COMPOT: Calibration-Optimized Matrix Procrustes Orthogonalization for Transformers Compression

Denis Makhov[1,2], **Dmitriy Shopkhoev**[1], **Magauiya Zhussip**[1,2],
Ammar Ali[1,2], **Baher Mohammad**[1,2], **Stamatios Lefkimmiatis**[1,2]
[1]MWS AI [2]ITMO
makhovds@gmail.com
https://github.com/mts-ai/COMPOT

Abstract

Post-training compression of transformer models commonly relies on truncated singular value decomposition (SVD) strategies leading to low-rank approximations of the original model weights. However, the underlying single shared subspace modeling can degrade accuracy even at moderate compression. Sparse dictionary learning provides a more flexible union-of-subspaces representation, but existing approaches often suffer from costly iterative dictionary and coefficient updates. We propose **COMPOT** (**C**alibration-**O**ptimized **M**atrix **P**rocrustes **O**rthogonalization for **T**ransformers), a training-free compression framework that uses a small calibration dataset to estimate a sparse weight factorization. COMPOT employs orthogonal dictionaries that enable closed-form Procrustes updates for the dictionary and analytical single-step sparse coding for the coefficients, reducing significantly the overall optimization cost. To handle heterogeneous layer sensitivity under a global compression budget, COMPOT further introduces a one-shot dynamic allocation strategy that adaptively redistributes layer-wise compression rates. Extensive experiments across diverse architectures and tasks show that COMPOT consistently delivers a superior quality–compression trade-off over strong low-rank and sparse baselines, while remaining fully compatible with post-training quantization for extreme compression.

1 Introduction

Transformer-based foundation models underpin state-of-the-art systems across different modalities, including language, vision, vision-language, and audio [44, 10, 34, 15]. As they scale to billions of parameters, deployment is increasingly limited by memory footprint, bandwidth, and compute [5, 8]. Yet transformers are highly redundant: attention heads and other components can often be removed or simplified with limited degradation, indicating that parameters contribute unevenly to downstream behavior [29, 45]. This has motivated extensive post-training compression work: quantization, pruning, and factorization – to reduce inference cost and memory footprint without expensive retraining [42, 21].

Low-rank matrix factorization is a widely used post-training paradigm, replacing dense projection matrices by products of smaller matrices. Activation-aware and task-aware low-rank methods show that calibration/activation information is crucial for preserving accuracy [18, 52]. Recent SVD-based methods further tailor truncation using calibration and principled losses: SVD-LLM uses whitening-based calibration to relate singular values to truncation loss and applies closed-form updates to mitigate degradation [48]. Despite these advances, SVD methods still enforce a *single shared subspace* per weight matrix, which can be restrictive when different matrix columns lie in different local subspaces [25].

Sparse dictionary learning offers an alternative strategy, modeling a weight matrix as a dense dictionary times a column-sparse coefficient matrix, enabling a *union-of-subspaces* representation.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

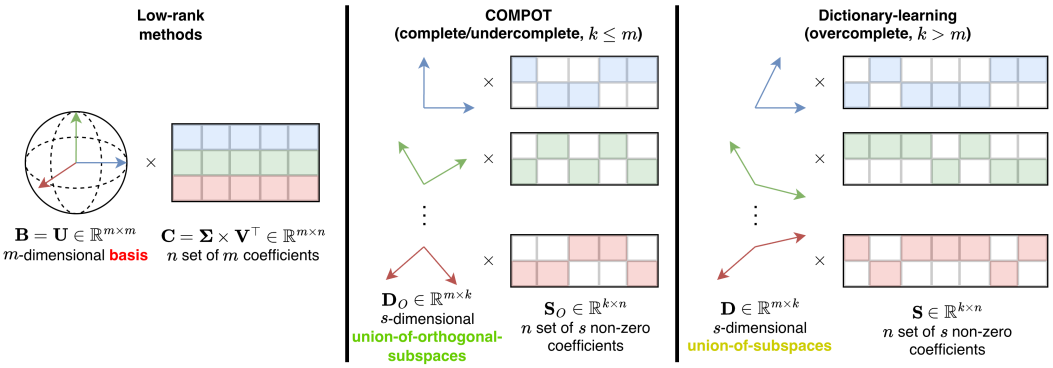

Figure 1: Comparison of low-rank decomposition, sparse dictionary learning, and COMPOT. Low-rank uses a rigid *shared orthogonal basis* $\mathbf{B}$; dictionary learning enables a flexible *union-of-subspaces*; COMPOT lies in between by using a *union-of-**orthogonal**-subspaces* (denoted with $_O$), enabling fast closed-form dictionary and coefficient updates, while improving compression performance.

CoSpaDi shows that calibration-guided sparse dictionary learning can outperform data-aware low-rank baselines for training-free LLM compression [25]. Relatedly, MASA proposes matrix-based dictionary learning for parameter sharing in transformer attention, representing layer-specific projections as combinations of shared matrix atoms [56]. However, existing dictionary learning pipelines typically rely on iterative and expensive dictionary and sparse coding updates (e.g., K-SVD/OMP [1]), limiting practicality at billion-parameter scale.

In this work, we introduce **COMPOT** (**C**alibration-**O**ptimized **M**atrix **P**rocrustes **O**rthogonalization for **T**ransformers), a training-free compression framework that improves calibration-guided dictionary learning in both accuracy and compression time. **First**, our key design choice is to consider complete/undercomplete dictionaries and enforce *column-orthogonality*, leveraging evidence that orthogonal dictionary learning can be effective and substantially faster than overcomplete alternatives [2]. Under orthogonality, the dictionary update transforms to an orthogonal Procrustes problem with a closed-form SVD solution [38], replacing atom-wise K-SVD updates. Meanwhile, the sparse coding update also reduces to a closed-form hard-thresholding operation, avoiding iterative pursuits and yielding an efficient alternating procedure [11].

Transformer layers and projection types exhibit heterogeneous redundancy. Prior work addresses this via non-uniform rank allocation (e.g., SVD-LLM V2) [47] or differentiable truncation optimization (Dobi-SVD) [33]. **Second**, we instead propose a *one-shot* global allocation strategy: we normalize each weight matrix, pool singular values across matrices into a global spectrum, and truncate globally to satisfy a model-wide budget, while enforcing constraints that prevent negative or low per-matrix compression and cap over-compression of sensitive layers. This yields a simple, deterministic allocation procedure without iterative search over layer-wise ratios.

Third, COMPOT integrates naturally with post-training quantization. Among PTQ methods such as SmoothQuant [50], GPTQ [13], and AWQ [20], we show that 4-bit GPTQ applied on top of COMPOT yields superior results compared to quantization alone under equal memory budgets. **Fourth**, COMPOT outperforms strong SVD baselines by a wide margin across architectures and downstream tasks (language, audio, vision). Overall, the proposed method advances the state of the art among structured matrix-factorization compression techniques, with closed-form alternating updates supported by theoretical convergence guarantees, shifting the paradigm beyond established SVD-based approaches.

2 Related Work

2.1 Overview of transformer-based Model Compression

Compression of transformer-based models spans several complementary families. Early *pruning* work showed that many attention heads can be removed with little degradation, motivating structured pruning of self-attention and encoder-decoder heads to reduce compute [29, 45]. *Post-training quantization* (PTQ) is now the dominant deployment tool for LLMs: SmoothQuant enables near-lossless W8A8 quantization by migrating activation outliers into weights via offline rescaling [50]; GPTQ/OPTQ use second-order, block-wise weight-only PTQ to reach 3-4 bits [13]; and AWQ improves low-bit PTQ by identifying and protecting activation-salient weights [20]. Orthogonal

to quantization, *low-rank and structured factorizations* replace dense projections with products of thinner matrices; recent SVD-based methods (SVD-LLM, SVD-LLM V2) combine truncation-aware whitening, closed-form updates, and optimized rank allocation to better align truncation with functional loss [48, 47]. Finally, *knowledge distillation* trains a smaller student to match a larger teacher; DistilBERT reduces BERT parameters by 40% while preserving most downstream accuracy [37].

2.2 SVD-based Matrix Factorization for Compression

Low-rank factorization is a standard post-training approach for compressing transformer projections, motivated by the empirical low-rank structure of weight matrices. While truncated SVD is optimal in Frobenius norm, this objective can be misaligned with preserving task behavior. Early data-aware formulations address this mismatch: DRONE derives a provably optimal low-rank decomposition with closed-form solutions for feed-forward and self-attention matrices in BERT-style models [7], and FWSVD improves robustness by Fisher-weighting reconstruction error [18]. Subsequent work incorporates activation information: ASVD leverages activation statistics and layer sensitivity to guide training-free truncation [52]. More recently, SVD-LLM introduces truncation-aware whitening and closed-form parameter updates to mitigate degradation at higher compression [48], Dobi-SVD proposes differentiable truncation with a remapping strategy to better connect truncation choices to storage behavior [33], and SVD-LLM V2 further refines truncation-loss modeling and allocates non-uniform compression ratios across matrices to reflect heterogeneous redundancy [47].

Matrix compression using truncated SVD essentially amounts to applying Principal Component Analysis (PCA) to the matrix columns: it finds the best L_2-norm subspace to project the data, creating an optimal low-rank approximation [3]. However, this approach forces the entire matrix into one shared subspace. This can be too restrictive if different columns would be better represented by their own unique, local subspaces.

2.3 Dictionary Learning and Sparse Coding

Sparse dictionary learning is a complementary paradigm: it represents a weight matrix as the product of a dictionary and a sparse coefficient matrix. This formulation enables different columns to select distinct subsets of atoms, thereby inducing a *union-of-subspaces* structure. Classic methods such as K-SVD iteratively update dictionary atoms and are widely used in signal and image processing [1]. CoSpaDi adapts calibration-guided dictionary learning to LLM compression by minimizing functional output mismatch rather than pure weight error, and shows that structured sparse factorizations can outperform data-aware SVD baselines at practical compression ratios [25]. However, K-SVD-style updates are computationally heavy at billion-model scale, motivating alternative constraints and update rules. To address this challenge, our method restricts the weight factorization to employ complete or undercomplete orthogonal dictionaries, depending on the matrix shape and storage budget. This structural constraint substantially simplifies optimization: the dictionary update reduces to a closed-form orthogonal Procrustes solution, while sparse coding admits an analytical solution, entirely eliminating the need for iterative sparse pursuit algorithms. Our approach is well motivated, as the orthogonality constraint on learned dictionaries has been investigated in image processing and shown to improve both numerical stability and computational efficiency [2].

2.4 Dynamic Allocation of Compression Ratios

A recurring theme in transformer compression is heterogeneous redundancy across layers and projection types, making uniform compression suboptimal. ASVD highlights sensitivity differences and uses activation-aware heuristics to reduce degradation [52]. SVD-LLM V2 assigns matrix-specific compression ratios via truncation-loss modeling under a global budget [47]. Dobi-SVD targets rank selection with differentiable truncation and introduces a remapping that better connects truncation to achievable compression [33]. Recent work continues this direction: D-Rank allocates ranks dynamically under a fixed budget via optimization [28], and ARA introduces adaptive rank allocation for SVD-based compression [51]. These results reinforce that *budget allocation* is central to minimizing quality drop for a given compression target. In this work, we contribute a simple one-shot allocation procedure that operates over pooled singular values while enforcing constraints that prevent negative compression and limit over-compression of individual matrices.

3 Method

3.1 Problem Setup

We consider a pretrained transformer with linear projections parameterized by weight matrices $\mathbf{W} \in \mathbb{R}^{m \times n}$ (e.g., attention and MLP projections). Our goal is *post-training* compression: replace

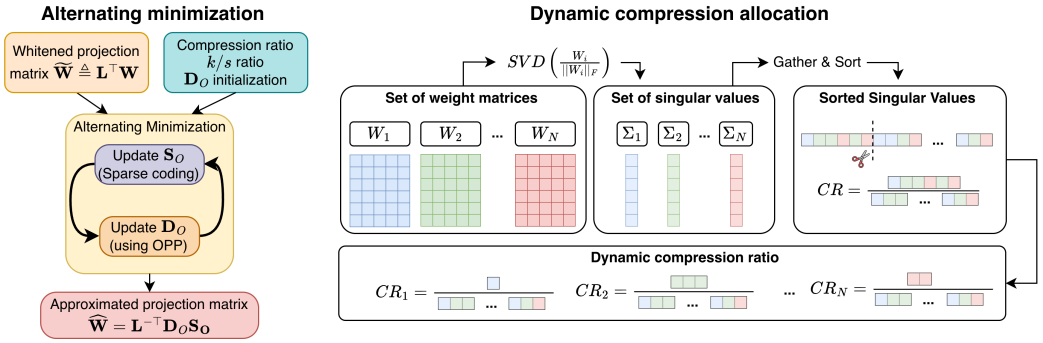

Figure 2: Overview of COMPOT framework. On the *left* part the alternating minimization process is visualized while on the *right* we represent our single-shot strategy for dynamic compression ratio allocation based on singular values of normalized projection matrices.

each $\mathbf{W}$ with a structured factorization that reduces the model parameter count while preserving the layer behavior on a small calibration set, without back-propagation (healing/fine-tuning).

Let $\mathbf{X} \in \mathbb{R}^{N\times m}$ denote a matrix of calibration inputs (layer activations entering the projection), where N is the number of calibration tokens/examples aggregated into rows. The layer output on calibration data is $\mathbf{Y} = \mathbf{X}\mathbf{W} \in \mathbb{R}^{N\times n}$. A natural data-aware objective is therefore to minimize output mismatch $\|\mathbf{X}(\mathbf{W} - \widehat{\mathbf{W}})\|_F^2$ over compressed $\widehat{\mathbf{W}}$.

3.2 Preliminaries: Subspace vs. Union-of-subspaces Modeling

Low-rank (single shared subspace). For a real-valued matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$, truncated SVD gives

$$\mathbf{W} \approx \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^\top, \tag{1}$$

where $\mathbf{U}_r \in \mathbb{R}^{m\times r}$, $\mathbf{\Sigma}_r \in \mathbb{R}^{r\times r}$, and $\mathbf{V}_r \in \mathbb{R}^{n\times r}$. From the direct link with PCA and the underlying concept of projecting columns of $\mathbf{W}$ to a lower-dimensional subspace [3] we have:

$$\mathbf{W} \approx \mathbf{B}\mathbf{C}, \text{ where } \mathbf{B} \triangleq \mathbf{U}_r, \ \mathbf{C} \triangleq \mathbf{\Sigma}_r\mathbf{V}_r^\top, \tag{2}$$

so each column $\mathbf{w}_j$ is represented in the *same* r-dimensional subspace spanned by $\mathbf{B}$ (Fig. 1 (left)).

Sparse dictionary learning (union-of-subspaces). Dictionary learning (Fig. 1 (right)) represents the projection matrix with an *overcomplete* dictionary $\mathbf{D}$ and corresponding sparse coefficient $\mathbf{S}$

$$\mathbf{W} \approx \mathbf{D}\mathbf{S}, \qquad \mathbf{D} \in \mathbb{R}^{m\times k}, \quad \mathbf{S} \in \mathbb{R}^{k\times n}, \tag{3}$$

where $\mathbf{S}$ is column-sparse: each column $\mathbf{s}_j$ has at most s nonzeros. As a result, each column $\mathbf{w}_j$ may be represented with a different subset of atoms, inducing a *union-of-subspaces* model: columns can live in different s-dimensional subspaces of $\mathbb{R}^m$ spanned by subsets of atoms of $\mathbf{D}$.

3.3 COMPOT: Calibration-optimized Orthogonal Dictionary Factorization

COMPOT preserves the union-of-subspaces flexibility of sparse coding, but enforces an *orthogonal* (complete/undercomplete) dictionary in a *whitened, data-aware* space (Fig. 1 (middle)). This yields closed-form updates (Fig. 2 (left)): Procrustes (thin SVD) for the dictionary and an analytical sparse coding step under orthogonality, resulting in a *union-of-orthogonal-subspaces* model.

Data-aware whitening. We adopt a standard calibration objective that measures functional error on calibration activations:

$$\min_{\widehat{\mathbf{W}}} \ \|\mathbf{X}(\mathbf{W} - \widehat{\mathbf{W}})\|_F^2. \tag{4}$$

First we define the Gram matrix, $\mathbf{G} \triangleq \mathbf{X}^\top\mathbf{X} \in \mathbb{R}^{m\times m}$, which we assume to be positive definite (this condition is typically satisfied using sufficient calibration data) and admits the Cholesky decomposition $\mathbf{G} = \mathbf{L}\mathbf{L}^\top$, with $\mathbf{L} \in \mathbb{R}^{m\times m}$ the Cholesky factor. Then it holds:

$$\|\mathbf{X}(\mathbf{W} - \widehat{\mathbf{W}})\|_F^2 = \mathrm{Tr}\Big((\mathbf{W} - \widehat{\mathbf{W}})^\top\mathbf{G}(\mathbf{W} - \widehat{\mathbf{W}})\Big) = \|\mathbf{L}^\top(\mathbf{W} - \widehat{\mathbf{W}})\|_F^2. \tag{5}$$

Therefore, optimizing the functional error is equivalent to optimizing the reconstruction error in the *whitened space*. We define the whitened weight matrix

$$\widetilde{\mathbf{W}} \triangleq \mathbf{L}^\top \mathbf{W} \in \mathbb{R}^{m \times n}. \tag{6}$$

COMPOT objective. COMPOT factorizes $\widetilde{\mathbf{W}}$ using an *orthogonal* (complete or undercomplete) dictionary $\mathbf{D}_O$ and column-sparse codes $\mathbf{S}_O$:

$$\mathbf{D}_O^\star, \mathbf{S}_O^\star = \arg\min_{\mathbf{D}_O, \mathbf{S}_O} \|\widetilde{\mathbf{W}} - \mathbf{D}_O \mathbf{S}_O\|_F^2 \quad \text{s.t.} \quad \mathbf{D}_O^\top \mathbf{D}_O = \mathbf{I}_k,\ \|\mathbf{s}_{\mathbf{O}j}\|_0 \le s\ \forall j \in [n], \tag{7}$$

where $\mathbf{D}_O \in \mathbb{R}^{m \times k}$ with $k \le m$ (complete/undercomplete), and $\mathbf{S}_O \in \mathbb{R}^{k \times n}$.

After solving (7), we map the factorization back to the original parameter space:

$$\mathbf{W} \approx \widehat{\mathbf{W}} \triangleq \mathbf{A}\mathbf{S}_O, \qquad \mathbf{A} \triangleq \mathbf{L}^{-\top}\mathbf{D}_O \in \mathbb{R}^{m \times k}. \tag{8}$$

Importantly, $\mathbf{A}$ is computed *offline* during compression, so inference uses only $(\mathbf{A}, \mathbf{S}_O)$.

