, we introduce the auxiliary matrix $\tilde{\mathbf{\Gamma}} = \mathbf{L}\tilde{\mathbf{W}}$ and the problem in Eq. (14) becomes:

$$\tilde{\mathbf{\Gamma}}^* = \arg\min_{\tilde{\mathbf{\Gamma}}} \left\| \mathbf{\Gamma} - \tilde{\mathbf{\Gamma}} \right\|_F \quad \text{s.t.} \quad \operatorname{rank}\left(\tilde{\mathbf{\Gamma}}\right) = r. \tag{15}$$

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

To conclude, if $\mathbf{\Gamma}$ admits the singular value decomposition $\mathbf{\Gamma} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$, then the optimal r-rank approximation of $\mathbf{W}$ that minimizes the loss in Eq. (12) can be written in the form:

$$\tilde{\mathbf{W}} = \mathbf{B}\mathbf{C} = \underbrace{\mathbf{L}^{-1}\mathbf{U}_r}_{\mathbf{B}}\underbrace{\mathbf{\Sigma}_r\mathbf{V}_r^\mathsf{T}}_{\mathbf{C}}. \tag{16}$$

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 $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{U}_r^\mathsf{T}\left(\mathbf{L}\mathbf{L}^\mathsf{T}\right)^{-1}\mathbf{U}_r \neq \mathbf{I}$.

Numerically robust Gram factorization. In practice, the empirical Gram matrix $\mathbf{G} = \mathbf{X}^\mathsf{T}\mathbf{X}$ may be ill-conditioned or only positive semidefinite. Following the robust scaling step used in SVD-LLM, we first attempt a Cholesky factorization. If it fails, we compute the smallest eigenvalue $\lambda_{\min}(\mathbf{G})$ and apply the diagonal shift

$$\mathbf{G}_\epsilon = \mathbf{G} + \max\{0, -\lambda_{\min}(\mathbf{G}) + \epsilon\}\mathbf{I}, \tag{17}$$

with a small $\epsilon > 0$, after which Cholesky is applied to $\mathbf{G}_\epsilon$. Up to the transpose convention of Cholesky, this gives $\mathbf{L}^\mathsf{T}\mathbf{L} = \mathbf{G}_\epsilon$. Thus, the transform is exact when $\mathbf{G}$ is positive definite and otherwise corresponds to a numerically stabilized activation-space objective.

A.2 Derivation of the Optimal Pair of Basis and Coefficient Matrices

Let us consider the weight matrix $\mathbf{W}$ as a collection of d_1-dimensional vectors $\mathbf{W} = \left[\mathbf{w}_1, \quad \ldots, \mathbf{w}_{d_2}\right]$, 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:

$$\mathcal{J} = \sum_{j=1}^{d_2} \left\| \mathbf{w}_j - \sum_{i=1}^{r} c_{i,j}\boldsymbol{b}_i \right\|_2^2 \equiv \|\mathbf{W} - \mathbf{B}\mathbf{C}\|_F^2. \tag{18}$$

subject to the orthogonality constraint $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}$, where $\mathbf{B} = \left[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r\right] \in \mathbb{R}^{d_1 \times r}$ 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.

To solve this problem, we first reformulate the objective $\mathcal{J}$ into an equivalent form:

$$\mathcal{J} = \operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{W}\right) - 2\operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{B}\mathbf{C}\right) + \operatorname{tr}\left(\mathbf{C}^\mathsf{T}\mathbf{B}^\mathsf{T}\mathbf{B}\mathbf{C}\right). \tag{19}$$

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

$$\nabla_{\mathbf{C}}\mathcal{J} = 2\mathbf{B}^\mathsf{T}\mathbf{B}\mathbf{C} - 2\mathbf{B}^\mathsf{T}\mathbf{W}. \tag{20}$$

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. (19) and using $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}$, we get:

$$\begin{aligned}\mathcal{J} &= \operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{W}\right) - 2\operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{B}\mathbf{B}^\mathsf{T}\mathbf{W}\right) \\ &\quad + \operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{B}\mathbf{B}^\mathsf{T}\mathbf{B}\mathbf{B}^\mathsf{T}\mathbf{W}\right) \\ &= \operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{W}\right) - \operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{B}\mathbf{B}^\mathsf{T}\mathbf{W}\right) \\ &= \operatorname{tr}\left(\mathbf{W}^\mathsf{T}\mathbf{W}\right) - \operatorname{tr}\left(\mathbf{B}^\mathsf{T}\mathbf{W}\mathbf{W}^\mathsf{T}\mathbf{B}\right).\end{aligned} \tag{21}$$

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

$$\mathbf{B}^\star = \arg\max_{\mathbf{B}} \operatorname{tr}\left(\mathbf{B}^\mathsf{T}\mathbf{W}\mathbf{W}^\mathsf{T}\mathbf{B}\right) \quad \text{s.t.} \quad \mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}. \tag{22}$$

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. (22) 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:

$$\begin{aligned} \mathbf{C}^\star &= (\mathbf{B}^\star)^\mathsf{T}\,\mathbf{W} = \mathbf{U}_r^\mathsf{T}\mathbf{W} \\ &= \mathbf{U}_r^\mathsf{T}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T} = \begin{bmatrix}\mathbf{I}_r & \mathbf{O}\end{bmatrix}\begin{bmatrix}\mathbf{\Sigma}_r\mathbf{V}_r^\mathsf{T} \\ \mathbf{\Sigma}_{d-r}\mathbf{V}_{d-r}^\mathsf{T}\end{bmatrix} \\ &= \mathbf{\Sigma}_r\mathbf{V}_r^\mathsf{T}. \end{aligned} \tag{23}$$