Alternating minimization (Sparse coding + Procrustes). Although joint minimization of (7) is not feasible, we can still opt for a simple alternating minimization with closed-form updates when enforcing $\mathbf{D}_O^\top \mathbf{D}_O = \mathbf{I}_k$. With $\mathbf{D}_O$ fixed, sparse coding decouples column-wise and has an analytical solution [11]:

$$\mathbf{S}_O \leftarrow \mathcal{H}_s\big(\mathbf{D}_O^\top \widetilde{\mathbf{W}}\big), \tag{9}$$

where $\mathcal{H}_s(\cdot)$ (hard-thresholding operator) keeps the s largest-magnitude entries in each column and zeroes out the rest. With $\mathbf{S}_O$ fixed, the dictionary update is an orthogonal Procrustes step [38]:

$$\mathbf{M} \triangleq \widetilde{\mathbf{W}}\mathbf{S}_O^\top, \quad \mathbf{M} \overset{\text{SVD}}{=} \mathbf{P}\mathbf{\Lambda}\mathbf{Q}^\top \ \Rightarrow\ \mathbf{D}_O \leftarrow \mathbf{P}\mathbf{Q}^\top. \tag{10}$$

We provide the full derivation of (9), the complete procedure, implementation details and convergence analysis in A.1.

One-shot compression allocation. Different transformer projections exhibit heterogeneous redundancy, so uniform compression is often suboptimal. We propose a simple one-shot global allocation strategy that distributes a model-wide budget by globally truncating the smallest singular values across matrices, while enforcing per-matrix compression guards and handling matrices where factorization is not beneficial.

Original or whitened space? Whitening is layer- and projection-specific (depends on $\mathbf{X}$), so singular values computed in whitened coordinates are not directly comparable across matrices. We therefore allocate ranks using singular values of the *raw* weight matrices in the original space, while still using whitening for COMPOT reconstruction.

Normalize or not? Even in the original space, spectra magnitudes vary across layers. To address this, we normalize each matrix by its Frobenius norm: $\mathbf{W}_i^{(f)} \triangleq \mathbf{W}_i / \|\mathbf{W}_i\|_F$.

Following normalization, we compute a thin SVD for each $\mathbf{W}_i^{(f)}$ and pool all singular values into a global multiset. We sort this pool and truncate the smallest values to satisfy the desired model-wide compression ratio; the number truncated per matrix then induces its layer-wise compression ratio. To prevent degenerate behavior, we enforce minimum and maximum compression guards and handle matrices where factorization is not beneficial. Fig. 2 (right) illustrates the simplified procedure; A.2 provides full details.

Compression ratio (CR). In the proposed method we need to store both $\mathbf{A}$ (unwhitened dictionary) and $\mathbf{S}_O$ with 16-bit values. Instead of storing the sparse matrix directly, we store only its nonzero elements and a binary mask of their position:

$$\text{CR}_{\text{COMPOT}} = 1 - \frac{\overbrace{16mk}^{\mathbf{A}} + \overbrace{16sn}^{\mathbf{S}_O} + \overbrace{(kn)}^{\text{Mask}}}{16mn}. \tag{11}$$

For the $\text{CR}_{\text{COMPOT}}$ computation we need to specify both the number of atoms k and the number of column-wise non-zero elements s. We express both parameters with a single hyperparameter k/s-ratio.

4 Experiments

We evaluate COMPOT across model families (Llama, OPT, Qwen; 0.6B–30B) and domains (language, vision-language, audio), under compression ratios (CR) 0.2–0.6. We compare against strong SVD-based baselines (SVD-LLM [48], SVD-LLM V2 [47], Dobi-SVD [33]), sparse dictionary learning (CoSpaDi [25]), and structured pruning (ReplaceMe [39], LLM-Pruner [23]), and study compatibility with post-training quantization (GPTQ [13]). Hardware setup and reproducibility analysis are presented in A.13 and A.12, respectively.

4.1 Experimental Setup

Unless stated otherwise, we set the dictionary-to-sparsity ratio to $k/s = 2$ (see A.8) – adjusted only when necessary to avoid overcomplete dictionaries that violate the orthogonality constraint – and compress all dense linear projections in self-attention (Q/K/V/O) and the MLP (gate/up/down), keeping token embeddings and `lm_head` uncompressed in alignment with prior works. We report zero-shot accuracy on PIQA [4], HellaSwag [54], OpenAI LAMBADA [31], ARC-easy/ARC-challenge [9], SciQ [49], RACE [19], and MMLU [16], and perplexity on WikiText [27] and LAMBADA-OpenAI. For SVD-LLM, CoSpaDi, and pruning baselines we use `lm-evaluation-harness` 0.4.8 [14] with normalized accuracies when available; for SVD-LLM V2 and Dobi-SVD comparisons we additionally follow the original SVD-LLM evaluation protocol (unnormalized accuracies and the authors' perplexity script). Additional benchmarks/ablations are reported in A.4.

4.2 Ablation Study

COMPOT involves several hyperparameters that must be chosen carefully, in particular the dictionary initialization, the number of alternating minimization steps, the strategy for dynamic allocation and the choice of calibration data. In this section we study their impact on performance. Unless stated otherwise, ablations report average accuracy (Avg. Acc.) over the datasets in Table 4, together with WikiText and LAMBADA perplexity (Wiki. PPL and LAMB. PPL, correspondingly).

Table 1: Effect of dictionary initialization for Llama3.2-1B at 0.2 compression with 20 alternating minimization iterations. Best results are shown in **bold**.

CR Allocation	Init.	Avg. Acc.↑	Wiki. PPL↓	LAMB. PPL↓
Static	Rand.	46.3	26.84	24.74
	SVD	**48.5**	**24.67**	**20.62**
Dynamic	Rand.	48.5	22.05	15.06
	SVD	**50.1**	**20.71**	**13.97**

Table 2: Effect of grouping for dynamic allocation for Llama3.2-1B at 0.2 compression with 20 alternating minimization iterations. Best results are shown in **bold**.

Grouping	Avg. Acc.↑	Wiki. PPL↓	LAMB. PPL↓
All indiv.	48.5	24.62	18.62
QKV&UpGate	49.2	21.41	15.52
All grouped	**50.1**	**20.71**	**13.97**

Dictionary initialization. We compare random-column initialization (randomly orthonormalized permuted subset of $\widetilde{\mathbf{W}}$ columns) against SVD initialization (top left singular vectors). Table 1 shows that SVD initialization consistently improves performance at fixed iteration budget.

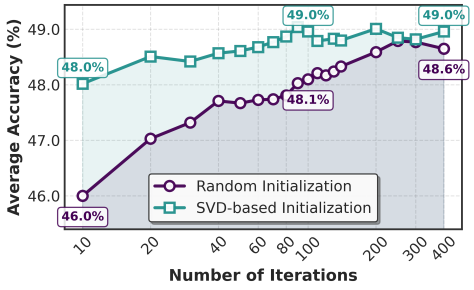

Figure 3: Average accuracy as a function of the number of alternating minimization steps on Llama3.2-1B at 0.2 compression, comparing random and SVD-based dictionary initialization.

Table 3: Calibration data effect on COMPOT with dynamic allocation and SVD initialization. Best results are shown in **bold**.

		Avg. Acc.↑	Wiki. PPL↓	LAMB. PPL↓
Llama3.2 1B		57.6	11.57	5.73
CR 0.2	WikiText	50.0	**18.52**	**13.15**
	Alpaca	**50.1**	23.58	18.33
	FineWeb	**50.1**	20.71	13.97
CR 0.3	WikiText	44.9	**31.12**	32.97
	Alpaca	44.8	51.28	82.64
	FineWeb	**45.0**	35.68	**32.35**
CR 0.4	WikiText	36.4	**109.11**	299.72
	Alpaca	35.3	447.37	2857.45
	FineWeb	**37.6**	129.28	**195.41**

Number of alternating updates. We vary the number of alternating updates on Llama3.2-1B at static CR 0.2 (Fig. 3). Random initialization improves up to ∼300 iterations, while SVD initialization saturates much earlier (∼100). We use 20 iterations in all main experiments as a pragmatic accuracy–time compromise.

Table 4: Performance comparison under *static* CR allocation of COMPOT† vs state-of-the-art SVD-based SVD-LLM [48] and dictionary-learning CoSpaDi [25] on Llama3-8B and Qwen3-8B at different compression levels on different benchmarks. Best results are shown in **bold**.

Method	CR	Accuracy↑									Perplexity↓	
		PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	WikiText	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	0.2	75.2	66.5	73.8	66.5	41.6	89.5	38.2	42.8	61.8	2.0E+01	**4.3E+00**
COMPOT†	0.2	77.8	72.7	69.4	71.0	44.5	89.8	41.0	50.1	**64.5**	**1.3E+01**	5.2E+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	0.3	70.5	56.2	61.3	54.2	33.5	85.7	36.2	32.2	53.7	4.5E+01	9.2E+00
COMPOT†	0.3	74.4	64.6	60.9	61.0	38.0	87.1	38.6	38.9	**58.0**	**2.1e+01**	**8.5E+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	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
COMPOT†	0.4	68.2	51.4	38.8	46.9	31.0	75.4	34.4	27.9	**46.7**	**6.2e+01**	**4.2E+01**
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	0.2	76.5	68.0	65.6	72.2	48.9	93.2	40.7	60.8	65.7	1.8E+01	**4.9E+00**
COMPOT†	0.2	75.5	72.2	62.5	70.7	49.0	92.4	42.9	66.0	**66.4**	**1.5E+01**	5.7E+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	0.3	72.4	60.5	62.6	63.9	41.2	88.4	39.5	51.3	60.0	2.3E+01	6.3E+00
COMPOT†	0.3	75.0	65.9	62.5	66.3	44.2	90.1	42.1	58.2	**63.0**	**1.8E+01**	**6.2E+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	0.4	68.9	49.0	49.9	49.4	29.9	82.0	36.8	36.6	50.3	3.6E+01	1.5E+01
COMPOT†	0.4	70.9	56.4	54.7	53.7	34.9	84.3	38.7	44.4	**54.7**	**2.5E+01**	**9.8E+00**

Grouping for dynamic allocation. We test pooling singular values by projection type (All indiv. similar to SVD-LLM V2), partial grouping (QKV&UpGate), and a single global pool (All grouped) on Llama3.2-1B at CR 0.2. Table 2 shows global pooling yields the best overall accuracy/perplexity, motivating our default global allocation.

Calibration data influence. We investigated the influence of calibration data used for Gram matrix estimation. We sampled 256 random samples of length 1024 from WikiText [26], FineWeb [32] and Alpaca [43] datasets. The ablation study in Table 3 reveals that FineWeb provides a reasonable compromise across all metrics, prompting its use in our further experiments. We additionally study sensitivity to the number of calibration samples in Appendix A.12.

4.3 Main Results

Uniform CR (no allocation). We first disable dynamic allocation and apply a uniform CR to all projections (COMPOT†). We compare against SVD-LLM [48] as a strong SVD-based baseline and CoSpaDi [25] as a K-SVD-based sparse dictionary method. Calibration uses 256 sequences from FineWeb [32] (length 1024). Table 4 reports results on Llama3-8B and Qwen3-8B; smaller models (Llama3.2-1B, Qwen3-0.6B) are provided in A.3. Across CRs, COMPOT† yields higher average accuracy and lower perplexity than both low-rank (SVD-LLM) and K-SVD-style dictionary learning (CoSpaDi), indicating that orthogonal dictionary factorization improves the quality–compression trade-off under a fixed per-layer budget.

Table 5: Performance comparison under *dynamic* compression ratio allocation of *training-free* COMPOT vs *training-based* state-of-the-art SVD-based Dobi-SVD [33] on Llama2-7B at different compression levels on different benchmarks. Best results are shown in **bold**.

Method	Training-free	CR	Perplexity↓		Accuracy↑							
			WikiText-2	C4	Openb.	ARC_e	WinoG.	HellaS.	ARC_c	PIQA	MathQA	Avg.
Llama2-7B	—		5.49	7.27	44.0	74.6	69.0	76.3	45.5	78.6	28.4	59.5
Dobi-SVD†	✗	**0.2**	9.39	19.70	32.8	47.6	58.5	51.5	29.3	65.2	22.7	43.9
COMPOT	✓	**0.2**	**6.22**	**9.34**	30.4	72.6	67.6	51.8	39.6	76.4	26.2	**52.1**
Dobi-SVD†	✗	**0.4**	17.50	55.58	29.2	34.4	54.4	35.4	24.5	57.3	21.3	36.6
COMPOT	✓	**0.4**	**8.91**	**20.40**	25.2	63.1	63.5	39.6	28.5	67.0	23.6	**44.4**
Dobi-SVD†	✗	**0.6**	53.40	200.11	26.0	28.3	50.5	28.4	24.7	51.9	22.0	**33.1**
COMPOT	✓	**0.6**	**32.10**	**179.02**	13.4	33.8	53.4	28.8	18.8	54.9	22.1	32.2

Dynamic CR allocation. We evaluate full COMPOT with our one-shot allocation and compare against methods that explicitly optimize layer-wise ranks, namely Dobi-SVD [33] and SVD-LLM V2 [47]. To ensure a like-for-like comparison, we follow the released SVD-LLM/Dobi-SVD evaluation protocols: we calibrate with 256 WikiText train split samples at context length 2048, report *unnormalized* accuracies via `lm-eval` 0.4.8, and compute perplexity on WikiText (test split) and C4 using the scripts provided in the SVD-LLM repository. Results are summarized in Tables 5 and 6, with additional Llama-13B/30B results in A.10. We also present CR allocation plots in A.7

For SVD-LLM V2, the paper points to the SVD-LLM repository for code, but at the time of our experiments the repository does not provide a dedicated, ready-to-run V2 implementation; therefore, we implemented the V2 dynamic allocation and loss-optimized truncation strategies on top of the released SVD-LLM codebase and report both our reproduction (denoted with 'repr.') and the paper-reported numbers in Table 6. Discrepancies between published and reproduced results, together with reproduction details, are discussed in A.10. For Dobi-SVD, we evaluate the provided checkpoints *without* the "remapping" step, which alters the effective storage/parameterization and mixes factorization with quantization; we include the full remapping-inclusive comparison in A.11. Under matched global CR, COMPOT consistently achieves stronger accuracy/perplexity while remaining fully training-free.

Table 6: Performance comparison under *dynamic* CR between COMPOT and SVD-LLM V2 [47] on Llama-7B, OPT-6.7B, and Llama3-8B at a CR 0.2. Best results are shown in **bold**

Methods	Llama-7B Wiki / C4 ↓	OPT-6.7B Wiki / C4 ↓	Llama3-8B Wiki / C4 ↓
Original	5.67 / 7.33	10.92 / 12.57	6.14 / 9.47
SVD-LLM V2	7.12 / 10.47	13.46 / 17.72	**8.01 / 11.72**
SVD-LLM V2 (repr.)	7.92 / 16.23	15.16 / 19.77	48.80 / 88.88
COMPOT	**6.52 / 9.63**	**10.92 / 13.17**	8.64 / 18.29

Table 7: Perplexity (↓) of COMPOT vs state-of-the-art SVD-based SVD-LLM V2 [47] accompanied by quantization on Llama-7B on WikiText-2. Best results are shown in **bold**.

Method	Memory	QCR	FCR	PPL
GPTQ-3bit	2.8 GB	0.81	N/A	16.28
SVD-LLM V2+GPTQ-4bit	2.8 GB	0.75	0.25	9.97
COMPOT†+GPTQ-4bit	2.8 GB	0.75	0.25	9.93
COMPOT+GPTQ-4bit	2.8 GB	0.75	0.25	**9.62**

Table 8: Comparison of the COMPOT with other training-free methods including state-of-the-art structured pruning methods ReplaceMe [39] and LLM-Pruner [23] on Llama3 8B under different compression ratios. We report accuracy on different benchmarks as well as its average and perplexity. Best results are shown in **bold**.

Method	CR	Accuracy↑									Perplexity↓	
		PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	WikiText	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	0.20	75.5	67.5	51.0	62.1	36.6	87.8	35.1	25.0	55.1	1.6E+01	1.1E+01
COMPOT		78.0	73.5	70.5	71.3	45.3	91.0	40.1	52.4	**65.3**	**1.2E+01**	**4.6E+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	0.30	67.3	45.1	20.9	45.4	28.8	63.4	30.1	22.9	40.5	3.8E+01	2.2E+02
COMPOT		76.6	67.2	64.3	69.4	41.6	90.1	38.4	39.8	**60.9**	**1.7E+01**	**6.6E+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	0.40	50.3	25.8	1.5	26.4	25.8	28.1	21.8	23.2	25.4	∞	5.7E+05
COMPOT		71.0	55.8	44.9	56.6	33.5	85.4	36.6	29.6	**51.7**	**4.3E+01**	**2.2E+01**

Comparison to structured pruning. We compare COMPOT against ReplaceMe [39] and LLM-Pruner [23] on Llama3-8B (Table 8). We exclude unstructured pruning methods since high sparsity often does not translate into memory/latency savings on dense backends. At comparable effective compression, COMPOT preserves substantially higher accuracy and lower perplexity than structured pruning, suggesting that factorizing projections is a more robust mechanism for removing redundancy than deleting entire channels/heads.

Compatibility with post-training quantization. Following SVD-LLM V2 [47], we apply 4-bit GPTQ on top of both SVD-based and COMPOT-based factorizations and compare against GPTQ-only under matched weight memory. Table 7 reports WikiText-2 perplexity on Llama-7B, together with the decomposition of total compression into quantization and factorization components (QCR and FCR, correspondingly). COMPOT+GPTQ yields lower perplexity than GPTQ-only and SVD-LLM V2+GPTQ, showing that COMPOT provides gains complementary to modern PTQ.

Vision-language transfer. We evaluate COMPOT on Qwen3-VL-8B-Instruct at CR 0.2 (additional CRs and setup are provided in the A.9). Table 9 reports MMMU [53], OCRBench [22], RealWorldQA, and MMStar [6]. SVD-LLM degrades severely, whereas static COMPOT† and COMPOT retain strong multimodal performance, with COMPOT achieving the best average accuracy.

Audio transfer. We evaluate COMPOT on Whisper [35] (Whisper Base: 72M; Whisper Medium: 763M; Whisper Large: 2B) and report WER on LibriSpeech *test-clean* (N=2,650) and *test-other* (N=2,939) [30]. As a baseline we apply SVD-LLM to the same decoder projections and calibrate on the same LibriSpeech-derived calibration set. Table 10 shows that on Whisper Large, COMPOT is consistently more robust than SVD-LLM across CRs; additional audio results and full setup are provided in A.9

Grouped sharing scenario. COMPOT naturally extends to a grouped setting where a single dictionary is shared across multiple projection layers. While related to BasisSharing [46], we instead learn a shared orthogonal dictionary across heterogeneous projections. For fair comparison, we follow their grouping protocol with static rank allocation and include CoSpaDi as a dictionary-learning baseline. Under equal compression, performance is regime-dependent: at 0.2, grouped CoSpaDi is on par in accuracy, but COMPOT is substantially more efficient due to orthogonal updates and no iterative sparse coding; at higher compression, COMPOT consistently outperforms both by a wide margin in reconstruction and downstream metrics (Table 11). We report normalized accuracies, with unnormalized values in parentheses following BasisSharing.