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.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. 11 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

$$\mathbf{s}_j^{(t)} \in \operatorname*{arg\,min}_{\|\mathbf{s}\|_0 \le s} \left\|(\mathbf{W}_L)_{:,j} - \mathbf{D}_L^{(t-1)}\mathbf{s}\right\|_2^2. \tag{24}$$

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:

$$\mathbf{R}_i = \mathbf{W}_L[:, \Omega_i] - \sum_{\ell \neq i} \mathbf{D}_{L,\ell}^{(t-1)}\mathbf{s}_{\ell,\Omega_i}^{(t)}. \tag{25}$$

Update $(\mathbf{D_{L,i}}^{(t)}, \mathbf{s}_{i,\Omega_i}^{(t)})$ 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.5 and Eq. 11.

(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.6; the number of active atoms controls the practical speedup.

A.4 Grouped CoSpaDi details

The exact shared-dictionary objective for a group $\mathcal{G}$ is

$$\begin{aligned} \min_{\mathbf{D},\{\mathbf{S}_\ell\}_{\ell\in\mathcal{G}}} & \sum_{\ell\in\mathcal{G}} \|\mathbf{X}_\ell\mathbf{W}_\ell - \mathbf{X}_\ell\mathbf{D}\mathbf{S}_\ell\|_F^2 \\ \text{s.t.} \quad & \|\mathbf{s}_{\ell,j}\|_0 \le s, \quad \forall \ell \in \mathcal{G},\ \forall j. \end{aligned} \tag{26}$$

Writing $\mathbf{G}_\ell = \mathbf{X}_\ell^\top\mathbf{X}_\ell$, this is equivalently

$$\sum_{\ell\in\mathcal{G}} \mathrm{tr}\left[(\mathbf{W}_\ell - \mathbf{D}\mathbf{S}_\ell)^\top\mathbf{G}_\ell(\mathbf{W}_\ell - \mathbf{D}\mathbf{S}_\ell)\right]. \tag{27}$$

Unlike the per-layer case, different layers induce different Gram metrics. With fixed sparse codes,

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 $\mathbf{Y}^\top \mathbf{Y} = \mathbf{I}_{d_1}$;
`% e.g., via QR:` $\mathbf{X} = \mathbf{Q}\mathbf{R} \Rightarrow \mathbf{L} = \mathbf{R}$
`% e.g., via Cholesky:` $\mathbf{X}^\top \mathbf{X} = \mathbf{C}^\top \mathbf{C} \Rightarrow \mathbf{L} = \mathbf{C}$
$\mathbf{W}_L \leftarrow \mathbf{L}\mathbf{W}$;
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 $\mathbf{R}_i$;
      $\mathbf{d}_{L,i}^{(t)} \leftarrow \mathbf{u}$;
      $\mathbf{s}_{i,\Omega_i}^{(t)} \leftarrow \sigma \cdot \mathbf{v}^\top$;
    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 $\mathbf{D}_a$, $\mathbf{S}^{(T)}$, $\tilde{\mathbf{W}}$;

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.

the exact dictionary update satisfies

$$\sum_{\ell \in \mathcal{G}} \mathbf{G}_\ell \mathbf{D} (\mathbf{S}_\ell \mathbf{S}_\ell^\top) = \sum_{\ell \in \mathcal{G}} \mathbf{G}_\ell \mathbf{W}_\ell \mathbf{S}_\ell^\top, \tag{28}$$

or, after vectorization,

$$\left[\sum_{\ell \in \mathcal{G}} (\mathbf{S}_\ell \mathbf{S}_\ell^\top)^\top \otimes \mathbf{G}_\ell \right] \mathrm{vec}(\mathbf{D}) = \mathrm{vec} \left[\sum_{\ell \in \mathcal{G}} \mathbf{G}_\ell \mathbf{W}_\ell \mathbf{S}_\ell^\top \right]. \tag{29}$$

Solving this system is expensive for LLM-scale projections. We therefore use the shared-metric approximation from Section 3.3, where

$$\bar{\mathbf{G}}_{\mathcal{G}} = \frac{1}{|\mathcal{G}|} \sum_{\ell \in \mathcal{G}} \mathbf{G}_\ell = \bar{\mathbf{L}}^\top \bar{\mathbf{L}}. \tag{30}$$

This replaces Eq. (26) by

$$\min_{\mathbf{D}, \{\mathbf{S}_\ell\}} \sum_{\ell \in \mathcal{G}} \|\bar{\mathbf{L}}(\mathbf{W}_\ell - \mathbf{D}\mathbf{S}_\ell)\|_F^2 \quad \text{s.t.} \quad \|\mathbf{s}_{\ell,j}\|_0 \leq s, \quad \forall \ell \in \mathcal{G},\ \forall j. \tag{31}$$

Defining $\mathbf{D}_L = \bar{\mathbf{L}}\mathbf{D}$ and concatenating transformed weights yields

$$\begin{aligned} &\min_{\mathbf{D}_L, \mathbf{S}_\mathcal{G}} \|\mathbf{W}_{\mathcal{G},L} - \mathbf{D}_L \mathbf{S}_\mathcal{G}\|_F^2 \\ &\text{s.t.} \quad \|\mathbf{s}_{\ell,j}\|_0 \le s, \quad \forall \ell, j. \end{aligned} \tag{32}$$

which has the same form as the per-layer CoSpaDi objective and is solved with the same OMP/K-SVD alternating procedure. After optimization, $\mathbf{D}_a = \bar{\mathbf{L}}^{-1}\mathbf{D}_L$, and each layer is reconstructed as $\tilde{\mathbf{W}}_\ell = \mathbf{D}_a \mathbf{S}_\ell$.

A.5 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

$$\mathbf{W} \approx \mathbf{D}\mathbf{S}, \qquad \mathbf{D} \in \mathbb{R}^{d_1 \times k}, \ \mathbf{S} \in \mathbb{R}^{k \times d_2}, \tag{33}$$

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

$$\underbrace{d_1 k}_{\text{dictionary}} + \underbrace{sd_2}_{\text{values}} + \underbrace{\tfrac{kd_2}{16}}_{\text{mask}} \tag{34}$$

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

$$\gamma^{\mathrm{SD}} := 1 - \frac{\overbrace{d_1 k}^{\text{dict. (bf16)}} + \overbrace{sd_2}^{\text{codes (bf16)}} + \overbrace{(kd_2)/16}^{\text{mask (1 bit/entry)}}}{d_1 d_2}. \tag{35}$$

Mask compensation by coefficient truncation. In our main experiments we use $\rho := k/s = 2$. Therefore, the packed binary mask requires

$$kd_2 = 2sd_2 \tag{36}$$

bits. At the same time, truncating two mantissa bits from each of the sd_2 stored coefficient values saves exactly

$$2sd_2 \tag{37}$$

bits. Thus, for $\rho = 2$, the storage cost of the binary mask is exactly compensated by the two-bit coefficient truncation:

$$\underbrace{14sd_2}_{\text{trunc. coef. values}} + \underbrace{kd_2}_{\text{mask}} \tag{38}$$

$$= 14sd_2 + 2sd_2 = 16sd_2 \quad \text{bits.} \tag{39}$$

Equivalently, the truncated coefficient values together with their packed mask occupy the same storage as sd_2 bf16 values. This gives the effective compression ratio used in our main experiments:

$$\hat{\gamma}^{\mathrm{SD}} = 1 - \frac{d_1 k + sd_2}{d_1 d_2}. \tag{40}$$

This accounting relies on both assumptions: $\rho = 2$ and two-bit coefficient truncation. Without truncation, the mask contributes an additional $(kd_2)/16$ bf16 words; for $\rho \neq 2$, truncating two bits no longer exactly compensates the mask term. This matches the expression used in the main text and makes explicit the dependence on the two design knobs (k, s).

A.6 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 <u>multiplications only</u> (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

$$O_{baseline} = N d_1 d_2. \tag{41}$$

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:

$$O_{LR} = Nd_1 r + Nrd_2 = Nr(d_1 + d_2). \quad (42)$$

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:

$$\begin{aligned} O_{SD,\text{dense}} &= Nd_1 k + Nkd_2 && (43) \\ &= Nk(d_1 + d_2). && (44) \end{aligned}$$

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

$$\begin{aligned} y_{i,j} &= \sum_{k=1}^{K} \mathbf{S}_{k,j} \langle \mathbf{X}_{i,:}, \mathbf{D}_{:,k} \rangle \\ &= \sum_{k \in \mathcal{S}_j} \mathbf{S}_{k,j} \langle \mathbf{X}_{i,:}, \mathbf{D}_{:,k} \rangle . \end{aligned} \quad (45)$$

where $\mathcal{S}_j = \text{supp}(s_j)$ and $|\mathcal{S}_j| = s$. 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 $\mathbf{U} = \bigcup_{j=1}^{d_2} \mathcal{S}_j$ and $K_{active} = |U|$ we have:

$$\begin{aligned} O_{SD,\text{sparse-reuse}} &= Nd_1 K_{\text{active}} + Nsd_2, \\ s \le K_{\text{active}} &\le \min(K, sd_2). \end{aligned} \quad (46)$$

Using the effective storage accounting from Appendix A.5, where $\rho = 2$ and two-bit coefficient truncation compensates the packed mask cost, we have

$$\hat{\gamma}^{\text{SD}} = 1 - \frac{d_1 k + sd_2}{d_1 d_2}. \quad (47)$$

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

$$r = \frac{(1 - \gamma^{LR}) d_1 d_2}{d_1 + d_2} \quad (48)$$

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

$$k = \frac{(1 - \hat{\gamma}^{\text{SD}})\, d_1 d_2}{d_1 + \frac{d_2}{\rho}}, \qquad s = \tfrac{k}{\rho}. \quad (49)$$

Under matched effective storage and using the upper bound $K_{\text{active}} \le k$, the nominal leading multiplication count matches that of low-rank compression:

$$\begin{aligned} O_{SD} &= Nd_1 k + Nsd_2 \\ &= N\left(d_1 k + d_2 \frac{k}{\rho}\right) \\ &= Nk\left(d_1 + \frac{d_2}{\rho}\right). \end{aligned} \quad (50)$$

$$\begin{aligned} k &= \frac{(1 - \hat{\gamma}^{SD}) d_1 d_2}{d_1 + \frac{d_2}{\rho}} \\ \Longrightarrow O_{SD} &= N(1 - \hat{\gamma}^{SD}) d_1 d_2. \end{aligned} \quad (51)$$

$$O_{LR} = O_{SD} = N(1 - \gamma) d_1 d_2. \quad (52)$$

This equality should be interpreted as a storage-aligned complexity comparison rather than a latency prediction. Actual wall-clock performance depends on support overlap, indexing overhead, memory traffic, batching, and sparse-kernel efficiency. For this reason, we report measured end-to-end deployment results separately in Appendix A.15, where the runtime setup, selective `up/gate` compression protocol, and throughput measurements are described explicitly.