Table 9: Multimodal performance of Qwen3-VL-8B-Instruct under 0.2 compression ratio (CR) on vision-language benchmarks. Best results are shown in **bold**.

Method	MMMU ↑	OCRBench ↑	RealWorldQA ↑	MMStar ↑
Original	52.6	82.6	69.7	63.2
SVD-LLM	29.2	35.1	52.9	31.6
COMPOT†	41.9	64.1	59.2	50.8
COMPOT	**44.7**	**66.9**	**62.2**	**54.2**

Table 10: ASR Performance (WER ↓) on LibriSpeech benchmarks for Whisper Large V3 model. Best results are shown in **bold**.

Method	CR	$\text{WER}_{\text{test-clean}}$ ↓	$\text{WER}_{\text{test-other}}$ ↓
Original	-	2.74	4.53
SVD-LLM	0.2	4.12	6.8
COMPOT†	0.2	**2.46**	**4.51**
SVD-LLM	0.3	12.78	15.54
COMPOT†	0.3	**2.74**	**5.21**

Table 11: Performance comparison of COMPOT†(static CR) vs low-rank and dictionary-learning methods in per-layer and grouping scenarios on Llama2-7B under different compression levels on various benchmarks. Best average accuracy and perplexities are highlighted with **bold**. Unnormalized accuracies are shown in parentheses for direct comparison with [46, 25].

Method	CR	Accuracy↑ (% (unnorm))									Perplexity↓	
		PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	WikiText	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
COMPOT†(per-layer)		76.9 (75.4)	68.8 (50.3)	69.4	64.2 (69.8)	38.7 (36.7)	85.7 (91.8)	39.5	29.4	59.1 (57.8)	12.46	5.00
BasisSharing		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**
COMPOT†(grouped)		74.2 (72.7)	70.0 (51.0)	68.6	63.6 (69.1)	39.8 (37.7)	90.4 (93.2)	39.6	28.4	**59.3** (57.5)	**11.58**	4.98
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
COMPOT†(per-layer)		74.1 (72.6)	64.0 (47.0)	66.6	59.7 (65.2)	36.5 (33.1)	83.2 (90.8)	38.3	27.4	56.2 (55.1)	16.00	5.74
BasisSharing		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
COMPOT†(grouped)		73.0 (71.4)	66.6 (48.8)	67.3	61.2 (66.5)	36.9 (36.0)	88.7 (92.8)	38.4	27.4	**57.4** (56.1)	**13.12**	**5.66**
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
COMPOT†(per-layer)		71.3 (69.3)	56.4 (41.8)	56.2	50.8 (56.9)	30.1 (27.1)	77.6 (86.8)	35.0	23.8	50.2 (49.6)	25.01	9.95
BasisSharing		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
COMPOT†(grouped)		68.0 (67.3)	58.5 (42.3)	58.2	57.0 (62.4)	33.7 (30.3)	86.1 (89.5)	34.8	23.2	**52.5 (51.0)**	**17.84**	**9.74**

Computational efficiency. A.5 reports substantial compression-time speedups over iterative dictionary learning. A.6 further shows comparable theoretical inference complexity to SVD and, using MACKO sparse kernels, a 21.1% average throughput improvement over dense inference.

5 Limitations and Conclusions

We presented COMPOT, a training-free weight compression framework using orthogonal dictionary learning with closed-form Procrustes updates, analytical sparse coding, one-shot global allocation and theoretical convergence guarantees for its alternating optimization procedure. Across language, vision-language, and speech models, COMPOT improves the quality–compression trade-off over strong SVD and sparse dictionary baselines.

COMPOT shares limitations of calibration-driven compression: performance depends on representative calibration data. Whitening is straightforward with a well-conditioned Gram/covariance matrix; otherwise mild regularization is needed. Future work could improve allocation, grouping or add a lightweight "healing" step, either with fixed or learned sparsity pattern on the matrix coefficients.

Broader impacts. COMPOT can reduce memory, compute, deployment cost, and energy use, improving access to large language, vision-language, and speech models. Easier deployment may also amplify existing model risks such as bias, privacy leakage, misinformation, or misuse; compressed models should therefore follow the same safeguards and domain-specific evaluations as their uncompressed counterparts.

Acknowledgments

This research was supported as part of the ITMO University Research Projects in AI Initiative

References

[1] AHARON, M., ELAD, M., AND BRUCKSTEIN, A. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing 54*, 11 (2006), 4311–4322.

[2] BAO, C., CAI, J.-F., AND JI, H. Fast sparsity-based orthogonal dictionary learning for image restoration. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (2013).

[3] BISHOP, C. M. *Pattern recognition and machine learning*, vol. 4. Springer, 2006.

[4] BISK, Y., ZELLERS, R., GAO, J., CHOI, Y., ET AL. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence* (2020), vol. 34, pp. 7432–7439.

[5] BROWN, T., MANN, B., RYDER, N., SUBBIAH, M., KAPLAN, J. D., DHARIWAL, P., NEELAKANTAN, A., SHYAM, P., SASTRY, G., ASKELL, A., ET AL. Language models are few-shot learners. *Advances in neural information processing systems 33* (2020), 1877–1901.

[6] CHEN, L., LI, J., DONG, X., ZHANG, P., ZANG, Y., CHEN, Z., DUAN, H., WANG, J., QIAO, Y., LIN, D., AND ZHAO, F. Are we on the right way for evaluating large vision-language models? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems* (2024).

[7] CHEN, P. H., YU, H.-F., DHILLON, I. S., AND HSIEH, C.-J. DRONE: Data-aware low-rank compression for large NLP models. In *Advances in Neural Information Processing Systems* (2021), vol. 34, pp. 29321–29334.

[8] CHOWDHERY, A., NARANG, S., DEVLIN, J., BOSMA, M., MISHRA, G., ROBERTS, A., BARHAM, P., CHUNG, H. W., SUTTON, C., GEHRMANN, S., ET AL. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research 24*, 240 (2023), 1–113.

[9] CLARK, P., COWHEY, I., ETZIONI, O., KHOT, T., SABHARWAL, A., SCHOENICK, C., AND TAFJORD, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457* (2018).

[10] DOSOVITSKIY, A., BEYER, L., KOLESNIKOV, A., WEISSENBORN, D., ZHAI, X., UNTERTHINER, T., DEHGHANI, M., MINDERER, M., HEIGOLD, G., GELLY, S., USZKOREIT, J., AND HOULSBY, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations* (2021).

[11] ELAD, M. *Sparse and redundant representations: from theory to applications in signal and image processing*. Springer Science & Business Media, 2010.

[12] FOURRIER, C., HABIB, N., LOZOVSKAYA, A., SZAFER, K., AND WOLF, T. Open llm leaderboard v2. `https://huggingface.co/collections/open-llm-leaderboard/open-llm-leaderboard-2`, 2024.

[13] FRANTAR, E., ASHKBOOS, S., HOEFLER, T., AND ALISTARH, D. Optq: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations* (2023).

[14] GAO, L., TOW, J., ABBASI, B., BIDERMAN, S., BLACK, S., DIPOFI, A., FOSTER, C., GOLDING, L., HSU, J., LE NOAC'H, A., LI, H., MCDONELL, K., MUENNIGHOFF, N., OCIEPA, C., PHANG, J., REYNOLDS, L., SCHOELKOPF, H., SKOWRON, A., SUTAWIKA, L., TANG, E., THITE, A., WANG, B., WANG, K., AND ZOU, A. The language model evaluation harness. `https://zenodo.org/records/12608602`, 07 2024.

[15] GONG, Y., CHUNG, Y.-A., AND GLASS, J. AST: Audio Spectrogram Transformer. In *Proc. Interspeech 2021* (2021), pp. 571–575.

[16] HENDRYCKS, D., BURNS, C., BASART, S., ZOU, A., MAZEIKA, M., SONG, D., AND STEINHARDT, J. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)* (2021).

[17] HENDRYCKS, D., BURNS, C., KADAVATH, S., ARORA, A., BASART, S., TANG, E., SONG, D., AND STEINHARDT, J. Measuring mathematical problem solving with the math dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* (2021), J. Vanschoren and S. Yeung, Eds., vol. 1.

[18] HSU, Y.-C., HUA, T., CHANG, S., LOU, Q., SHEN, Y., AND JIN, H. Language model compression with weighted low-rank factorization. In *International Conference on Learning Representations* (2022).

[19] LAI, G., XIE, Q., LIU, H., YANG, Y., AND HOVY, E. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (Copenhagen, Denmark, Sept. 2017), M. Palmer, R. Hwa, and S. Riedel, Eds., Association for Computational Linguistics, pp. 785–794.

[20] LIN, J., TANG, J., TANG, H., YANG, S., CHEN, W.-M., WANG, W.-C., XIAO, G., DANG, X., GAN, C., AND HAN, S. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems 6* (2024), 87–100.

[21] LIU, D., ZHU, Y., LIU, Z., LIU, Y., HAN, C., TIAN, J., LI, R., AND YI, W. A survey of model compression techniques: Past, present, and future. *Frontiers in Robotics and AI 12* (2025), 1518965.

[22] LIU, Y., LI, Z., HUANG, M., YANG, B., YU, W., LI, C., YIN, X.-C., LIU, C.-L., JIN, L., AND BAI, X. Ocrbench: on the hidden mystery of ocr in large multimodal models. *Science China Information Sciences 67*, 12 (Dec. 2024).

[23] MA, X., FANG, G., AND WANG, X. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems 36* (2023), 21702–21720.

[24] MACKO, V., AND BOŽA, V. MACKO: Sparse matrix-vector multiplication for low sparsity. In *Forty-third International Conference on Machine Learning* (2026).

[25] MAKHOV, D., SHOPKHOEV, D., ZHUSSIP, M., ALI, A., AND LEFKIMMIATIS, S. CoSpaDi: Compressing llms via calibration-guided sparse dictionary learning. *EMNLP Findings* (2026).

[26] MERITY, S., XIONG, C., BRADBURY, J., AND SOCHER, R. Pointer sentinel mixture models, 2016.

[27] MERITY, S., XIONG, C., BRADBURY, J., AND SOCHER, R. Pointer sentinel mixture models. In *International Conference on Learning Representations* (2017).

[28] MI, Z., SUN, B., ZHANG, G. L., AND HUANG, S. Layer-wise dynamic rank for compressing large language models. *arXiv preprint arXiv:2509.25622* (2025).

[29] MICHEL, P., LEVY, O., AND NEUBIG, G. Are sixteen heads really better than one? *Advances in neural information processing systems 32* (2019).

[30] PANAYOTOV, V., CHEN, G., POVEY, D., AND KHUDANPUR, S. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2015), pp. 5206–5210.

[31] PAPERNO, D., KRUSZEWSKI, G., LAZARIDOU, A., PHAM, N.-Q., BERNARDI, R., PEZZELLE, S., BARONI, M., BOLEDA, G., AND FERNÁNDEZ, R. 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)* (2016), pp. 1525–1534.

[32] PENEDO, G., KYDLÍČEK, H., ALLAL, L. B., LOZHKOV, A., MITCHELL, M., RAFFEL, C., WERRA, L. V., AND WOLF, T. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track* (2024).

[33] QINSI, W., KE, J., TOMIZUKA, M., KEUTZER, K., AND XU, C. Dobi-SVD: Differentiable SVD for LLM compression and some new perspectives. In *The Thirteenth International Conference on Learning Representations* (2025).

[34] RADFORD, A., KIM, J. W., HALLACY, C., RAMESH, A., GOH, G., AGARWAL, S., SASTRY, G., ASKELL, A., MISHKIN, P., CLARK, J., ET AL. Learning transferable visual models from natural language supervision. In *International conference on machine learning* (2021), PmLR, pp. 8748–8763.

[35] RADFORD, A., KIM, J. W., XU, T., BROCKMAN, G., MCLEAVEY, C., AND SUTSKEVER, I. Robust speech recognition via large-scale weak supervision. In *International conference on machine learning* (2023), PMLR, pp. 28492–28518.

[36] REIN, D., HOU, B. L., STICKLAND, A. C., PETTY, J., PANG, R. Y., DIRANI, J., MICHAEL, J., AND BOWMAN, S. R. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling* (2024).

[37] SANH, V., DEBUT, L., CHAUMOND, J., AND WOLF, T. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. In *NeurIPS 2019 Workshop on Energy Efficient Machine Learning and Cognitive Computing* (2019).

[38] SCHÖNEMANN, P. H. A generalized solution of the orthogonal procrustes problem. *Psychometrika 31*, 1 (1966), 1–10.

[39] SHOPKHOEV, D., ALI, A., ZHUSSIP, M., MALYKH, V., LEFKIMMIATIS, S., KOMODAKIS, N., AND ZAGORUYKO, S. Replaceme: Network simplification via depth pruning and transformer block linearization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems* (2025).

[40] SPRAGUE, Z. R., YE, X., BOSTROM, K., CHAUDHURI, S., AND DURRETT, G. MuSR: Testing the limits of chain-of-thought with multistep soft reasoning. In *The Twelfth International Conference on Learning Representations* (2024).

[41] SUZGUN, M., SCALES, N., SCHÄRLI, N., GEHRMANN, S., TAY, Y., CHUNG, H. W., CHOWDHERY, A., LE, Q., CHI, E., ZHOU, D., ET AL. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023* (2023), pp. 13003–13051.

[42] TANG, Y., WANG, Y., GUO, J., TU, Z., HAN, K., HU, H., AND TAO, D. A survey on transformer compression. *arXiv preprint arXiv:2402.05964* (2024).

[43] TAORI, R., GULRAJANI, I., ZHANG, T., DUBOIS, Y., LI, X., GUESTRIN, C., LIANG, P., AND HASHIMOTO, T. B. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[44] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, Ł., AND POLOSUKHIN, I. Attention is all you need. *Advances in neural information processing systems 30* (2017).

[45] VOITA, E., TALBOT, D., MOISEEV, F., SENNRICH, R., AND TITOV, I. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (Florence, Italy, July 2019), Association for Computational Linguistics, pp. 5797–5808.

[46] WANG, J., CHEN, Y.-G., LIN, I.-C., LI, B., AND ZHANG, G. L. Basis sharing: Cross-layer parameter sharing for large language model compression. In *The Thirteenth International Conference on Learning Representations* (2025).

[47] WANG, X., ALAM, S., WAN, Z., SHEN, H., AND ZHANG, M. Svd-llm v2: Optimizing singular value truncation for large language model compression. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)* (2025), pp. 4287–4296.

[48] WANG, X., ZHENG, Y., WAN, Z., AND ZHANG, M. Svd-llm: Truncation-aware singular value decomposition for large language model compression. In *The Thirteenth International Conference on Learning Representations* (2025).

[49] WELBL, J., LIU, N. F., AND GARDNER, M. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text* (Copenhagen, Denmark, Sept. 2017), L. Derczynski, W. Xu, A. Ritter, and T. Baldwin, Eds., Association for Computational Linguistics, pp. 94–106.

[50] XIAO, G., LIN, J., SEZNEC, M., WU, H., DEMOUTH, J., AND HAN, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning (ICML)* (2023).

[51] XV, L., GAO, J., GAO, X., LIU, T., AND FU, Y. Ara: Adaptive rank allocation for efficient large language model svd compression. *arXiv preprint arXiv:2510.19389* (2025).

[52] YUAN, Z., SHANG, Y., SONG, Y., YANG, D., WU, Q., YAN, Y., AND SUN, G. Asvd: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821* (2023).

[53] YUE, X., NI, Y., ZHANG, K., ZHENG, T., LIU, R., ZHANG, G., STEVENS, S., JIANG, D., REN, W., SUN, Y., WEI, C., YU, B., YUAN, R., SUN, R., YIN, M., ZHENG, B., YANG, Z., LIU, Y., HUANG, W., SUN, H., SU, Y., AND CHEN, W. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of CVPR* (2024).

[54] ZELLERS, R., HOLTZMAN, A., BISK, Y., FARHADI, A., AND CHOI, Y. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (2019), Association for Computational Linguistics.

[55] ZHOU, J., LU, T., MISHRA, S., BRAHMA, S., BASU, S., LUAN, Y., ZHOU, D., AND HOU, L. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911* (2023).

[56] ZHUSSIP, M., SHOPKHOEV, D., ALI, A., AND LEFKIMMIATIS, S. Share your attention: Transformer weight sharing via matrix-based dictionary learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2026), vol. 40, pp. 29260–29268.

A Appendix

Due to the page limitations in the main text, we present here additional results and descriptions to provide more details and evidence on the superiority of our proposed COMPOT compression method.

A.1 COMPOT Optimization Details

In this section we provide the full optimization details for COMPOT. We first describe the general calibration-aware setup and the orthogonal sparse factorization problem, then present the alternating minimization procedure, the dictionary initialization strategies used in practice, the analytical solutions for the sparse coding and dictionary update steps, and finally the convergence analysis.

A.1.1 General Setup

Recall the calibration loss (4) and the Gram matrix $\mathbf{G} = \mathbf{X}^\top \mathbf{X}$. Assuming $\mathbf{G} \succ 0$, let $\mathbf{G} = \mathbf{L}\mathbf{L}^\top$ be its Cholesky factorization. Then

$$\|\mathbf{X}(\mathbf{W} - \widehat{\mathbf{W}})\|_F^2 = \mathrm{Tr}\big((\mathbf{W} - \widehat{\mathbf{W}})^\top \mathbf{G}(\mathbf{W} - \widehat{\mathbf{W}})\big) = \|\mathbf{L}^\top(\mathbf{W} - \widehat{\mathbf{W}})\|_F^2. \tag{12}$$

Defining $\widetilde{\mathbf{W}} \triangleq \mathbf{L}^\top \mathbf{W}$ and $\widetilde{\widehat{\mathbf{W}}} \triangleq \mathbf{L}^\top \widehat{\mathbf{W}}$, minimizing functional error in the original space is equivalent to minimizing reconstruction error in the whitened space:

$$\min_{\widehat{\mathbf{W}}} \|\mathbf{X}(\mathbf{W} - \widehat{\mathbf{W}})\|_F^2 \iff \min_{\widetilde{\widehat{\mathbf{W}}}} \|\widetilde{\mathbf{W}} - \widetilde{\widehat{\mathbf{W}}}\|_F^2. \tag{13}$$

COMPOT solves the orthogonal sparse factorization in whitened coordinates:

$$\min_{\mathbf{D}_O, \mathbf{S}_O} \|\widetilde{\mathbf{W}} - \mathbf{D}_O \mathbf{S}_O\|_F^2 \quad \text{s.t.} \quad \mathbf{D}_O^\top \mathbf{D}_O = \mathbf{I}_k, \;\; \|\mathbf{s}_{\mathbf{O}j}\|_0 \le s \; \forall j \in [n], \tag{14}$$

with $\mathbf{D}_O \in \mathbb{R}^{m \times k}$, $k \le m$ (complete/undercomplete), and $\mathbf{S}_O \in \mathbb{R}^{k \times n}$.