A.7 Relationship to Recent SVD-Based Extensions

Our main comparison focuses on isolating the effect of the factorization family: SVD-based compression represents each projection matrix in a single shared low-dimensional subspace, whereas CoSpaDi replaces this assumption with a sparse dictionary and a union-of-subspaces representation. For this reason, we primarily compare against SVD-LLM (Wang et al., 2025d), a strong activation-aware SVD baseline under a comparable post-training setting. More recent methods such as SVD-LLM V2 (Wang et al., 2025c) and Dobi-SVD (Wang et al., 2025b) introduce important advances, but these advances are largely orthogonal to the modeling change studied in this work.

SVD-LLM V2. SVD-LLM V2 extends SVD-LLM mainly through two mechanisms: non-uniform compression-ratio allocation based on theoretical truncation loss, and a loss-optimized truncation procedure. Both mechanisms improve the

allocation and truncation policy within the low-rank SVD family; they do not change the underlying single-subspace factorization class. In contrast, CoSpaDi changes the compressed representation itself, replacing a rank-r surrogate with a sparse dictionary factorization. Therefore, a direct comparison between uniform CoSpaDi and dynamically allocated SVD-LLM V2 would conflate two effects: the factorization family and the budget-allocation rule. The same allocation idea can be transferred to CoSpaDi by choosing layer-wise or group-wise parameters (k_ℓ, s_ℓ) under a global storage budget. We therefore regard dynamic allocation from SVD-LLM V2 as a complementary extension rather than a competing modeling paradigm.

There is also a practical reproducibility concern. The public SVD-LLM repository does not provide a dedicated ready-to-run SVD-LLM V2 implementation, and the V2 description leaves several implementation conventions implicit, including the distinction between reported compression ratios and retained-parameter ratios used for rank computation, as well as numerical details of the truncation-loss computation. For this reason, using a reimplementation of SVD-LLM V2 as a primary baseline risks introducing implementation-dependent confounders. Instead, our main experiments use the reproducible SVD-LLM code path to isolate the effect of replacing low-rank single-subspace compression with sparse dictionary learning.

Dobi-SVD. Dobi-SVD is also not a purely training-free SVD baseline in the same sense as SVD-LLM and CoSpaDi. It optimizes layer-wise ranks through a differentiable objective using backpropagation, which introduces an additional training-based allocation step. This is conceptually different from our setting, where calibration data are used only to construct activation statistics and no gradient-based optimization of the compressed model is performed. The differentiable allocation strategy of Dobi-SVD is nevertheless complementary: an analogous objective could be used to allocate CoSpaDi parameters (k_ℓ, s_ℓ) across layers.

A second distinction is Dobi-SVD's remapping mechanism. When factorization is followed by b-bit quantization, the effective compression ratio combines structural compression and quantization:

$$\gamma_{\text{target}} = 1 - (1 - \gamma_{\text{fact}})\frac{b}{16}. \tag{53}$$

Thus, if remapping increases the number of factorized parameters, the final compression can be driven primarily by quantization rather than by a more compact factorization. This makes remapped Dobi-SVD a hybrid factorization–quantization method, whereas the main goal of CoSpaDi is to study structural compression from a richer sparse dictionary factorization. Since CoSpaDi also supports post-training coefficient quantization, quantization-specific remapping is best viewed as an orthogonal storage technique rather than a direct test of the factorization class.

Summary. SVD-LLM V2 and Dobi-SVD are valuable but address different axes of the compression problem: dynamic allocation, loss-optimized truncation, differentiable rank selection, and quantization/remapping. These ideas can in principle be combined with CoSpaDi without changing its core contribution. Accordingly, our main baseline choice is designed to isolate the central modeling question: whether replacing SVD's single shared subspace with a calibration-guided union-of-subspaces representation improves post-training LLM compression at a matched storage budget.

A.8 Implementation details and reproducibility

This appendix consolidates the main implementation details used in the experiments. Unless otherwise specified, CoSpaDi uses K-SVD with power-iteration-based rank-1 atom updates, with $T = 60$ alternating iterations and 8 power iterations per update. Sparse coding is performed with batched OMP using a mini-batch size of 8192. The dictionary is initialized from a random permutation of columns of the original projection matrix. All experiments use a fixed random seed of 42.

Calibration is performed using 256 sequences of length 1024 sampled from RefinedWeb. Unless stated otherwise, the compressed models use bfloat16 dictionaries and truncated 14-bit coefficients, and the reported compression ratios follow the storage convention described in Section 4.2.1. The summary of the hyperparameters used are listed in Table 8

A.9 Number of Alternating Minimization Steps 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 Figure 4 shows that average accuracy stabilizes after roughly 50 K-SVD

Table 8: Main hyperparameters used in the experiments unless otherwise specified. The dictionary size k and sparsity level s are determined from the target compression ratio and the ratio $\rho = k/s$ via Eq. (11).