After optimizing (14), we map the solution back to the original parameter space as in (8):

$$\widehat{\mathbf{W}} \triangleq \mathbf{A}\mathbf{S}_O, \qquad \mathbf{A} \triangleq \mathbf{L}^{-\top} \mathbf{D}_O. \tag{15}$$

The dewhitening factor $\mathbf{A}$ is computed offline during compression; inference uses only $(\mathbf{A}, \mathbf{S}_O)$.

Numerically stable whitening. When the number of calibration samples is limited, the empirical Gram matrix $\mathbf{G} = \mathbf{X}^\top \mathbf{X}$ may be singular or poorly conditioned, which can make the Cholesky-based whitening step unstable. In practice, we follow the stabilization used in the SVD-LLM implementation [48]. We first attempt the Cholesky decomposition in double precision. If it succeeds, no ridge term is added. In our calibration setup, plain Cholesky factorization failed for only $3/576$ (0.52%) activation Gram matrices, all for Qwen3-8B ($3/144$, 2.1%); no failures were observed for the remaining model and calibration-data. If it fails, we compute the smallest eigenvalue $\lambda_{\min}(\mathbf{G})$ and replace $\mathbf{G}$ by

$$\mathbf{G}_\lambda = \mathbf{G} + \big(-\lambda_{\min}(\mathbf{G}) + 7 \cdot 10^{-6}\big)\,\mathbf{I}.$$

This shifts the smallest eigenvalue to $7 \cdot 10^{-6}$ before retrying the Cholesky decomposition. The corresponding Cholesky factor is then used for whitening and dewhitening exactly as in the non-regularized case.

A.1.2 Alternating Minimization Procedure

We optimize (14) by alternating between sparse coding and dictionary update steps. With $\mathbf{D}_O$ fixed, sparse coding admits a closed-form hard-thresholding solution under the orthogonality constraint. With $\mathbf{S}_O$ fixed, the dictionary update becomes an orthogonal Procrustes problem with a closed-form SVD solution [38]. Algorithm 1 summarizes the full optimization for a single projection matrix $\mathbf{W}$ given calibration inputs $\mathbf{X}$.

Algorithm 1 COMPOT factorization for one projection matrix

Require: Weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$; calibration inputs $\mathbf{X} \in \mathbb{R}^{N\times m}$; dictionary size $k \le m$; sparsity s; number of iterations T_{it}; initialization mode $\text{init} \in \{\text{random}, \text{svd}\}$.

Ensure: Compressed factors $\mathbf{A} \in \mathbb{R}^{m\times k}$ and $\mathbf{S}_O \in \mathbb{R}^{k\times n}$ such that $\mathbf{W} \approx \mathbf{A}\mathbf{S}_O$.

1: Compute $\mathbf{G} \leftarrow \mathbf{X}^\top\mathbf{X}$ and its Cholesky factor $\mathbf{G} = \mathbf{L}\mathbf{L}^\top$.
2: Whiten weights: $\widetilde{\mathbf{W}} \leftarrow \mathbf{L}^\top\mathbf{W}$.
3: Initialize $\mathbf{D}_O^{(0)} \in \mathbb{R}^{m\times k}$ with $(\mathbf{D}_O^{(0)})^\top\mathbf{D}_O^{(0)} = \mathbf{I}_k$ using either random-column or SVD initialization.
4: **for** $t = 0, \dots, T_{\text{it}} - 1$ **do**
5: **Sparse coding (closed form):** $\mathbf{S}_O^{(t+1)} \leftarrow \mathcal{H}_s\big((\mathbf{D}_O^{(t)})^\top\widetilde{\mathbf{W}}\big)$.
6: **Dictionary update (Procrustes):** $\mathbf{M}^{(t+1)} \leftarrow \widetilde{\mathbf{W}}(\mathbf{S}_O^{(t+1)})^\top$; compute thin SVD $\mathbf{M}^{(t+1)} = \mathbf{P}^{(t+1)}\mathbf{\Lambda}^{(t+1)}(\mathbf{Q}^{(t+1)})^\top$; set $\mathbf{D}_O^{(t+1)} \leftarrow \mathbf{P}^{(t+1)}(\mathbf{Q}^{(t+1)})^\top$.
7: **end for**
8: Dewhiten dictionary: $\mathbf{A} \leftarrow \mathbf{L}^{-\top}\mathbf{D}_O^{(T_{\text{it}})}$.
9: **return** $(\mathbf{A}, \mathbf{S}_O^{(T_{\text{it}})})$.

A.1.3 Analytical Solution of the Sparse Coding Step

Fix $\mathbf{D}_O$ with $\mathbf{D}_O^\top\mathbf{D}_O = \mathbf{I}_k$. The sparse coding step for a column $\widetilde{\mathbf{w}}_j$ is

$$\min_{\mathbf{s}\in\mathbb{R}^k} \|\widetilde{\mathbf{w}}_j - \mathbf{D}_O\mathbf{s}\|_2^2 \quad \text{s.t.} \quad \|\mathbf{s}\|_0 \le s. \tag{16}$$

Let $\mathbf{z}_j \triangleq \mathbf{D}_O^\top\widetilde{\mathbf{w}}_j$. Using $\mathbf{D}_O^\top\mathbf{D}_O = \mathbf{I}_k$,

$$\begin{aligned}\|\widetilde{\mathbf{w}}_j - \mathbf{D}_O\mathbf{s}\|_2^2 &= \|\widetilde{\mathbf{w}}_j\|_2^2 - 2\,\mathbf{s}^\top\mathbf{D}_O^\top\widetilde{\mathbf{w}}_j + \|\mathbf{D}_O\mathbf{s}\|_2^2 \\ &= \|\widetilde{\mathbf{w}}_j\|_2^2 - 2\,\mathbf{s}^\top\mathbf{z}_j + \|\mathbf{s}\|_2^2 = \|\widetilde{\mathbf{w}}_j\|_2^2 + \|\mathbf{s} - \mathbf{z}_j\|_2^2 - \|\mathbf{z}_j\|_2^2. \end{aligned}\tag{17}$$

Since the first and last terms in (17) do not depend on $\mathbf{s}$, (16) is equivalent to

$$\min_{\mathbf{s}\in\mathbb{R}^k} \|\mathbf{s} - \mathbf{z}_j\|_2^2 \quad \text{s.t.} \quad \|\mathbf{s}\|_0 \le s, \tag{18}$$

i.e., Euclidean projection of $\mathbf{z}_j$ onto the set of s-sparse vectors. This projection has a closed form:

$$\mathbf{s}_O^\star = \mathcal{H}_s(\mathbf{z}_j), \qquad \mathbf{z}_j = \mathbf{D}_O^\top\widetilde{\mathbf{w}}_j, \tag{19}$$

where $\mathcal{H}_s(\cdot)$ retains the s largest entries by magnitude and sets the rest to zero; ties can be broken arbitrarily without changing optimality. Applying this column-wise yields the matrix update

$$\mathbf{S}_O \leftarrow \mathcal{H}_s\big(\mathbf{D}_O^\top\widetilde{\mathbf{W}}\big). \tag{20}$$

Thus, under orthogonality, sparse coding is solved exactly without iterative pursuit methods.

A.1.4 Analytical Solution of the Dictionary Update

For fixed $\mathbf{S}_O$, the dictionary update seeks the solution of the following problem:

$$\operatorname*{arg\,min}_{\mathbf{D}_O^\top\mathbf{D}_O=\mathbf{I}_k} \|\widetilde{\mathbf{W}} - \mathbf{D}_O\mathbf{S}_O\|_F^2. \tag{21}$$

Expanding (21), we obtain

$$\begin{aligned}\|\widetilde{\mathbf{W}} - \mathbf{D}_O\mathbf{S}_O\|_F^2 &= \|\widetilde{\mathbf{W}}\|_F^2 + \|\mathbf{D}_O\mathbf{S}_O\|_F^2 - 2\langle\widetilde{\mathbf{W}}, \mathbf{D}_O\mathbf{S}_O\rangle \\ &= \|\widetilde{\mathbf{W}}\|_F^2 + \|\mathbf{S}_O\|_F^2 - 2\langle\mathbf{D}_O, \widetilde{\mathbf{W}}\mathbf{S}_O^\top\rangle, \end{aligned}\tag{22}$$

where we used $\mathbf{D}_O^\top\mathbf{D}_O = \mathbf{I}_k$. Let

$$\mathbf{M} \triangleq \widetilde{\mathbf{W}}\mathbf{S}_O^\top \in \mathbb{R}^{m\times k}. \tag{23}$$

Dropping terms independent of $\mathbf{D}_O$, the dictionary update is equivalent to

$$\operatorname*{arg\,max}_{\mathbf{D}_O^\top\mathbf{D}_O=\mathbf{I}_k} \operatorname{Tr}(\mathbf{D}_O^\top\mathbf{M}), \tag{24}$$

which is the classical orthogonal Procrustes problem. Let $\mathbf{M} = \mathbf{P}\mathbf{\Lambda}\mathbf{Q}^\top$ be the thin SVD; then the optimizer is

$$\mathbf{D}_O \leftarrow \mathbf{P}\mathbf{Q}^\top. \tag{25}$$

A.1.5 Convergence Analysis

This section analyzes the convergence behavior of the alternating minimization procedure used in COMPOT. We first show that the sparse-coding and dictionary-update steps are exact global minimizers of their corresponding block subproblems, which implies monotone convergence of the objective values. We then provide a KL-based iterate-convergence result under standard non-degeneracy conditions that rule out support switching in hard thresholding and non-unique Procrustes updates.

Let

$$\widetilde{\mathbf{W}} \equiv \mathbf{L}^\top \mathbf{W} \in \mathbb{R}^{m\times n}, \qquad \mathrm{St}(m,k) \equiv \{\mathbf{D} \in \mathbb{R}^{m\times k} : \mathbf{D}^\top \mathbf{D} = \mathbf{I}_k\}, \tag{26}$$

and define the column-wise sparsity set

$$\mathcal{S}_s \equiv \{\mathbf{S} \in \mathbb{R}^{k\times n} : \|\mathbf{s}_j\|_0 \le s \text{ for all } j \in [n]\}. \tag{27}$$

In all main experiments, we use SVD initialization, $\mathbf{D}_O^{(0)} = \mathbf{U}_{[:,1:k]}$, where $\widetilde{\mathbf{W}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$. Hence $\mathbf{D}_O^{(0)} \in \mathrm{St}(m,k)$, and every subsequent Procrustes update preserves the Stiefel constraint. The analysis below is stated for this initialization.

The COMPOT objective is

$$\Phi(\mathbf{D},\mathbf{S}) \equiv \frac{1}{2}\|\widetilde{\mathbf{W}} - \mathbf{D}\mathbf{S}\|_F^2, \qquad \mathbf{D} \in \mathrm{St}(m,k), \quad \mathbf{S} \in \mathcal{S}_s. \tag{28}$$

Equivalently, this can be written as the nonsmooth constrained objective

$$\Psi(\mathbf{D},\mathbf{S}) = \frac{1}{2}\|\widetilde{\mathbf{W}} - \mathbf{D}\mathbf{S}\|_F^2 + \iota_{\mathrm{St}(m,k)}(\mathbf{D}) + \iota_{\mathcal{S}_s}(\mathbf{S}), \tag{29}$$

where $\iota_{\mathcal{C}}$ is the indicator function of a set $\mathcal{C}$. The feasible set is nonconvex because of both the Stiefel constraint and the ℓ_0-type sparsity constraint. Nevertheless, the two COMPOT blocks admit closed-form global minimizers.

At iteration t, COMPOT performs

$$\mathbf{S}^{(t+1)} \in \arg\min_{\mathbf{S}\in\mathcal{S}_s} \Phi(\mathbf{D}^{(t)},\mathbf{S}), \qquad \mathbf{D}^{(t+1)} \in \arg\min_{\mathbf{D}\in\mathrm{St}(m,k)} \Phi(\mathbf{D},\mathbf{S}^{(t+1)}). \tag{30}$$

The first update is implemented by hard thresholding,

$$\mathbf{S}^{(t+1)} = \mathcal{H}_s\Big((\mathbf{D}^{(t)})^\top \widetilde{\mathbf{W}}\Big), \tag{31}$$

where $\mathcal{H}_s(\cdot)$ keeps the s largest-magnitude entries in each column. The second update is the orthogonal Procrustes update: if

$$\mathbf{M}^{(t+1)} \equiv \widetilde{\mathbf{W}}(\mathbf{S}^{(t+1)})^\top = \mathbf{P}^{(t+1)}\mathbf{\Lambda}^{(t+1)}(\mathbf{Q}^{(t+1)})^\top \tag{32}$$

is a thin SVD, then

$$\mathbf{D}^{(t+1)} = \mathbf{P}^{(t+1)}(\mathbf{Q}^{(t+1)})^\top. \tag{33}$$

Lemma A.1 (exact sparse-coding update). For fixed $\mathbf{D} \in \mathrm{St}(m,k)$, the update $\mathbf{S}^+ = \mathcal{H}_s(\mathbf{D}^\top\widetilde{\mathbf{W}})$ is a global minimizer of $\Phi(\mathbf{D},\mathbf{S})$ over $\mathbf{S} \in \mathcal{S}_s$.

Proof. This follows directly from the derivation in (17)–(20): under $\mathbf{D}^\top\mathbf{D} = \mathbf{I}_k$, sparse coding is equivalent to projecting $\mathbf{D}^\top\widetilde{\mathbf{W}}$ onto the set of column-wise s-sparse matrices. This projection decouples over columns and is obtained by keeping the s largest-magnitude entries in each column, with arbitrary tie-breaking when needed. Hence $\mathbf{S}^+ = \mathcal{H}_s(\mathbf{D}^\top\widetilde{\mathbf{W}})$ is globally optimal. □

Lemma A.2 (exact orthogonal Procrustes update). For fixed $\mathbf{S}$, let $\mathbf{M} = \widetilde{\mathbf{W}}\mathbf{S}^\top$ and let $\mathbf{M} = \mathbf{P}\mathbf{\Lambda}\mathbf{Q}^\top$ be a thin SVD. Then

$$\mathbf{D}^+ = \mathbf{P}\mathbf{Q}^\top \tag{34}$$

is a global minimizer of $\Phi(\mathbf{D},\mathbf{S})$ over $\mathbf{D} \in \mathrm{St}(m,k)$.

Proof. This follows directly from the derivation in (21)–(25): for fixed $\mathbf{S}$, minimizing $\Phi(\mathbf{D}, \mathbf{S})$ is equivalent to maximizing $\langle \mathbf{D}, \widetilde{\mathbf{W}}\mathbf{S}^\top \rangle$ subject to $\mathbf{D}^\top \mathbf{D} = \mathbf{I}_k$. By the classical orthogonal Procrustes theorem and von Neumann's trace inequality, a global maximizer is $\mathbf{D}^+ = \mathbf{P}\mathbf{Q}^\top$. □

Proposition A.3 (monotone objective-value convergence). The sequence of objective values

$$\phi_t \equiv \Phi(\mathbf{D}^{(t)}, \mathbf{S}^{(t)}) \tag{35}$$

generated by COMPOT is monotonically nonincreasing and convergent.

Proof. By Lemma A.1, $\mathbf{S}^{(t+1)}$ globally minimizes the objective over the sparse-code block while $\mathbf{D}^{(t)}$ is fixed. Hence

$$\Phi(\mathbf{D}^{(t)}, \mathbf{S}^{(t+1)}) \leq \Phi(\mathbf{D}^{(t)}, \mathbf{S}^{(t)}). \tag{36}$$

By Lemma A.2, $\mathbf{D}^{(t+1)}$ globally minimizes the objective over the orthogonal dictionary block while $\mathbf{S}^{(t+1)}$ is fixed. Hence

$$\Phi(\mathbf{D}^{(t+1)}, \mathbf{S}^{(t+1)}) \leq \Phi(\mathbf{D}^{(t)}, \mathbf{S}^{(t+1)}). \tag{37}$$

Combining (36) and (37) gives

$$\Phi(\mathbf{D}^{(t+1)}, \mathbf{S}^{(t+1)}) \leq \Phi(\mathbf{D}^{(t)}, \mathbf{S}^{(t)}) \qquad \text{for all } t. \tag{38}$$

Since $\Phi(\mathbf{D}, \mathbf{S}) \geq 0$, the monotone sequence $\{\phi_t\}$ is bounded below and therefore converges. □

We next state a KL-based convergence guarantee for the iterates. Because the sparse coding map can be discontinuous when the hard-thresholding support changes, and because the Procrustes update can be non-unique when $\widetilde{\mathbf{W}}\mathbf{S}^\top$ is rank-deficient, the result is stated under the following standard non-degeneracy conditions.

Assumption A.4 (non-degenerate regime). There exists an iteration T_0 such that for all $t \geq T_0$:

1. the support pattern selected by $\mathcal{H}_s((\mathbf{D}^{(t)})^\top \widetilde{\mathbf{W}})$ is fixed, so the sparse-code iterates lie in a fixed coordinate subspace;
2. the Procrustes matrix $\mathbf{M}^{(t+1)} = \widetilde{\mathbf{W}}(\mathbf{S}^{(t+1)})^\top$ has full column rank, and its smallest singular value is uniformly bounded below: $\sigma_{\min}(\mathbf{M}^{(t+1)}) \geq \mu > 0$.

Lemma A.5 (sufficient descent under Assumption A.4). Under Assumption A.4, for all $t \geq T_0$ there exists $a > 0$ such that

$$\Psi(\mathbf{D}^{(t)}, \mathbf{S}^{(t)}) - \Psi(\mathbf{D}^{(t+1)}, \mathbf{S}^{(t+1)}) \geq a \left(\|\mathbf{S}^{(t+1)} - \mathbf{S}^{(t)}\|_F^2 + \|\mathbf{D}^{(t+1)} - \mathbf{D}^{(t)}\|_F^2 \right). \tag{39}$$

Proof. For the sparse-code step, once the support pattern is fixed, the feasible sparse-code set locally coincides with a fixed coordinate subspace. Since $\mathbf{D}^{(t)} \in \mathrm{St}(m, k)$, the sparse-coding objective can be written as

$$\Phi(\mathbf{D}^{(t)}, \mathbf{S}) = \frac{1}{2}\|\mathbf{S} - (\mathbf{D}^{(t)})^\top \widetilde{\mathbf{W}}\|_F^2 + \text{const}.$$

Thus the update $\mathbf{S}^{(t+1)}$ is the Euclidean projection onto a fixed linear subspace, and the Pythagorean identity gives

$$\Phi(\mathbf{D}^{(t)}, \mathbf{S}^{(t)}) - \Phi(\mathbf{D}^{(t)}, \mathbf{S}^{(t+1)}) \geq \frac{1}{2}\|\mathbf{S}^{(t+1)} - \mathbf{S}^{(t)}\|_F^2. \tag{40}$$

For the dictionary step, let

$$\mathbf{M}^{(t+1)} = \widetilde{\mathbf{W}}(\mathbf{S}^{(t+1)})^\top = \mathbf{P}^{(t+1)}\mathbf{\Lambda}^{(t+1)}(\mathbf{Q}^{(t+1)})^\top$$

and $\mathbf{D}^{(t+1)} = \mathbf{P}^{(t+1)}(\mathbf{Q}^{(t+1)})^\top$. For fixed $\mathbf{S}^{(t+1)}$, the objective differs from $-\langle \mathbf{D}, \mathbf{M}^{(t+1)} \rangle$ only by constants. Therefore

$$\Phi(\mathbf{D}^{(t)}, \mathbf{S}^{(t+1)}) - \Phi(\mathbf{D}^{(t+1)}, \mathbf{S}^{(t+1)}) = \langle \mathbf{D}^{(t+1)} - \mathbf{D}^{(t)}, \mathbf{M}^{(t+1)} \rangle.$$

Writing $\mathbf{R} = (\mathbf{P}^{(t+1)})^\top \mathbf{D}^{(t)} \mathbf{Q}^{(t+1)}$, we have

$$\langle \mathbf{D}^{(t+1)} - \mathbf{D}^{(t)}, \mathbf{M}^{(t+1)} \rangle = \sum_{\ell=1}^{k} \lambda_\ell^{(t+1)} (1 - R_{\ell\ell}) \geq \frac{\mu}{2} \|\mathbf{D}^{(t+1)} - \mathbf{D}^{(t)}\|_F^2,$$

where the last inequality uses $\lambda_\ell^{(t+1)} \geq \mu$ and

$$\|\mathbf{D}^{(t+1)} - \mathbf{D}^{(t)}\|_F^2 = 2 \sum_{\ell=1}^{k} (1 - R_{\ell\ell}).$$

Combining this bound with (40) proves (39) with $a = \frac{1}{2} \min\{1, \mu\}$. □

Lemma A.6 (relative-error bound under Assumption A.4). Under Assumption A.4, for all $t \geq T_0$ there exists $b > 0$ such that

$$\operatorname{dist}\Big(\mathbf{0}, \partial\Psi(\mathbf{D}^{(t+1)}, \mathbf{S}^{(t+1)})\Big) \leq b \left(\|\mathbf{D}^{(t+1)} - \mathbf{D}^{(t)}\|_F + \|\mathbf{S}^{(t+1)} - \mathbf{S}^{(t)}\|_F \right), \tag{41}$$

where $\partial\Psi$ denotes the limiting subdifferential.

Proof. The Procrustes update is an exact minimizer of $\Phi(\mathbf{D}, \mathbf{S}^{(t+1)})$ over $\mathrm{St}(m, k)$, hence the dictionary block satisfies the first-order condition

$$\mathbf{0} \in \nabla_{\mathbf{D}} \Phi(\mathbf{D}^{(t+1)}, \mathbf{S}^{(t+1)}) + N_{\mathrm{St}(m,k)}(\mathbf{D}^{(t+1)}).$$

For the sparse-code block, under the fixed-support condition in Assumption A.4, $\mathcal{S}_s$ locally coincides with a fixed coordinate subspace. Since $\mathbf{S}^{(t+1)}$ minimizes $\Phi(\mathbf{D}^{(t)}, \mathbf{S})$ over this subspace, there exists $\mathbf{N}_S^{(t+1)} \in N_{\mathcal{S}_s}(\mathbf{S}^{(t+1)})$ such that

$$\mathbf{0} = \nabla_{\mathbf{S}} \Phi(\mathbf{D}^{(t)}, \mathbf{S}^{(t+1)}) + \mathbf{N}_S^{(t+1)}.$$

Therefore, an element of $\partial\Psi(\mathbf{D}^{(t+1)}, \mathbf{S}^{(t+1)})$ is obtained by replacing $\nabla_{\mathbf{S}} \Phi(\mathbf{D}^{(t)}, \mathbf{S}^{(t+1)})$ with $\nabla_{\mathbf{S}} \Phi(\mathbf{D}^{(t+1)}, \mathbf{S}^{(t+1)})$. Since $\mathbf{D}^{(t)}, \mathbf{D}^{(t+1)} \in \mathrm{St}(m, k)$,

$$\nabla_{\mathbf{S}} \Phi(\mathbf{D}, \mathbf{S}) = \mathbf{S} - \mathbf{D}^\top \widetilde{\mathbf{W}},$$

and hence

$$\left\| \nabla_{\mathbf{S}} \Phi(\mathbf{D}^{(t+1)}, \mathbf{S}^{(t+1)}) - \nabla_{\mathbf{S}} \Phi(\mathbf{D}^{(t)}, \mathbf{S}^{(t+1)}) \right\|_F = \|(\mathbf{D}^{(t)} - \mathbf{D}^{(t+1)})^\top \widetilde{\mathbf{W}}\|_F \leq \|\widetilde{\mathbf{W}}\|_F \|\mathbf{D}^{(t+1)} - \mathbf{D}^{(t)}\|_F.$$

This proves (41) with $b = \|\widetilde{\mathbf{W}}\|_F$. □

Proposition A.7 (KL property). The constrained objective Ψ in (29) satisfies the Kurdyka–Łojasiewicz property.

Proof. The squared reconstruction term is polynomial in $(\mathbf{D}, \mathbf{S})$. The Stiefel set $\mathrm{St}(m, k)$ is defined by polynomial equalities $\mathbf{D}^\top \mathbf{D} = \mathbf{I}_k$, and the column-wise sparsity set $\mathcal{S}_s$ is a finite union of coordinate subspaces. Hence both constraint sets are semi-algebraic, and Ψ is a proper lower-semicontinuous semi-algebraic function. Proper lower-semicontinuous semi-algebraic functions satisfy the Kurdyka–Łojasiewicz property. □

Theorem A.8 (conditional KL convergence of iterates). Under Assumption A.4, the sequence $\{(\mathbf{D}^{(t)}, \mathbf{S}^{(t)})\}_{t \geq T_0}$ has finite length and converges to a critical point $(\mathbf{D}^\star, \mathbf{S}^\star)$ of Ψ. Consequently,

$$\Phi(\mathbf{D}^{(t)}, \mathbf{S}^{(t)}) \to \Phi(\mathbf{D}^\star, \mathbf{S}^\star).$$

Proof. The sequence is bounded: $\mathbf{D}^{(t)} \in \mathrm{St}(m, k)$, and the sparse-code iterates are obtained by hard thresholding $(\mathbf{D}^{(t)})^\top \widetilde{\mathbf{W}}$, hence are bounded. By Proposition A.7, Ψ satisfies the KL property. Lemma A.5 gives sufficient descent, and Lemma A.6 gives the relative-error condition. Applying the standard KL convergence theorem for nonsmooth descent methods yields finite length,

$$\sum_{t=T_0}^{\infty} \left(\|\mathbf{D}^{(t+1)} - \mathbf{D}^{(t)}\|_F + \|\mathbf{S}^{(t+1)} - \mathbf{S}^{(t)}\|_F \right) < \infty,$$

and therefore convergence of $(\mathbf{D}^{(t)}, \mathbf{S}^{(t)})$ to a limiting point $(\mathbf{D}^\star, \mathbf{S}^\star)$. The relative-error condition then implies $\mathbf{0} \in \partial\Psi(\mathbf{D}^\star, \mathbf{S}^\star)$, so the limit is a critical point. The convergence of objective values follows from continuity of Φ on the active feasible manifold and Proposition A.3. □

Discussion. Proposition A.3 holds unconditionally for the exact COMPOT updates with the SVD initialization used in the main experiments. The stronger iterate-convergence result in Theorem A.8 follows under the stated non-degeneracy assumptions. These assumptions exclude degenerate cases such as persistent hard-thresholding support switches or rank-deficient Procrustes matrices, where standard KL sufficient-descent and relative-error arguments may fail.

A.2 Dynamic Compression Ratio Allocation

Transformer models exhibit heterogeneous redundancy patterns across different layers and projection types. Uniform compression ratios often lead to suboptimal performance as sensitive layers may be over-compressed while redundant layers remain under-compressed. To address this challenge, we propose a one-shot global allocation strategy that distributes a model-wide compression budget while respecting per-matrix constraints.

Our allocation procedure operates on the singular values of weight matrices in the original, non-whitened space. In the main experiments, we normalize each weight matrix $\mathbf{W}_i$ by its Frobenius norm before computing its spectrum, $\bar{\mathbf{W}}_i = \mathbf{W}_i / \|\mathbf{W}_i\|_F$. This normalization equalizes spectral scales across layers and projection types. Consequently, the pooled singular-value rule should be interpreted as balancing relative reconstruction error across matrices, rather than as optimizing absolute Frobenius reconstruction error.

The core heuristic is simple: smaller singular values of the normalized matrices are treated as less important and are truncated first. By pooling singular values across the model, layer-specific compression ratios emerge from a single global selection rule. The global parameter budget is enforced separately by a binary search over the total truncation budget, using the rank-storage proxy $r_i(m_i + n_i)$ for a matrix of shape $m_i \times n_i$.

To prevent pathological allocation scenarios, we incorporate the following constraints:

- **Minimum compression guard**: We impose a lower bound $cr_{\min}$ on each factorized matrix by converting it into an upper bound on the retained rank. This prevents layers from receiving negligible compression.
- **Maximum compression guard**: We impose an upper bound $cr_{\max}$ by converting it into a lower bound on the retained rank. This prevents over-compression of sensitive layers.
- **Non-beneficial factorization handling**: If the feasible rank interval does not admit a parameter-saving factorization under the proxy $r_i(m_i + n_i) < m_i n_i$, the matrix is left dense and assigned $cr_i = 0$.

In all main experiments, we use unwhitened Frobenius-normalized spectra with $cr_{\min} = 0.05$ and $cr_{\max} = 0.8$. With $cr_{\min} = 0.05$, every factorized matrix is constrained to achieve at least 5% parameter reduction under the rank-storage proxy, so the DENSE fallback is inactive in our main runs; it is retained only as a safety mechanism for other configurations.

Our allocation algorithm (Algorithm 2) implements this deterministic, parameter-budgeted heuristic. First, it converts the compression guards into per-matrix rank and truncation bounds. Then, for a candidate global truncation budget, it allocates mandatory truncations induced by $cr_{\min}$ and greedily selects the smallest remaining singular values of Frobenius-normalized matrices while respecting the cap induced by $cr_{\max}$. Finally, a binary search chooses the smallest truncation budget whose implied parameter count satisfies the target model-wide compression ratio. We therefore do not claim optimality for the absolute Frobenius-error knapsack problem; the procedure is a practical heuristic for balancing relative spectral error under explicit storage constraints.

The computational complexity of our allocation strategy is dominated by the SVD computations for each weight matrix, which is $O(\sum_i \min(m_i, n_i)^2 \max(m_i, n_i))$. However, since these computations are performed only once during compression preparation and can be parallelized across layers, the practical overhead remains modest. For a Llama3.2-1B model, the entire allocation procedure completes in under 1 minute on a single A100 GPU without any optimizations.

Our one-shot approach differs from iterative methods that require multiple rounds of compression and validation. By avoiding iterative search over layer-wise compression ratios, we eliminate the need for additional calibration data passes or performance estimation heuristics, making our method both deterministic and computationally efficient.

Algorithm 2 ALLOCATE-GLOBAL: pooled-SV truncation with per-matrix CR guards

Require: Matrices $\{\mathbf{W}_i \in \mathbb{R}^{m_i \times n_i}\}_{i=1}^{N}$, target global compression ratio cr, guard bounds $(cr_{\min}, cr_{\max})$.

Ensure: Per-matrix compression ratios $\{cr_i\}$ and implied retained ranks $\{r_i\}$.

Normalize and compute spectra: $\widetilde{\mathbf{W}}_i \leftarrow \mathbf{W}_i / \|\mathbf{W}_i\|_F$; compute singular values $\sigma_{i,1} \geq \cdots \geq \sigma_{i,L_i}$, where $L_i = \min(m_i, n_i)$.

Convert CR guards to rank bounds: using the proxy storage model $r_i(m_i + n_i)$, define integer bounds

$$r_i^{\min} \approx \frac{(1 - cr_{\max}) m_i n_i}{m_i + n_i}, \qquad r_i^{\max} \approx \frac{(1 - cr_{\min}) m_i n_i}{m_i + n_i},$$

with rounding to feasible integer ranks and clipping to $[0, L_i]$.

Convert rank bounds to truncation bounds: $t_i = L_i - r_i$, giving

$$t_i^{\min} = L_i - r_i^{\max}, \qquad t_i^{\max} = L_i - r_i^{\min}.$$

Initialize all matrices as active and set the DENSE mask to false.

Let $P_0 = \sum_i m_i n_i$ and $P_{\text{tgt}} = (1 - cr) P_0$.

repeat

  Mark any active matrix as DENSE if no feasible rank satisfies $r_i(m_i + n_i) < m_i n_i$.

  Let $\mathcal{A}$ be the remaining active matrices.

  Set binary-search bounds

$$K_{\min} = \sum_{i \in \mathcal{A}} t_i^{\min}, \qquad K_{\max} = \sum_{i \in \mathcal{A}} t_i^{\max}.$$

  while $K_{\min} < K_{\max}$ **do**

    Set candidate truncation budget $\widehat{K} = \lfloor (K_{\min} + K_{\max})/2 \rfloor$.

    Allocate mandatory truncations $t_i \leftarrow t_i^{\min}$ for all $i \in \mathcal{A}$.

    From the remaining singular values of active matrices, greedily select the smallest values until the candidate budget $\widehat{K}$ is reached, while enforcing $t_i \leq t_i^{\max}$.

    Set $r_i(\widehat{K}) = L_i - t_i(\widehat{K})$ and compute

$$P(\widehat{K}) = \sum_{i \in \text{DENSE}} m_i n_i + \sum_{i \in \mathcal{A}} r_i(\widehat{K})(m_i + n_i).$$

    if any active matrix satisfies $r_i(\widehat{K})(m_i + n_i) \geq m_i n_i$ **then**

      Reclassify it as DENSE and restart the outer loop.

    else if $P(\widehat{K}) \leq P_{\text{tgt}}$ **then**

      $K_{\max} \leftarrow \widehat{K}$.

    else

      $K_{\min} \leftarrow \widehat{K} + 1$.

    end if

  end while

  Set $K^\star = K_{\max}$ and recompute $\{t_i\}$ using the same constrained greedy selection.

until no additional matrix is reclassified as DENSE

Return per-matrix ratios: for non-dense matrices, set

$$cr_i \leftarrow 1 - \frac{r_i(m_i + n_i)}{m_i n_i};$$

for dense matrices, set $cr_i \leftarrow 0$.

Table 12: Performance comparison under static (COMPOT†) and dynamic (COMPOT) vs state-of-the-art SVD-based SVD-LLM and dictionary-learning CoSpaDi on Llama3.2-1B at different compression levels on different benchmarks. Best results are shown in **bold**.

Method	CR	Accuracy↑									Perplexity↓	
		PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	WikiText	LAMBADA
Llama3.2-1B	–	74.5	63.7	63.0	60.5	36.2	88.3	37.8	37.0	57.6	1.2E+01	5.7E+00
SVD-LLM	0.2	62.1	36.4	24.4	36.0	25.1	64.9	29.0	23.0	37.6	1.7E+02	1.7E+02
CoSpaDi		66.1	42.9	38.4	39.9	26.0	71.6	31.7	24.8	42.7	6.4E+01	3.5E+01
COMPOT†		69.7	52.1	42.1	49.9	30.2	81.8	33.9	28.5	48.5	2.5E+01	2.1E+01
COMPOT		70.6	54.6	47.6	52.5	31.1	81.6	35.7	26.7	**50.1**	**2.1E+01**	**1.4E+01**
SVD-LLM	0.3	55.7	30.1	9.1	30.5	21.5	45.9	25.8	23.2	30.2	5.9E+02	2.5E+03
CoSpaDi		56.9	32.4	18.2	31.9	22.1	56.7	28.0	23.1	33.7	2.9E+02	6.6E+02
COMPOT†		64.2	41.0	27.5	39.8	26.0	73.2	31.4	23.5	40.8	5.5E+01	8.4E+01
COMPOT		67.0	46.8	37.9	47.1	27.1	77.3	33.1	23.8	**45.0**	**3.6E+01**	**3.2E+01**
SVD-LLM	0.4	51.8	27.3	1.3	26.9	22.9	32.3	24.4	23.0	26.2	1.6E+03	3.3E+04
CoSpaDi		53.5	28.2	3.8	27.8	23.0	36.9	24.0	23.1	27.5	8.0E+02	9.2E+03
COMPOT†		57.0	31.1	7.9	31.7	22.1	53.0	27.0	23.1	31.6	2.8E+02	2.2E+03
COMPOT		62.9	37.5	21.3	38.0	25.3	62.4	30.6	23.0	**37.6**	**1.3E+02**	**1.9E+02**

Table 13: Performance comparison under static (COMPOT†) and dynamic (COMPOT) vs state-of-the-art SVD-based SVD-LLM and dictionary-learning CoSpaDi on Qwen3-0.6B at different compression levels on different benchmarks. Best results are shown in **bold**.

Method	CR	Accuracy↑									Perplexity↓	
		PIQA	Hella Swag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	WikiText	LAMBADA
Qwen3-0.6B	–	67.5	47.4	40.0	55.6	34.3	83.5	33.7	40.2	50.3	2.6E+01	2.5E+01
SVD-LLM	0.2	57.0	33.0	12.6	33.6	24.8	54.9	29.5	26.1	33.9	3.1E+03	7.3E+03
CoSpaDi		59.3	37.5	18.7	37.2	27.1	68.6	32.4	26.6	38.4	1.9E+03	9.4E+02
COMPOT†		62.2	40.6	15.9	41.3	29.6	70.8	33.0	26.3	40.0	1.4E+03	4.2E+02
COMPOT		61.2	42.6	20.6	42.9	28.8	74.0	32.3	24.9	**40.9**	**7.0E+01**	**1.9E+02**
SVD-LLM	0.3	52.2	29.0	8.7	28.2	23.8	42.1	29.1	24.4	29.7	1.9E+04	1.3E+05
CoSpaDi		56.4	32.3	14.3	31.0	24.0	57.4	31.0	24.6	33.9	3.8E+03	1.1E+04
COMPOT†		58.2	34.8	13.6	34.1	27.1	59.9	31.7	26.3	35.7	3.1E+03	3.6E+03
COMPOT		61.3	38.3	20.8	40.5	27.1	65.2	32.7	27.2	**39.1**	**2.5E+02**	**3.8E+02**
SVD-LLM	0.4	53.3	26.9	3.8	26.5	23.9	32.5	25.4	23.6	27.0	7.2E+04	5.4E+06
CoSpaDi		51.6	28.2	7.9	27.3	24.1	43.8	27.0	24.0	29.2	1.2E+04	2.4E+05
COMPOT†		54.5	29.7	8.4	28.8	23.6	44.3	28.3	23.5	30.1	6.7E+03	1.2E+05
COMPOT		56.9	32.7	17.5	31.7	25.1	57.4	31.2	24.7	**34.7**	**3.8E+03**	**5.0E+03**

A.3 Comparison with Low-rank and Sparse Dictionary Learning Methods on Small Models

In this section we provide comparison of COMPOT method across different models: Qwen3-0.6B and Llama3.2-1B across different compression ratios to verify its scalability. The corresponding results are presented in Tables 13 and 12. For small models COMPOT in both static and dynamic CR cases outperforms SVD-based and dictionary learning based methods by a wide margin.