Hyperparameter	Value
Random seed	42
K-SVD iterations T	60
Ratio $\rho = k/s$	2.0
Dictionary update solver	power-iteration K-SVD
Power iterations	8
OMP batch size	8192
Target modules	linear layers in transformer blocks
Dictionary size k	determined by CR and ρ via Eq. (11)
Sparsity s	determined by CR and ρ via Eq. (11)
Dictionary initialization	random columns of W
Calibration dataset	RefinedWeb
Number of calibration samples	256
Sequence length	1024
Storage convention for reported CR	Eq. (11) unless stated otherwise

Table 9: Runtime breakdown of CoSpaDi on a 1B-scale model for end-to-end compression of all targeted projection matrices under the current solver configuration.

Component	Time (min)	Share (%)
Sparse coding (OMP)	178.5	27.6
Dictionary update (K-SVD)	467.6	72.3
Misc. overhead	0.7	0.1
Total	646.8	100.0

iterations, while perplexity continues to decrease slightly before flattening out. The right plot of Figure 4 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.10 Runtime Breakdown and Scalability Discussion

The compression runtime of CoSpaDi is driven by the alternating optimization procedure, which combines OMP-based sparse coding with K-SVD-style dictionary updates. To clarify the contribution of the main components, Table 9 reports a runtime breakdown for end-to-end compression of all targeted projection matrices in a 1B-scale model under the current solver configuration.

The results show that the dominant cost comes from the dictionary-update stage, while sparse coding accounts for a smaller but still substantial fraction of the overall runtime. This behavior is consistent with the optimization structure of CoSpaDi, since K-SVD updates atoms sequentially through repeated rank-1 approximation subproblems.

Several acceleration directions are compatible with the current formulation. First, online or stochastic dictionary learning can replace full-batch updates with incremental ones, improving scalability to larger problems (Mairal et al., 2009). Second, implementation-level optimizations such as Batch-OMP and efficient update procedures can substantially reduce wall-clock time in practice (Rubinstein et al., 2008). Third, the rank-1 update step can be approximated using truncated or randomized low-rank routines (Halko et al., 2011), often combined with a small number of power iterations (Golub and Van Loan, 2013), yielding improved accuracy–runtime trade-offs.

These observations suggest that the current compression time is primarily a property of the present solver implementation rather than of the CoSpaDi objective itself. In this work, the focus is on establishing activation-aware sparse dictionary learning as a post-training compression paradigm and evaluating its compression–quality trade-offs; improving compression-time efficiency remains an important direction for future work.

A.11 Detailed storage breakdown

To make the storage accounting more transparent, we provide in Table 10 a component-wise memory breakdown for CoSpaDi on the main settings. Specifically, we separate the contribution of the dictionary, stored coefficient values, and mask/index overhead. This complements the derivation in Appendix A.5 and clarifies the relation between the full accounting of Eq. (10) and the simplified accounting convention of Eq. (11).

The columns "Applied eq." and "Quant. coeff." indicate whether the target compression ratio is computed using the full accounting of Eq. 10 or the simplified convention of Eq. 11, and whether coefficient truncation is applied. Importantly, the table reports the actual component-wise allocation of storage in each case. Thus, while the relative allocation between dictionary and coefficients changes across the two conventions, the total compressed footprint remains essentially unchanged up to rounding.

For additional intuition, consider a single `down_proj` matrix in Qwen-3 8B with dimensions $d_1 = 4096$ and $d_2 = 12288$.

Under Eq. 10, the dictionary has shape (4096×3657), requiring approximately 28.57 MB. The coefficient matrix has shape (3657×12288) with

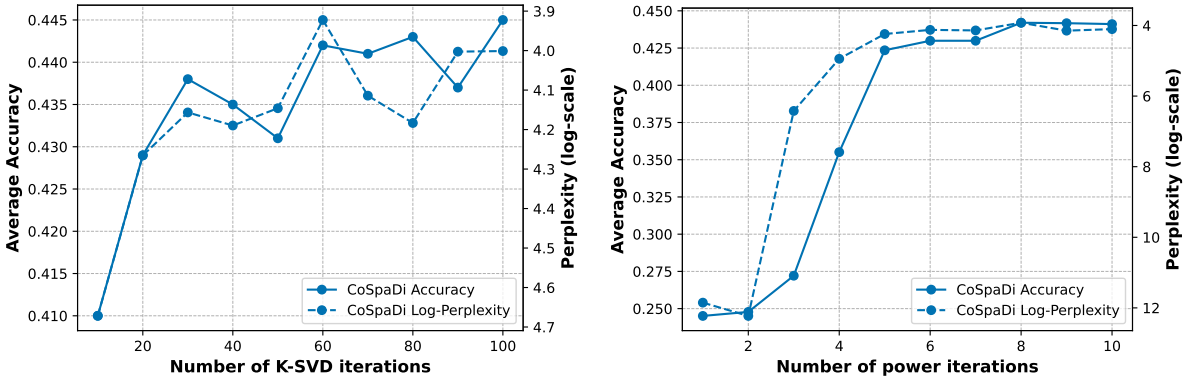

Figure 4: Average benchmark accuracy and WikiText perplexity with respect to the number of K-SVD iterations (left) and the number of power iterations (right) for Llama3.2-1B with $\rho = 2$

Table 10: Memory footprint breakdown (MB) for storing all compressed linear layers (excluding embeddings, `lm_head`, and normalization layers) under CoSpaDi at different compression ratios.