A.4 Evaluation on Recent Benchmarks

To validate our method, we report results on a set of recent and more challenging benchmarks that better reflect modern LLM evaluation. We include Open LLM Leaderboard v2 as an aggregate, community-maintained reference [12], which includes IFEval [55], MATH [17], GPQA [36], BBH [41] and MUSR [40], ensuring that observed improvements generalize beyond a single evaluation suite. See Table 14 for detailed results.

A.5 Wall-clock Time for Optimization

In Table 15 we report the wall-clock time required to compress a *single* layer of Llama 3.2–1B under a static compression ratio of 0.2 with $k/s = 2$. All measurements were obtained on a single NVIDIA A100 GPU; note that different layers are independent, so the procedure is trivially parallelizable across processes or devices.

For CoSpaDi estimation, we follow the original paper and implement K-SVD-based dictionary learning, i.e., we replace the orthogonal Procrustes (OPP) update in COMPOT with a K-SVD update. In line with CoSpaDi, we use power iterations instead of a full SVD for dictionary updates, with 60 K-SVD iterations and 8 power iterations per update. For COMPOT we use 20 alternating iterations, which is the configuration used for all main results unless otherwise stated.

Table 14: Performance comparison under static (COMPOT†) and dynamic (COMPOT) vs SVD-LLM on Open LLM Leaderboard v2 [12]

Method	CR	Accuracy↑					
		BBH	GPQA	IFEVAL	MATH HARD	MMLU Pro	MUSR
Qwen3-0.6B	–	36.85	28.27	27.58	12.69	24.26	31.35
SVD-LLM	0.2	28.95	24.75	25.3	0	11.29	35.58
COMPOT†		29.2	27.94	27.22	1.06	12.81	35.19
COMPOT		30.19	25.34	25.9	0.91	14.81	33.2
SVD-LLM	0.3	29.73	25.5	24.34	0	11.22	33.73
COMPOT†		30.83	23.49	25.06	0.23	10.85	32.8
COMPOT		29.99	26.59	27.7	0.68	11.43	35.71
SVD-LLM	0.4	29.01	24.33	22.3	0	10.98	36.24
COMPOT†		29.21	25.84	24.58	0.98	10.85	36.51
COMPOT		29.06	23.74	26.62	0.38	10.66	33.73
Qwen3-8B	–	60.84	36.33	39.21	52.49	47.62	43.12
SVD-LLM	0.2	41.03	28.27	25.66	1.06	26.3	39.81
COMPOT†		49.33	30.54	30.46	6.72	35.92	45.11
COMPOT		50.22	31.96	33.45	12.76	38.55	41.8
SVD-LLM	0.3	34.3	25.59	22.66	1.06	18.73	41.4
COMPOT†		41.45	28.86	26.74	1.51	29.46	40.61
COMPOT		44.87	28.94	31.89	2.49	32.85	40.34
SVD-LLM	0.4	30.24	22.99	22.42	0.83	11.54	37.57
COMPOT†		34.13	24.75	23.14	1.06	21.23	41.27
COMPOT		37.51	28.1	26.14	0.98	26.44	38.89

We observe that SVD-LLM is substantially faster than both CoSpaDi and COMPOT, reflecting the computational simplicity of its single truncated SVD factorization per matrix. Nevertheless, COMPOT achieves significantly lower wall-clock time than CoSpaDi: even *per iteration*, COMPOT's closed-form Orthogonal Procrustes Problem (OPP) update (thin SVD on an $m \times k$ matrix) combined with hard-thresholding sparse coding is approximately $2\times$ faster than CoSpaDi's K-SVD dictionary updates with power iterations and OMP sparse coding. The computational advantage widens further when accounting for CoSpaDi's typically higher iteration requirements.

A key insight behind this efficiency is that when the dictionary is constrained to be orthonormal (as in COMPOT), hard-thresholding yields identical sparse coding solutions to Orthogonal Matching Pursuit (OMP) in theory, while being substantially faster in practice. This equivalence holds except in rare cases involving numerical instabilities at identical coefficient magnitudes.

This computational profile highlights a fundamental advantage of orthogonal dictionary learning for large-scale transformer compression: it preserves the modeling flexibility of sparse, union-of-subspaces representations while maintaining optimization overhead comparable to SVD-style methods. The result is a favorable trade-off between compression performance and computational cost.

Table 15: Wall-clock time in seconds required for optimization with SVD-LLM, CoSpaDi and COMPOT for Llama3.2-1B with static 0.2 compression ratio.

Layer	Dimensions	SVD-LLM	CoSpaDi	COMPOT	Speedup over CoSpaDi
model.layers.0.mlp.up_proj	(2048, 8192)	1.96	508.82	17.84	28.52x
model.layers.0.mlp.down_proj	(8192, 2048)	13.99	255.51	19.23	13.29x
model.layers.0.mlp.gate_proj	(2048, 8192)	2.19	522.17	17.78	29.37x
model.layers.0.self_attn.q_proj	(2048, 2048)	1.98	154.01	5.37	28.68x
model.layers.0.self_attn.k_proj	(2048, 512)	1.03	44.70	1.99	22.41x
model.layers.0.self_attn.v_proj	(2048, 512)	0.50	44.56	1.77	25.12x
model.layers.0.self_attn.o_proj	(2048, 2048)	1.21	153.92	5.49	28.06x
AVERAGE		**3.27**	**240.53**	**9.92**	**24.23x**

A.6 Computational Efficiency and Acceleration

Inference complexity. For a projection $\mathbf{W} \in \mathbb{R}^{m\times n}$ and N input tokens, a rank-r SVD factorization requires

$$\mathcal{C}_{\text{SVD}} = Nr(m+n) \tag{42}$$

multiplications. COMPOT evaluates $\mathbf{XAS}_O$, where $\mathbf{A} \in \mathbb{R}^{m\times k}$ contains k dictionary atoms and each column of $\mathbf{S}_O \in \mathbb{R}^{k\times n}$ contains at most s nonzero coefficients, giving

$$\mathcal{C}_{\mathrm{COMPOT}} = N(mk + sn). \tag{43}$$

Here, k/s denotes the ratio between the total number of dictionary atoms and the number activated for reconstructing one weight column. When SVD and COMPOT are configured for the same value-storage budget,

$$r(m + n) = mk + sn, \tag{44}$$

and therefore their nominal multiplication counts are identical. Under the complete storage accounting of Eq. (11), the binary support mask occupies part of the COMPOT storage budget without adding arithmetic, resulting in slightly smaller factors. Thus, COMPOT and SVD have comparable theoretical arithmetic complexity at matched storage. Actual latency additionally depends on sparse metadata, memory access, batching, projection shapes, and kernel efficiency.

Practical inference efficiency. To evaluate whether this theoretical efficiency translates into practical gains, we use MACKO [24], a compressed sparse representation and GPU SpMV kernel designed for low-to-moderate unstructured sparsity and previously used for the deployment evaluation of CoSpaDi [25]. We reproduce the same setup using Llama3-8B on an NVIDIA RTX 3090, compressing only the MLP up and gate projections at $\mathrm{CR} = 0.6$ and $k/s = 2$ with the same MACKO backend. We evaluate prompt lengths of 1, 16, 32, 64, and 128 with 1, 4, 8, and 16 concurrent users. COMPOT

Table 16: End-to-end inference throughput of COMPOT relative to the original dense model and CoSpaDi on Llama3-8B using the MACKO backend. Values above 1 indicate higher throughput for COMPOT.

Prompt length	Concurrent users	COMPOT / Dense	COMPOT / CoSpaDi
1	1	1.340×	1.079×
1	4	1.373×	1.060×
1	8	1.350×	1.049×
1	16	1.136×	1.008×
16	1	1.214×	1.103×
16	4	1.201×	1.102×
16	8	1.203×	1.112×
16	16	1.163×	1.066×
32	1	1.198×	1.101×
32	4	1.223×	1.106×
32	8	1.191×	1.076×
32	16	1.182×	1.092×
64	1	1.194×	1.000×
64	4	1.202×	0.984×
64	8	1.181×	0.953×
64	16	1.188×	1.051×
128	1	1.172×	1.028×
128	4	1.199×	1.019×
128	8	1.167×	1.013×
128	16	1.142×	0.960×
Average	–	**1.211**×	**1.048**×

outperforms the dense model in all 20 configurations, with throughput gains ranging from 13.6% to 37.3%. It matches or outperforms CoSpaDi in $17/20$ configurations and improves throughput by 4.8% on average over the complete sweep. In the three configurations where CoSpaDi is faster, COMPOT remains within 4.7%. Overall, COMPOT improves throughput by 21.1% on average over dense inference. These results show that COMPOT's storage and arithmetic reduction translates into consistent end-to-end acceleration over the original dense model when paired with a kernel designed for its sparsity regime. Since COMPOT and CoSpaDi use the same dense-dictionary–sparse-coefficient inference operator, we do not attribute their runtime difference directly to orthogonality. Rather, the experiment shows that COMPOT's improved compression quality and substantially faster offline optimization are achieved without sacrificing practical deployment efficiency.

Hardware-oriented 2:4 sparsity. We additionally evaluate COMPOT with hardware-oriented 2:4 coefficient sparsity. With dynamic allocation, it achieves average accuracies of 42.92, 38.98, and 30.58 at CR $= 0.2$, 0.3, and 0.4, respectively, compared with 37.62, 30.22, and 26.21 for SVD-LLM. Thus, COMPOT retains its advantage under a hardware-oriented semi-structured sparsity constraint. Joint optimization of the dictionary size, sparsity level, layer-wise allocation, and sparse backend is left for future work.

Early stopping for compression. While the main experiments use a fixed budget of $T = 20$ alternating minimization steps, we also study an optional early-stopping mechanism to reduce compression time. The stopping rule is based on the relative Frobenius reconstruction error

$$\rho_t = \frac{\|\widetilde{\mathbf{W}} - \mathbf{D}_O^{(t)}\mathbf{S}_O^{(t)}\|_F}{\|\widetilde{\mathbf{W}}\|_F},$$

where $\widetilde{\mathbf{W}}$ is the whitened weight matrix optimized by COMPOT. Let τ denote the early-stopping tolerance and p the patience. At iteration t, we compute the relative improvement between consecutive iterations as

$$\Delta_j = \frac{\rho_{j-1} - \rho_j}{\rho_{j-1}}, \qquad j = t - p + 1, \ldots, t.$$

Early stopping is triggered if none of the most recent p intervals improves by at least τ, i.e.,

$$\max_{j \in \{t-p+1,\ldots,t\}} \Delta_j < \tau.$$

If $\rho_{j-1} = 0$, we set $\Delta_j = 0$ to avoid division by zero. In the early-stopping ablation, we use patience $p = 3$ and set the maximum number of iterations to $T_{\max} = 150$.

The results in Table 17 illustrate the influence of the relative tolerance τ on the final performance of the framework under this optional early-stopping rule. In our experimental setup, the dictionary is initialized with left singular vectors and the execution time corresponds to setup with 4 GPUs and 2 processes each in parallel. We observe that lower tolerance thresholds generally improve final accuracy and perplexity, as they allow additional alternating minimization steps before termination.

Table 17: Performance of COMPOT under varying relative tolerance (τ) thresholds for early stopping. Results are evaluated using Llama 3.2-1B at 20% compression with dictionary-to-sparsity ratio $k/s = 2.0$. The maximum number of iterations is set to 150.

Tolerance	Time (s)	Avg. Iters. per Layer							Accuracy↑									Perplexity↓	
		Up	Down	Gate	Q	K	V	O	PIQA	HellaSwag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU	Avg.	WikiText	LAMBADA
Non-compressed	–	–	–	–	–	–	–	–	74.5	63.7	63.0	60.5	36.2	88.3	37.8	37.0	57.6	11.57	5.73
$10^{-1.0}$	203.4	4.0	4.0	4.0	4.0	5.0	4.5	4.0	69.4	52.9	45.3	51.1	30.1	81.4	34.2	25.9	48.8	22.45	16.50
$10^{-1.5}$	207.1	7.6	5.0	6.3	5.0	7.3	6.3	5.9	69.9	53.5	45.8	51.4	30.9	82.1	34.8	26.1	49.3	21.94	15.76
$10^{-2.0}$	246.8	15.6	7.6	12.2	7.5	13.8	10.8	10.2	70.8	53.8	47.3	50.8	30.3	82.2	35.4	26.3	49.6	21.20	14.52
$10^{-2.5}$	342.4	32.5	13.4	25.4	14.1	28.3	19.9	19.9	70.9	54.6	48.1	51.7	30.2	81.9	34.8	26.1	49.8	20.56	13.88
$10^{-3.0}$	504.5	64.3	24.9	50.6	28.5	56.0	36.9	39.1	71.7	54.7	47.8	53.1	30.6	83.2	34.7	26.2	50.3	20.18	13.80
$10^{-3.5}$	810.5	119.6	46.2	95.3	55.6	107.6	67.9	73.4	71.5	54.9	47.6	53.4	31.7	82.6	34.3	26.2	50.3	19.93	13.79
$10^{-4.0}$	1056.9	148.8	83.7	146.7	106.4	150.0	121.5	131.6	71.7	55.2	47.6	52.6	31.7	82.1	34.1	26.5	50.2	19.86	14.06

A.7 Allocation Strategy Results

In this part we visualize the results of our allocation strategy across different models used throughout our extensive study (Figures 4 – 12)

A.8 Optimal Dictionary-to-sparsity ratio

We evaluate the performance of the proposed COMPOT framework using the Llama 3.2-1B architecture as the primary backbone. The experimental setup involves optimizing the model under varying dictionary-to-sparsity (k/s) ratios, while maintaining a fixed computational budget of $N = 100$ iterations. To ensure a robust assessment of the learning process, the dictionary is initialized randomly. Our empirical results demonstrate (see Table 18) that a k/s ratio of 2 yields the optimal balance between representation capacity and sparse regularization, achieving the highest accuracy across the evaluated benchmarks.

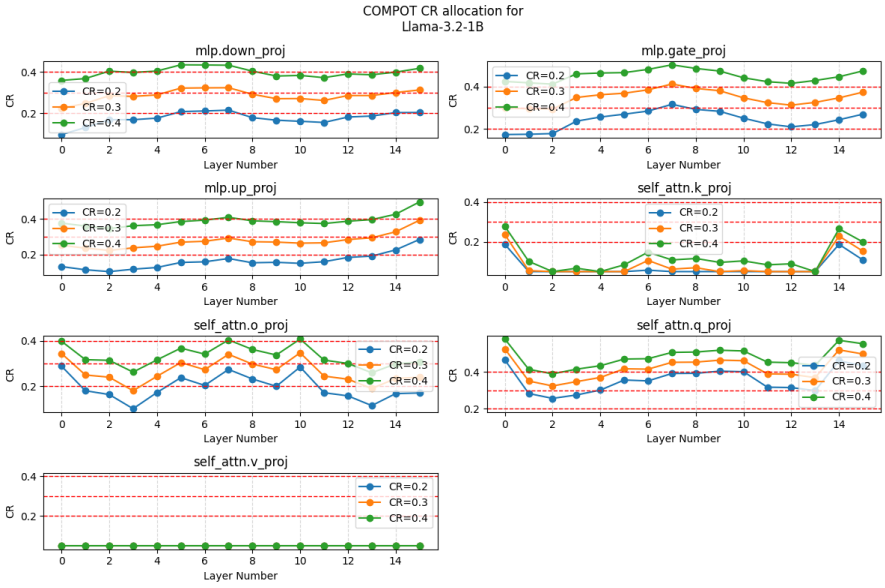

Figure 4: Results of COMPOT allocation strategy for Llama3.2-1B

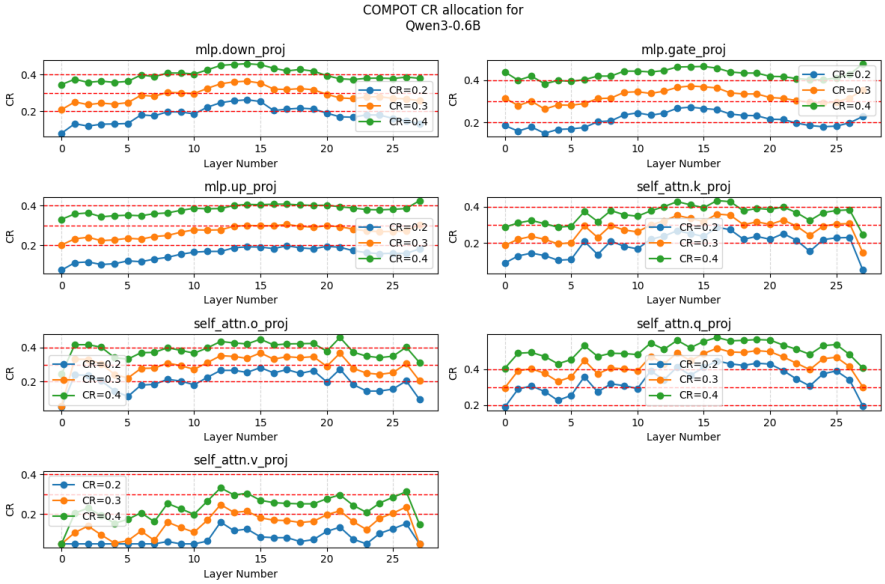

Figure 5: Results of COMPOT allocation strategy for Qwen3-0.6B

A.9 Additional results for vision and audio models

This section provides complete evaluation results for vision and audio models across various compression ratios (CR). For each benchmark, we report the corresponding CR and include the uncompressed model alongside SVD-LLM [48] as reference points. Unless otherwise noted, we adhere to the same evaluation protocol used in the main paper (employing identical prompts, decoding strategies, preprocessing, and metric computation) to ensure direct comparability across all methods.