Model / Setting	Applied eq.	Quant. coeff.	Dictionary	Coeff. values	Mask / indices	Full total
Llama-3.2-1B	–	No	–	–	–	1856
CoSpaDi, CR=20%	Eq. 10	No	990	438	54	1484
	Eq. 11	Yes	1018	408	58	1484
CoSpaDi, CR=30%	Eq. 10	No	866	383	47	1298
	Eq. 11	Yes	890	357	51	1298
CoSpaDi, CR=40%	Eq. 10	No	743	329	41	1113
	Eq. 11	Yes	763	306	43	1113
Qwen-3-8B	–	No	–	–	–	13248
CoSpaDi, CR=20%	Eq. 10	No	7005	3192	399	10596
	Eq. 11	Yes	7226	2948	421	10596
CoSpaDi, CR=30%	Eq. 10	No	6129	2793	349	9272
	Eq. 11	Yes	6322	2579	368	9271
CoSpaDi, CR=40%	Eq. 10	No	5253	2393	299	7946
	Eq. 11	Yes	5419	2211	315	7947

1828 nonzero entries per column, yielding approximately 42.84 MB for the stored coefficient values and 5.36 MB for the packed binary mask.

Under Eq. 11, the dictionary has shape (4096×3932), requiring approximately 30.72 MB. The coefficient matrix has shape (3932×12288) with 1966 nonzero entries per column, yielding approximately 40.32 MB for the truncated coefficient values and 5.76 MB for the packed binary mask.

This breakdown makes explicit how much each component contributes to the compressed representation and clarifies that the difference between the two accounting conventions lies primarily in how the storage budget is allocated between dictionary capacity and coefficient precision, rather than in the overall total size.

A.12 Results for LLama-3.2-1B, Qwen-3 0.6B and Qwen-3 14B

Figure 5 summarizes the accuracy–compression and perplexity–compression curves for small models (LLaMA-3.2-1B and Qwen-3-0.6B).

A.13 More Benchmarks

We additionally evaluate CoSpaDi and SVD-LLM on the Qwen model family using a set of more recent reasoning-oriented benchmarks. Table 13 reports results for Qwen3-8B and Qwen3-14B across several compression ratios. Overall, CoSpaDi matches or improves upon the SVD-LLM baseline on most benchmarks and settings, indicating that sparse dictionary learning provides a stronger compression–quality trade-off than low-rank factorization beyond the standard evaluation suite.

A.14 Comparison with semi-structured pruning baseline

To further position CoSpaDi relative to pruning-based compression, we compare it with Wanda (Sun et al., 2024) in a 2:4 semi-structured setting on Llama3 8B. This baseline is particularly relevant from a deployment perspective because it follows a hardware-friendly N:M sparsity pattern.

For the 2:4 baseline, it is important to distinguish

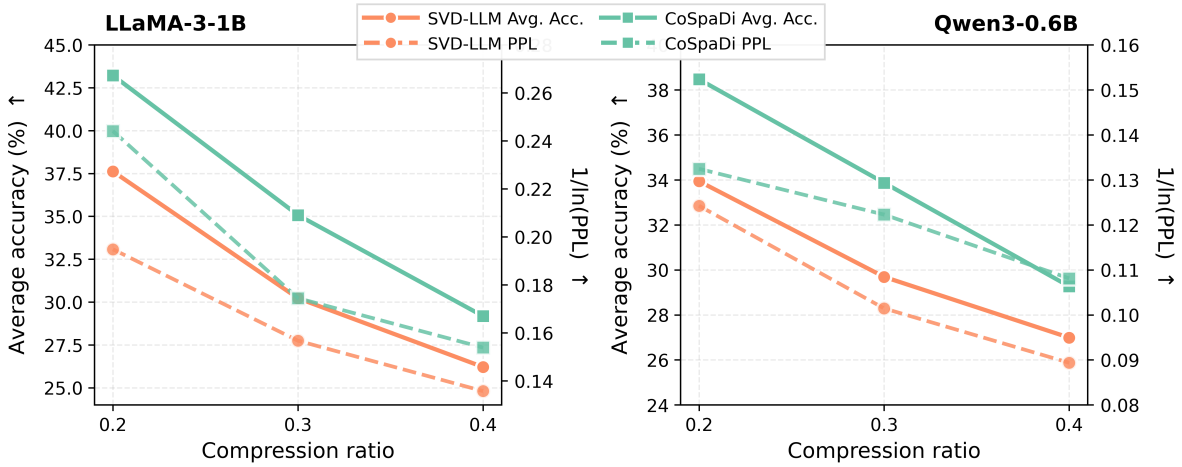

Figure 5: Average benchmark accuracy and WikiText perplexity for LLaMA-3.2-1B and Qwen-3 0.6B using SVD-LLM and CoSpaDi with respect to compression ratio.