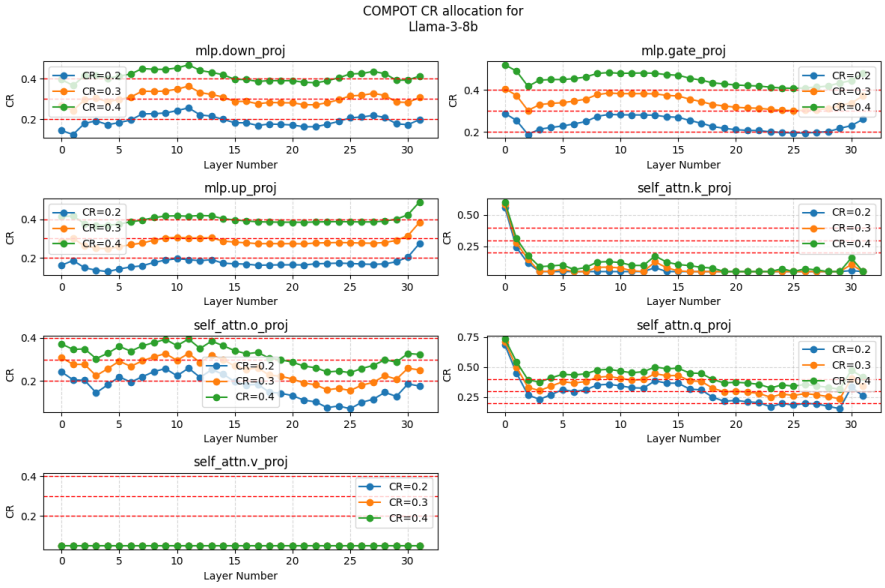

Figure 6: Results of COMPOT allocation strategy for Llama3-8B

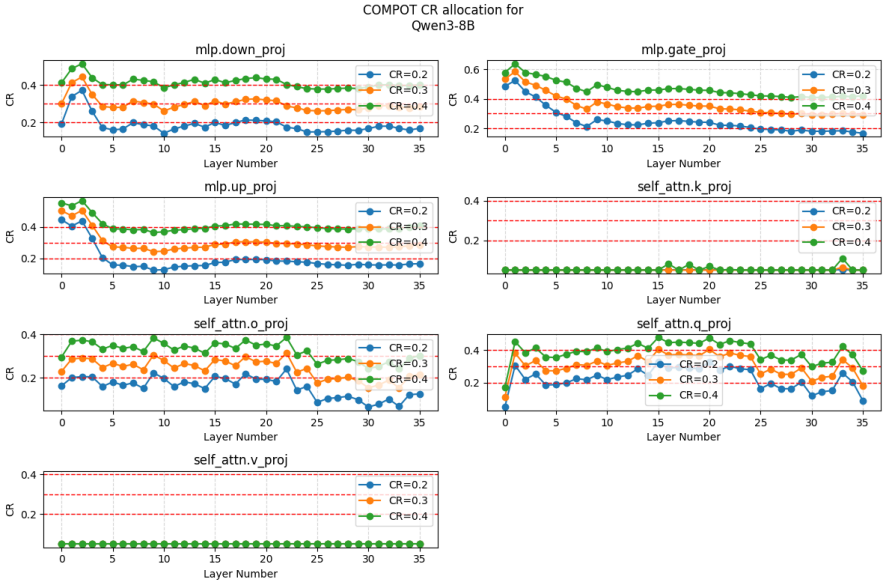

Figure 7: Results of COMPOT allocation strategy for Qwen3-8B

Vision. For the Qwen 3 VL 8B-Instruct model, we compressed only the language module using 256 samples from the MathVerse training data for calibration. Comprehensive results on four vision-language benchmarks, namely, MMMU, OCRBench, RealWorldQA, and MMStar, are presented in Table 19. Across all compression ratios, our COMPOT method consistently preserves substantially more performance than the SVD-LLM baseline.

Audio. For all Whisper models in this experiment, we compressed the decoder using 1024 samples from the LibriSpeech clean validation dataset. Table 20 reports automatic speech recognition (ASR) results, measured in word error rate (WER, lower is better), for Whisper Base-EN, Medium-EN V3,

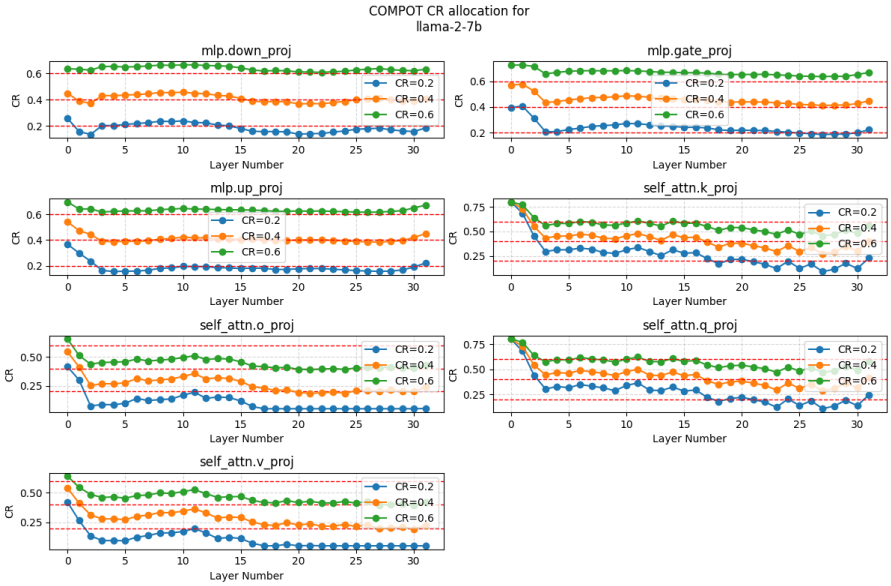

Figure 8: Results of COMPOT allocation strategy for Llama2-7B

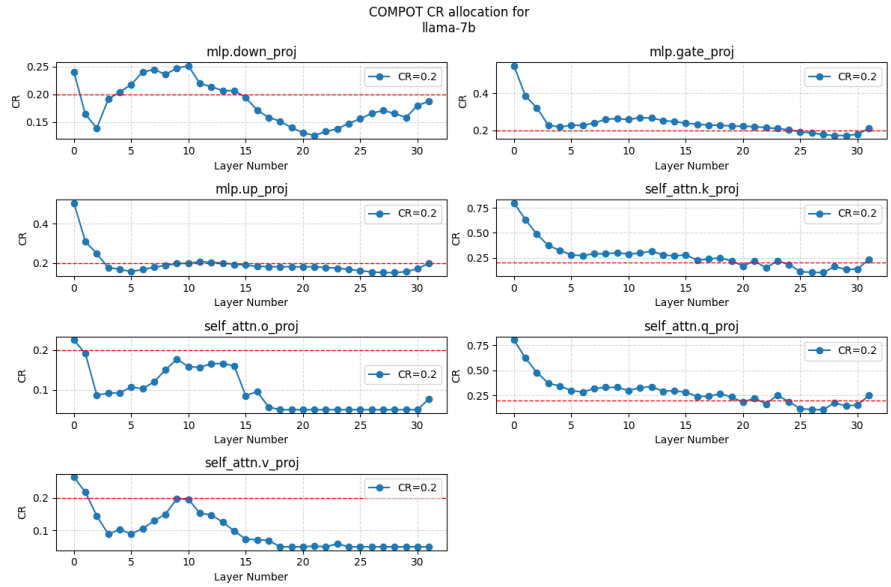

Figure 9: Results of COMPOT allocation strategy for Llama-7B

and Large V3 on the LibriSpeech test-clean and test-other sets. We evaluate multiple compression ratios where applicable. Overall, while SVD-LLM exhibits rapid performance degradation with stronger compression (particularly for smaller models), COMPOT remains close to the uncompressed baseline across a wide range of CRs. In several settings, COMPOT matches or even slightly improves upon the original model's performance.

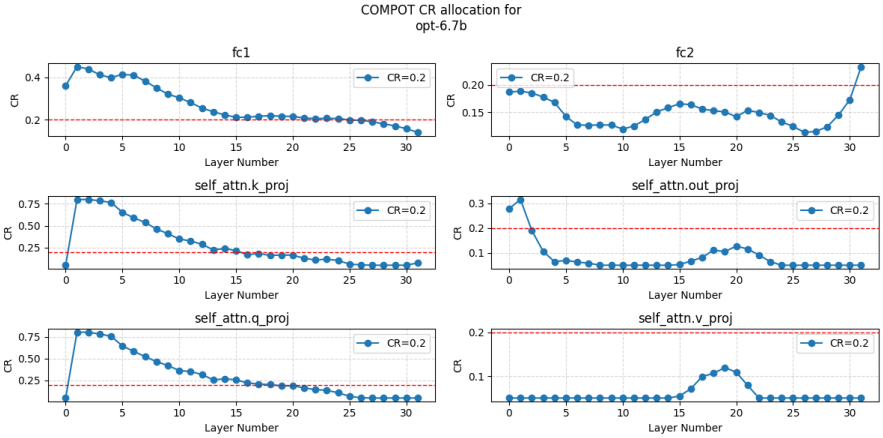

Figure 10: Results of COMPOT allocation strategy for OPT-6.7B

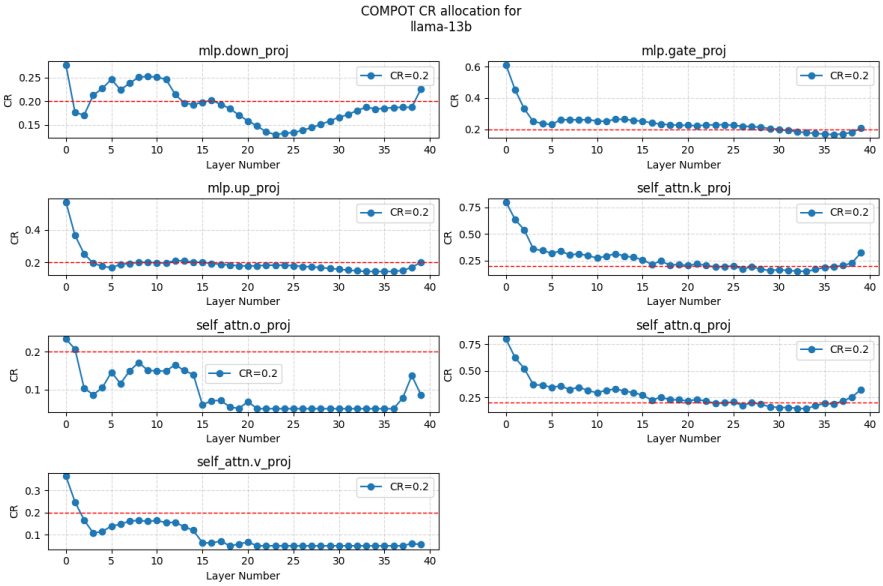

Figure 11: Results of COMPOT allocation strategy for Llama-13B

A.10 Reproducing SVD-LLM V2

Codebase and evaluation protocol. We based our reproduction on the official SVD-LLM repository[1], which provides scripts for compression and evaluation. To ensure comparability, we adopted the same evaluation protocol used by the repository and aligned our calibration setup accordingly: 256 WikiText samples with context length 2048, and perplexity evaluated using the provided scripts on WikiText and C4.

Implementing SVD-LLM V2 components. SVD-LLM V2 proposes two modifications over SVD-LLM: (i) assigning non-uniform compression ratios using theoretical truncation loss and (ii) replacing Cholesky-based truncation with a loss-optimized two-SVD procedure [47]. Since the public SVD-LLM repository does not provide a dedicated, ready-to-run SVD-LLM V2 script, we implemented these two components on top of the released SVD-LLM codebase. The implementation

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

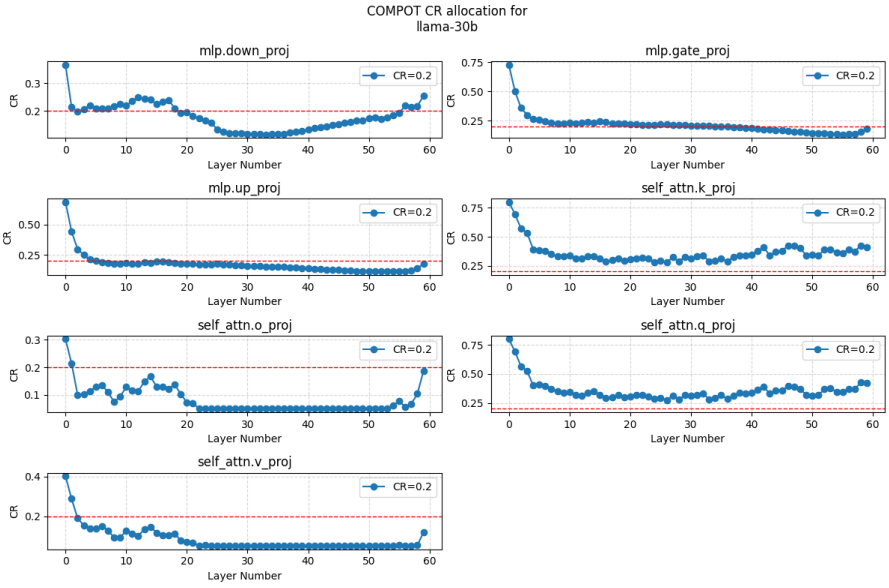

Figure 12: Results of COMPOT allocation strategy for Llama-30B

Table 18: Performance of COMPOT across different dictionary-to-sparsity (k/s) ratios. The model compression rate is fixed at 20%. All experiments are conducted using the Llama 3.2-1B architecture. Values in **bold** indicate the highest performance achieved for each respective metric.

Method	k/s	Accuracy↑								Avg.	Perplexity↓	
	ratio	PIQA	HellaSwag	LAMBADA	ARC-e	ARC-c	SciQ	Race	MMLU		WikiText	LAMBADA
Llama 3.2-1B	–	74.5	63.7	62.95	60.5	36.2	88.3	37.8	37.0	57.6	11.57	5.73
COMPOT†	1.2	64.2	40.3	30.22	39.6	25.8	72.5	31.1	23.1	40.9	73.73	83.31
COMPOT†	1.4	66.7	45.7	36.64	43.7	27.9	77.9	31.4	24.9	44.3	35.38	41.48
COMPOT†	1.6	68.8	49.1	40.69	47.3	29.2	79.9	33.4	25.8	46.8	25.29	29.75
COMPOT†	1.8	69.2	51.4	43.06	48.5	29.2	81.4	34.5	25.9	47.9	19.95	26.18
COMPOT†	2.0	69.6	52.3	41.53	48.9	30.1	80.9	34.1	27.4	**48.1**	20.52	**24.55**
COMPOT†	2.4	69.1	51.6	42.62	48.3	30.2	78.9	34.4	25.1	47.5	**18.65**	25.42
COMPOT†	2.8	68.6	50.8	39.53	47.8	30.1	79.6	33.8	25.6	47.0	23.77	26.76
COMPOT†	3.2	67.5	49.6	35.71	47.1	27.7	77.1	33.9	24.8	45.4	30.55	28.78
COMPOT†	3.6	66.2	47.3	37.47	45.6	26.8	78.6	33.1	25.1	45.0	31.01	31.86
COMPOT†	4.0	65.9	45.5	29.90	40.4	27.4	71.5	32.0	23.9	42.1	50.58	37.71

below follows the V2 paper as closely as possible while making explicit several conventions that are implicit or ambiguous in the paper pseudocode.

Compression-ratio convention. Throughout this paper, we report the compression ratio c as the fraction of parameters removed. Thus $c = 0.2$ means that approximately 80% of the original parameters are retained. By contrast, the rank formula for an SVD factorization is naturally expressed using the retained-parameter ratio

$$\rho = 1 - c.$$

For a matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, the rank used by the compressed two-layer low-rank factorization is

$$r = \left\lfloor \frac{\rho m n}{m + n} \right\rfloor.$$

Therefore, our implementation uses c only for reporting and allocation of removed parameters, while all SVD rank computations use ρ. This distinction is important: treating a compression ratio as a retained ratio reverses the intended allocation and changes the parameter budget.

Table 19: Full vision-language results for Qwen3-VL 8B under compression. We report benchmark scores on MMMU, OCRBench, RealWorldQA, and MMStar, along with the uncompressed model.

Method	CR	MMMU val ↑	OCRBench ↑	RealWorldQA ↑	MMStar ↑	Average ↑
Qwen 3 VL 8B	–	0.5256	0.8259	0.6967	0.6315	0.67
SVD-LLM	0.20	0.2922	0.3510	0.5294	0.3161	0.37
COMPOT†	0.20	0.4189	0.6410	0.5922	0.5080	0.54
COMPOT	0.20	0.4467	0.6690	0.6222	0.5417	0.57
SVD-LLM	0.30	0.2333	0.2120	0.2562	0.0398	0.19
COMPOT†	0.30	0.3378	0.5260	0.5137	0.3574	0.43
COMPOT	0.30	0.3578	0.5760	0.5098	0.4050	0.46
SVD-LLM	0.40	0.2467	0.0390	0.0000	0.0030	0.07
COMPOT†	0.40	0.2367	0.3300	0.3359	0.0576	0.24
COMPOT	0.40	0.2511	0.2630	0.4588	0.1638	0.28

Table 20: ASR Performance (WER ↓) on LibriSpeech benchmarks for Whisper family.