Table 11: Performance comparison of CoSpaDi vs SVD-LLM on Llama3-8B, Qwen3-8B and Qwen3-14B 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
Qwen3 14B	–	79.9	78.9	67.9	82.8	60.2	96.5	43.3	77.2	73.3	1.1E+01	3.7E+00
SVD-LLM	0.2	76.2	67.6	69.1	69.8	46.8	91.0	42.9	62.5	65.8	1.8E+01	4.1E+00
CoSpaDi		77.3	71.7	72.0	73.3	51.3	91.9	43.1	65.8	**68.3**	**1.6E+01**	**3.4E+00**
SVD-LLM	0.3	72.0	59.7	61.5	62.6	39.3	88.9	41.5	52.0	59.7	2.3E+01	6.5E+00
CoSpaDi		74.6	64.2	69.0	65.3	43.6	90.7	42.5	58.0	**63.5**	**2.0E+01**	**4.2E+00**
SVD-LLM	0.4	67.4	48.7	46.0	48.5	30.7	80.6	36.7	36.8	49.4	3.5E+01	1.8E+01
CoSpaDi		69.5	52.3	57.3	53.2	33.0	84.4	37.5	43.6	**53.9**	**3.3E+01**	**8.7E+00**

nominal sparsity from effective storage compression. Although 2:4 sparsity removes 50% of the weights, the realized memory reduction is smaller once metadata is taken into account. In the standard semi-structured representation, each retained 16-bit weight requires an additional 2 bits of metadata, yielding an effective storage cost of 9 bits per original weight position and therefore an effective compression ratio of $1 - 9/16 \approx 43.8\%$, rather than the naive 50%. We therefore compare Wanda 2:4 to CoSpaDi at the closest matched storage budget.

Results are reported in Table 12. Overall, CoSpaDi remains competitive with pruning-based alternatives and provides the strongest average accuracy at the matched compression ratio. In particular, CoSpaDi outperforms both SVD-LLM and Wanda 2:4 in average accuracy, while also achieving substantially better perplexity than both baselines on LAMBADA and better perplexity than SVD-LLM on WikiText. More broadly, this comparison highlights that CoSpaDi should be viewed not only as an alternative to low-rank factorization, but also as a sparse reparameterization method that is naturally related to pruning. At the same time, our current formulation is not limited to unstructured sparsity patterns: the same N:M constraints could in principle be imposed directly on the coefficient matrix, allowing each column to activate a hardware-friendly subset of dictionary atoms. We view this as a promising direction for future work.

A.15 From Theory to Practice: Deployment Considerations

While CoSpaDi improves the accuracy–compression trade-off in the uniform compression setting, practical deployment also depends on where compression is applied. Not all projection layers contribute equally to parameter count, accuracy degradation, or runtime. In modern LLMs, MLP expansion projections are substantially wider than attention projections, making the `up` and `gate` matrices especially attractive targets for selective compression. We therefore evaluate a hardware-aware setting in which only these two projection types are replaced by compressed modules. We denote such variants with the superscript UG.

Table 12: Performance comparison on Llama3 8B between SVD-LLM, CoSpaDi, and Wanda 2:4 at compression ratio 0.4. Best compressed result in each column is highlighted in **bold**.

Method	CR	Accuracy↑									Perplexity↓	
		PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	Wiki Text	LAMBADA
Llama3 8B	–	80.7	79.1	75.6	77.7	53.5	93.9	40.3	62.2	70.4	7.3E+00	3.1E+00
SVD-LLM	0.4	60.3	34.5	11.4	32.4	24.5	44.2	25.7	23.1	32.0	5.5E+02	1.3E+03
Wanda 2:4	0.44	62.4	39.7	0.8	41.9	25.6	74.7	26.8	23.4	36.9	2.1E+02	5.7E+03
CoSpaDi	0.4	63.7	41.4	30.3	39.1	26.6	68.5	30.5	25.4	**40.7**	**1.8E+02**	**1.2E+02**

Table 13: Performance comparison of CoSpaDi vs sota SVD-LLM on Qwen3-8B and Qwen3-14B at different compression levels on different benchmarks. Best results are highlighted with **bold**.

Model	CR	IFEval (%)	BBH (%)	MATH (%)	GPQA (%)	MUSR (%)	MMLU-Pro (%)
Qwen 3 8B	—	39.2	60.9	52.6	36.2	43.1	47.7
SVDLLM	0.2	25.5	41.0	1.1	28.4	39.8	26.3
CoSpaDi		**28.9**	**45.3**	**2.0**	**28.6**	**42.1**	**31.5**
SVDLLM	0.3	22.9	34.4	**1.0**	**25.6**	**41.4**	18.8
CoSpaDi		**25.2**	**38.2**	**1.0**	24.8	38.4	**22.8**
SVDLLM	0.4	22.7	30.2	**0.8**	23.1	37.7	11.6
CoSpaDi		**26.1**	**32.7**	**0.8**	**26.0**	**38.1**	**16.7**
Qwen 3 14B	—	43.3	63.2	53.3	38.6	40.7	53.3
SVDLLM	0.2	25.5	50.1	1.6	29.2	**41.5**	32.7
CoSpaDi		**29.6**	**54.2**	**4.2**	**30.6**	40.7	**38.9**
SVDLLM	0.4	23.0	34.3	1.1	**26.4**	**39.7**	12.8
CoSpaDi		**26.6**	**38.3**	**1.2**	25.5	38.0	**15.7**

This selective setting is favorable for CoSpaDi because the wide up/gate matrices are naturally represented as a dense dictionary followed by sparse coefficients. In contrast to uniform compression, selective compression concentrates the storage budget on the layers that offer the largest parameter savings and the most suitable dense-times-sparse computation pattern. For CoSpaDi^{UG}, we use pretrained compressed checkpoints at the corresponding layer-wise compression ratio and replace only the up and gate modules in the dense model. We compare against SVD-LLM^{UG} under the same replacement protocol.

Quality of selective compression. Table 14 reports the resulting accuracy and perplexity on LLaMA-3-8B. Selective CoSpaDi preserves the quality of the uniformly compressed CoSpaDi model at a similar global compression ratio, and substantially outperforms SVD-LLM^{UG} at matched global CR. This indicates that the union-of-subspaces representation remains beneficial even when compression is restricted to deployment-favorable projection types.

Synchronous throughput. We first measure synchronous generation throughput on an RTX 3090 across prompt lengths from 1 to 256 tokens. Table 15 reports tokens/s for SVD-LLM^{UG} and CoSpaDi^{UG} at several up/gate compression ratios. SVD-LLM^{UG} is faster in raw tokens/s under the current implementation, but CoSpaDi^{UG} remains close in throughput while providing substantially stronger model quality at matched global CR.

Asynchronous throughput. We also evaluate asynchronous serving with concurrent users. Table 16 compares the dense model, SVD-LLM^{UG}, and CoSpaDi^{UG} for prompt length 1. Both compressed variants improve throughput over the dense baseline for most concurrency settings, and CoSpaDi^{UG} achieves the highest throughput at the largest tested concurrency.

Table 17 reports the full asynchronous prompt-length sweep for CoSpaDi^{UG}. Throughput remains stable across prompt lengths and scales with the number of concurrent users, indicating that selective CoSpaDi compression is compatible with multi-request serving.

Overall, selective up/gate compression shows that CoSpaDi is not only a stronger quality-preserving compression method, but also a practical deployment candidate. Its current throughput is competitive with a low-rank counterpart while retaining much higher accuracy and lower perplexity at matched global CR. Since CoSpaDi evaluates projections as $(\mathbf{X}\mathbf{D}_a)\mathbf{S}$, future improvements in sparse matrix kernels, sparse coefficient layouts, and inference runtimes can directly improve its latency profile.

Table 14: Quality of selective up/gate compression on LLaMA-3-8B. Superscript UG denotes compressing only the MLP up and gate projections.

Method	Global CR	Accuracy↑									Perplexity↓	
		PIQA	HellaSwag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	Wiki	LAMBADA
LLaMA-3-8B	–	80.7	79.1	75.6	77.7	53.5	93.9	40.3	62.2	70.4	7.3E+00	3.1E+00
CoSpaDi	0.20	75.2	66.5	73.8	66.5	41.6	89.5	38.2	42.8	61.8	2.0E+01	4.3E+00
CoSpaDi^{UG}	0.22	75.3	63.2	76.7	67.8	42.1	92.5	37.3	40.2	**61.9**	**2.5E+01**	**2.8E+00**
SVD-LLM^{UG}	0.22	71.9	51.9	56.7	58.8	33.5	87.7	33.8	33.9	53.5	5.6E+01	8.5E+00
CoSpaDi	0.30	70.5	56.2	61.3	54.2	33.5	85.7	36.2	32.2	53.7	**4.5E+01**	9.2E+00
CoSpaDi^{UG}	0.28	70.1	51.5	64.3	58.4	35.3	89.7	34.5	34.7	**54.8**	5.2E+01	**5.8E+00**
SVD-LLM^{UG}	0.28	65.0	40.9	33.0	48.1	28.3	79.9	28.8	26.1	43.8	2.1E+02	5.5E+01

Table 15: Synchronous generation throughput on LLaMA-3-8B using an RTX 3090. Throughput is reported in tokens/s; higher is better.

Prompt	UG CR	SVD-LLM^{UG}	CoSpaDi^{UG}
1	0.3	51.2	46.8
16		50.3	46.8
32		51.3	45.9
64		50.3	45.3
128		50.0	44.8
256		49.5	44.0
Avg.		**50.4**	45.6
1	0.4	53.3	48.4
16		53.5	48.3
32		54.4	48.0
64		54.2	48.4
128		52.7	47.8
256		51.9	47.3
Avg.		**53.3**	48.0
1	0.5	53.8	53.0
16		52.4	52.0
32		52.9	49.1
64		52.4	50.4
128		52.5	49.5
256		50.9	49.0
Avg.		**52.5**	50.5

Table 16: Asynchronous serving throughput on LLaMA-3-8B for prompt length 1. SVD-LLM^{UG} and CoSpaDi^{UG} use UG CR $= 0.60$ and $\rho = 2$; higher is better.

Method	1 user	4 users	8 users	16 users
Dense baseline	77.0	290.0	589.0	1344.0
SVD-LLM^{UG}	**99.4**	**386.0**	**798.0**	1332.0
CoSpaDi^{UG}	95.6	375.6	758.0	**1515.0**

Table 17: Asynchronous serving throughput of CoSpaDi^{UG} on LLaMA-3-8B with UG CR $= 0.60$ and $\rho = 2$. Throughput is reported in tokens/s; higher is better.

Users	Prompt 1	Prompt 16	Prompt 32	Prompt 64	Prompt 128
1	95.6	93.5	92.4	101.0	98.0
4	375.6	374.8	372.4	393.0	401.0
8	758.0	742.0	749.4	802.0	792.0
16	1515.0	1479.6	1472.0	1521.0	1573.0