Method	**CR**	**WER$_{\text{test-clean}}$ ↓**	**WER$_{\text{test-other}}$ ↓**
Whisper Base-EN	-	4.40	10.40
SVD-LLM	0.05	32.37	32.18
COMPOT†	0.05	**5.41**	**12.34**
SVD-LLM	0.10	52.80	51.02
COMPOT†	0.10	**5.88**	**12.75**
SVD-LLM	0.15	85.64	83.31
COMPOT†	0.15	**8.57**	**15.30**
Whisper Medium-EN	-	2.93	5.93
SVD-LLM	0.05	3.84	7.26
COMPOT†	0.05	**2.92**	**6.06**
SVD-LLM	0.10	5.11	8.39
COMPOT†	0.10	**3.07**	**6.18**
SVD-LLM	0.15	6.95	10.14
COMPOT†	0.15	**3.12**	**6.34**
SVD-LLM	0.20	12.16	15.23
COMPOT†	0.20	**3.26**	**6.53**
SVD-LLM	0.30	32.20	33.09
COMPOT†	0.30	**4.08**	**7.78**
Whisper Large V3	-	2.74	4.53
SVD-LLM	0.2	4.12	6.8
COMPOT†	0.2	**2.46**	**4.51**
SVD-LLM	0.3	12.78	15.54
COMPOT†	0.3	**2.74**	**5.21**

```python
def retained_ratio_from_compression(compression_ratio: float) -> float:
    """
    Paper/tables report c = fraction of parameters removed.
    SVD rank computation uses rho = fraction of parameters retained.
    """
    if not (0.0 <= compression_ratio < 1.0):
        raise ValueError("compression_ratio must be in [0, 1).")
    return 1.0 - compression_ratio


def low_rank_svd_rank(W: torch.Tensor, retained_ratio: float) -> int:
    """
    Rank r for replacing W in R^{m x n} by two matrices
    U in R^{m x r} and V in R^{r x n}. The retained parameter
    count is approximately rho * m * n.
    """
    m, n = W.shape
    r = int(m * n * retained_ratio / (m + n))
    return max(1, min(r, min(m, n)))
```

Listing 1: Explicit compression-ratio and retained-ratio utilities

V2 theoretical truncation loss. The original SVD-LLM implementation computes a whitening factor using Cholesky decomposition. In contrast, SVD-LLM V2 explicitly replaces the Cholesky-

based truncation with a two-SVD construction. Given the input covariance/statistics matrix $\mathbf{S} = \mathbf{X}^\top \mathbf{X}$, we compute

$$\mathbf{S} = \mathbf{U}_s \operatorname{diag}(\sigma_s) \mathbf{V}_s^\top, \qquad \mathbf{D} = \mathbf{W}\mathbf{U}_s \operatorname{diag}(\sqrt{\sigma_s}).$$

The theoretical loss at rank r is then the Frobenius norm of the discarded singular spectrum of $\mathbf{D}$. We use this quantity in the allocation step.

```python
def theoretical_loss_v2(
    W: torch.Tensor,
    S: torch.Tensor,
    retained_ratio: float,
    dev: str = "cuda",
    eps: float = 1e-12,
) -> float:
    """
    Compute the SVD-LLM V2 theoretical truncation loss.

    W: weight matrix with shape [out_features, in_features].
    S: raw input covariance/statistics matrix X^T X with shape
       [in_features, in_features].
    retained_ratio: rho, not the reported compression ratio c.

    The loss is computed from the singular values of
        D = W U_s sqrt(S_s),
    following the loss-optimized truncation view in SVD-LLM V2.
    """
    W = W.to(device=dev, dtype=torch.float64)
    S = S.to(device=dev, dtype=torch.float64)

    # Symmetrize for numerical stability, since S is accumulated
    # from finite calibration data.
    S = 0.5 * (S + S.T)

    U_s, s_s, _ = torch.linalg.svd(S, full_matrices=False)
    s_s = torch.clamp(s_s, min=eps)

    D = W @ U_s @ torch.diag(torch.sqrt(s_s))
    singular_values = torch.linalg.svdvals(D)

    r = low_rank_svd_rank(W, retained_ratio)
    discarded = singular_values[r:]

    if discarded.numel() == 0:
        return 0.0
    return torch.linalg.norm(discarded, ord=2).item()
```

Listing 2: SVD-LLM V2 theoretical truncation loss without Cholesky

Best-effort dynamic allocation. For heterogeneous allocation, we group matrices by projection type, as described in SVD-LLM V2. For each group G, we first compute the theoretical truncation loss L_i at the target retained ratio $\rho = 1 - c$. The paper then applies an inverse-log transformation to obtain allocation weights. Since the pseudocode does not specify how to handle numerically invalid cases such as $\log(L_i) \leq 0$, we use a guarded implementation and normalize the resulting values within each projection-type group. The allocated per-matrix compression ratio is

$$c_i = |G|\, c\, \frac{\omega_i}{\sum_{j \in G} \omega_j},$$

and the retained ratio used by the SVD rank formula is

$$\rho_i = 1 - c_i.$$

The dictionary returned by the function below therefore stores retained ratios, because these are the values consumed by the low-rank SVD modules.

```python
def cr_allocation_v2(
    model_name: str,
    model: nn.Module,
    profiling_mat,
    target_compression_ratio: float,
    dev: str = "cuda",
    eps: float = 1e-12,
    min_keep_ratio: float = 0.01,
    max_keep_ratio: float = 0.99,
):
    """
    Allocate per-matrix retained ratios following the SVD-LLM V2
    heterogeneous allocation idea.

    target_compression_ratio is c, the fraction of parameters removed.
    The returned dictionary contains rho_i = 1 - c_i, i.e. retained
    ratios used in rank computation.
    """
    print("Starting SVD-LLM V2 dynamic allocation...")
    model.eval()

    target_retained_ratio = retained_ratio_from_compression(
        target_compression_ratio
    )

    if "opt" in model_name:
        layers = model.model.decoder.layers
        layer_prefix = "model.decoder.layers"
        projection_types = [
            "fc1",
            "fc2",
            "self_attn.q_proj",
            "self_attn.k_proj",
            "self_attn.v_proj",
            "self_attn.out_proj",
        ]
    else:
        layers = model.model.layers
        layer_prefix = "model.layers"
        projection_types = [
            "mlp.up_proj",
            "mlp.down_proj",
            "mlp.gate_proj",
            "self_attn.q_proj",
            "self_attn.k_proj",
            "self_attn.v_proj",
            "self_attn.o_proj",
        ]

    retained_ratio_dict = {}

    for projection_type in projection_types:
        losses = []
        names = []

        for layer_idx in tqdm(range(len(layers)), total=len(layers)):
            full_name = f"{layer_prefix}.{layer_idx}.{projection_type}"

            # HuggingFace Linear weights are [out_features, in_features].
            W = model.get_submodule(full_name).weight.data.float()

            # Use raw covariance/statistics S = X^T X, not the Cholesky
    factor.
            S = profiling_mat[layer_idx]["raw_scaling_diag_matrix"][
                projection_type
            ]

```

```
            loss = theoretical_loss_v2(
                W=W,
                S=S,
                retained_ratio=target_retained_ratio,
                dev=dev,
                eps=eps,
            )
            losses.append(loss)
            names.append(full_name)

        # The paper specifies an inverse-log transformation. We guard it
        # to avoid sign changes or division by zero when losses are close
        # to one or smaller than one.
        weights = []
        for loss in losses:
            safe_loss = max(float(loss), 1.0 + eps)
            weights.append(1.0 / max(math.log(safe_loss), eps))

        weight_sum = sum(weights)

        for full_name, weight in zip(names, weights):
            # Allocate removed-parameter fractions c_i.
            c_i = (
                len(weights)
                * target_compression_ratio
                * weight
                / weight_sum
            )

            # Convert to retained ratio rho_i for rank computation.
            rho_i = 1.0 - c_i
            rho_i = max(min_keep_ratio, min(max_keep_ratio, rho_i))
            retained_ratio_dict[full_name] = rho_i

    return retained_ratio_dict
```

Listing 3: Best-effort dynamic compression-ratio allocation with explicit ratio conversion

Loss-optimized truncation. After obtaining the per-matrix retained ratios, we compress each weight matrix using the two-SVD truncation from SVD-LLM V2. The function below returns the two factors stored in the compressed model. It also clamps small singular values of the calibration covariance before applying the inverse square root, which avoids numerical failures for rank-deficient or nearly rank-deficient calibration statistics.

```
def loss_optimized_svd_factors_v2(
    W: torch.Tensor,
    S: torch.Tensor,
    retained_ratio: float,
    dev: str = "cuda",
    eps: float = 1e-12,
):
    """
    Compute low-rank factors for SVD-LLM V2 loss-optimized truncation.

    W: dense weight with shape [out_features, in_features].
    S: raw covariance/statistics matrix X^T X.
    retained_ratio: rho_i used for this matrix.

    Returns:
        svd_u: [out_features, r]
        svd_v: [r, in_features]
    so that W is approximated by svd_u @ svd_v.
    """
    dtype = W.dtype
    W = W.to(device=dev, dtype=torch.float64)
    S = S.to(device=dev, dtype=torch.float64)

```

```
    S = 0.5 * (S + S.T)

    U_s, s_s, _ = torch.linalg.svd(S, full_matrices=False)

    # Stabilize sqrt and inverse sqrt.
    scale = torch.clamp(s_s.max(), min=1.0)
    s_floor = eps * scale
    s_s = torch.clamp(s_s, min=s_floor)

    sqrt_s = torch.sqrt(s_s)
    inv_sqrt_s = torch.rsqrt(s_s)

    D = W @ U_s @ torch.diag(sqrt_s)
    U_d, s_d, Vh_d = torch.linalg.svd(D, full_matrices=False)

    r = low_rank_svd_rank(W, retained_ratio)

    svd_u = U_d[:, :r] @ torch.diag(s_d[:r])
    svd_v = Vh_d[:r, :] @ torch.diag(inv_sqrt_s) @ U_s.T

    return svd_u.to(dtype).cpu(), svd_v.to(dtype).cpu()
```

Listing 4: Loss-optimized SVD-LLM V2 truncation used for each linear layer

Integration into the compression pass. The function above replaces the Cholesky-based whitening truncation used in the original SVD-LLM code path. In our implementation, the profiling pass still collects the same raw activation statistics, but the actual V2 truncation consumes $\mathbf{S} = \mathbf{X}^\top \mathbf{X}$ directly.

```
@torch.no_grad()
def compress_linear_layers_v2(
    model_name: str,
    model: nn.Module,
    profiling_mat,
    retained_ratio_dict,
    save_path: str,
    dev: str = "cuda",
):
    """
    Skeleton of the V2 replacement loop. The surrounding module
    replacement code follows the released SVD-LLM implementation,
    but each linear layer is decomposed using the V2 two-SVD routine.
    """
    model.eval()

    if "opt" in model_name:
        layers = model.model.decoder.layers
        layer_prefix = "model.decoder.layers"
    else:
        layers = model.model.layers
        layer_prefix = "model.layers"

    for layer_idx in tqdm(range(len(layers))):
        layer = layers[layer_idx]
        subset = find_layers(layer)

        for name, linear in subset.items():
            full_name = f"{layer_prefix}.{layer_idx}.{name}"

            W = linear.weight.data.float()
            S = profiling_mat[layer_idx]["raw_scaling_diag_matrix"][name]

            retained_ratio = retained_ratio_dict[full_name]

            svd_u, svd_v = loss_optimized_svd_factors_v2(
                W=W,
                S=S,
                retained_ratio=retained_ratio,
```

```
                dev=dev,
            )

            # The assignment to q_u_proj/q_v_proj, k_u_proj/k_v_proj,
            # etc. follows the same module-specific logic as the
            # released SVD-LLM codebase. For example:
            #
            #   svd_attn.q_u_proj.weight.data = svd_u
            #   svd_attn.q_v_proj.weight.data = svd_v
            #
            # and analogously for K, V, O, Gate, Up, and Down
            # projections.
```

Listing 5: Replacing each linear layer using V2 retained ratios

Ambiguities and fine-tuning. The SVD-LLM repository contains an optional sequential local update stage inherited from the original SVD-LLM pipeline. We did not use this stage in our SVD-LLM V2 reproduction. The V2 paper describes SVD-LLM V2 as a post-training compression method and attributes the improvement to dynamic compression-ratio allocation and loss-optimized truncation, but it does not clearly state whether any additional fine-tuning or post-compression local update was applied in the reported V2 numbers. Because the paper also reports SVD-LLM results as a baseline without fine-tuning, and because the V2 method description emphasizes post-training truncation rather than optimization with gradient updates, we conclude that no additional fine-tuning was intended for the main SVD-LLM V2 results. Accordingly, our reproduction uses calibration data only for collecting activation statistics and does not apply post-compression fine-tuning or the sequential local update step.

Implementation caveats. The published SVD-LLM V2 pseudocode leaves several numerical details underspecified: the exact convention for compression versus retained ratios, how to handle inverse-log allocation when $\log(L_i)$ is ill-conditioned or changes sign, and how to stabilize the inverse square root of the calibration covariance spectrum. We therefore treat our implementation as a best-effort reproduction rather than an official implementation. To avoid relying solely on these reproduction details, the main comparison reports paper-reported SVD-LLM V2 numbers separately from our reproduced values.

Model coverage and reproduction gaps. We attempted to reproduce the reported SVD-LLM V2 results for Llama-7B, OPT-6.7B, and Llama3-8B. For Llama-7B, our reproduction follows the released SVD-LLM code convention and gives results in the same broad range as the reported values, providing a useful sanity check for the implementation. For OPT-6.7B and Llama3-8B, we encountered model- and environment-specific difficulties in the public repository; community reports also describe failures or substantial degradations for related settings (e.g., GitHub Issues #49 and #34)[2][3]. For Llama3-8B, the run completed but produced substantially worse results than those reported in the SVD-LLM V2 paper. We therefore report our reproduction separately from the paper-reported SVD-LLM V2 numbers and use the latter as the primary reference when discussing V2 performance.

Possible sources of discrepancy. Several factors may explain the gap between reproduced and paper-reported SVD-LLM V2 results: unresolved ratio-notation ambiguity in the V2 pseudocode, numerical sensitivity of whitening/statistics inversion, model-version differences, and optional post-compression updates in the SVD-LLM repository (e.g., Issue #20)[4]. We do not use additional fine-tuning, sequential local update, or other post-compression optimization in our SVD-LLM V2 reproduction. Accordingly, all tables distinguish paper-reported SVD-LLM V2 numbers from our reproduced values. Importantly, even when compared against the paper-reported SVD-LLM V2 values rather than our reproduction, COMPOT achieves lower WikiText/C4 perplexity on two of the three models in Table 6.

[2] https://github.com/AIoT-MLSys-Lab/SVD-LLM/issues/49
[3] https://github.com/AIoT-MLSys-Lab/SVD-LLM/issues/34
[4] https://github.com/AIoT-MLSys-Lab/SVD-LLM/issues/20

Table 21: Perplexity (PPL, ↓) and average accuracy (Avg. Acc., ↑) for Llama-13B and Llama-30B.

Method	Llama-13B		Llama-30B	
	PPL	Avg. Acc.	PPL	Avg. Acc.
Original	5.09	0.59	4.10	0.61
FWSVD	15.98	0.43	20.54	0.42
ASVD	6.74	0.54	22.71	0.44
SVD-LLM	6.61	0.55	5.63	0.57
SVD-LLM v2	5.46	0.56	4.71	0.60
COMPOT	6.22	0.57	5.00	0.60

Results on larger scale. Table 21 reports results for Llama-13B and Llama-30B using the SVD-LLM evaluation protocol to enable a like-for-like comparison. COMPOT (dynamic allocation) matches or improves upon prior SVD-based and activation-aware baselines, and is comparable to the SVD-LLM V2 numbers reported in the literature. However, we treat SVD-LLM V2 as a *reference* point: a dedicated, ready-to-run V2 implementation is not publicly available, and our attempts to reproduce V2-style results at smaller scales showed sizable gaps relative to the paper-reported values. We therefore report only the paper-claimed numbers in the appendix, and interpret the V2 comparison in Table 21 accordingly.

A.11 Detailed Description on Comparison with Dobi-SVD

A direct comparison between COMPOT and Dobi-SVD requires careful methodological alignment, as the two approaches differ fundamentally in their compression paradigms. We identify two critical distinctions that limit comparability:

(1) Training-free vs. training-based compression. COMPOT operates strictly in the post-training regime without any gradient-based optimization. In contrast, Dobi-SVD employs backpropagation to optimize layer-wise ranks through a differentiable truncation objective [33]. This training step—absent in COMPOT—introduces a confounding factor that advantages Dobi-SVD through implicit fine-tuning of the compressed model, violating the core premise of training-free compression.

(2) Remapping-induced overparameterization. Dobi-SVD introduces a "remapping" operation that reparameterizes factorized layers to match the original dense matrix dimensions before quantization. Crucially, this remapping often *increases* the parameter count of the factorized representation. When matrix factorization (yielding factorization compression ratio CR_{fact}) is followed by quantization to b bits (original weights stored in 16-bit FP16), the effective model-wide compression ratio is:

$$CR_{target} = 1 - (1 - CR_{fact}) \cdot \frac{b}{16}. \tag{45}$$

For Dobi-SVD's 8-bit quantization ($b = 8$), Equation (45) simplifies to $CR_{target} = \frac{1}{2}(1 + CR_{fact})$. Critically, when remapping yields $CR_{fact} < 0$ (i.e., the factorized representation consumes *more* parameters than the original dense matrix), the net compression arises *entirely* from quantization rather than structural factorization. In such cases, Dobi-SVD effectively trades parameter inflation for quantization headroom—a strategy orthogonal to COMPOT's goal of reducing stored parameters through structural compression.

To ensure a methodologically sound comparison, we evaluate Dobi-SVD *without* remapping (denoted Dobi-SVD†), which preserves its factorization-based compression objective. Table 22 reports results on Llama2-7B under identical evaluation protocols (WikiText/C4 perplexity via SVD-LLM scripts; zero-shot accuracy via `lm-eval` 0.4.8). COMPOT consistently outperforms Dobi-SVD†across compression ratios while remaining strictly training-free.

When remapping is enabled, Dobi-SVD's gains derive predominantly from quantization rather than factorization efficiency. For instance, at target $CR_{target} = 0.2$, Dobi-SVD applies *negative* factorization compression ($CR_{fact} = -0.6$) followed by 8-bit quantization to achieve net compression—effectively overparameterizing the model by 60% to exploit quantization benefits. This paradigm shift renders comparisons against pure factorization methods like COMPOT fundamentally misaligned. We include these results in Table 22 for completeness, but emphasize that they reflect quantization efficacy rather than factorization quality.

Table 22: Comparison of COMPOT with Dobi-SVD on Llama2-7B without (denoted with †) and with remapping. We provide also CR from matrix factorization (Fact. CR) and quantization (Quant. CR) to achieve specified target CR. In most cases Dobi-SVD with remapping leads to overparametrization with matrix factorization rather than compression

Method	Target CR	Fact. CR	Quant. CR	WikiText	C4	Openb.	Arc_e	WinoG.	HellaS.	Arc_c	PIQA	MathQA	Average
Llama2 7B	—	—	—	5.49	7.27	44.0	74.6	69.0	76.3	45.5	78.6	28.4	59.5
Dobi-SVD†		0.2	—	9.39	19.70	32.8	47.6	58.5	51.5	29.3	65.2	22.7	43.9
Dobi-SVD	0.2	-0.6	0.5	5.92	8.68	42.2	72.3	69.1	72.9	44.0	78.1	27.5	58.0
COMPOT		0.2	—	6.22	9.34	30.4	72.6	67.6	51.8	39.6	76.4	26.2	52.1
Dobi-SVD†		0.4	—	17.50	55.58	29.2	34.4	54.4	35.4	24.5	57.3	21.3	36.6
Dobi-SVD	0.4	-0.2	0.5	7.89	14.23	37.2	59.4	62.7	60.6	35.0	72.1	25.5	50.4
COMPOT		0.4	—	8.91	20.40	25.2	63.1	63.5	39.6	28.5	67.0	23.6	44.4
Dobi-SVD†		0.6	—	53.37	200.11	26.0	28.3	50.5	28.4	24.7	51.9	22.0	33.1
Dobi-SVD	0.6	0.2	0.5	9.66	20.28	35.6	44.7	57.9	49.6	27.9	64.3	23.6	43.4
COMPOT		0.6	—	32.10	179.02	13.4	33.8	53.4	28.8	18.8	54.9	22.1	32.2

A.12 Calibration Robustness and Reproducibility

Number of calibration samples. We evaluate sensitivity to the calibration-set size on Llama3.2-1B at CR 0.2, using FineWeb while keeping the sequence length, dynamic allocation, SVD initialization, and 20 COMPOT iterations fixed. Results are shown in Table 23.

Table 23: Effect of the number of calibration samples on COMPOT for Llama3.2-1B at CR 0.2.

Samples	Avg. Acc.↑	WikiText PPL↓	LAMBADA PPL↓
64	49.86	21.52	14.52
128	48.83	20.76	14.01
256	50.06	20.71	13.97
512	**50.65**	**20.49**	**12.37**

Performance remains stable across relatively small calibration sets: from 64 to 256 samples, average accuracy varies within 1.23 percentage points, while perplexity changes moderately. Increasing the calibration set from 256 to 512 samples yields only a further 0.59-point accuracy improvement, supporting 256 samples as a practical quality–calibration-cost trade-off.

Sampling reproducibility. To assess reproducibility, we evaluate our compression method across 5 random seeds for data sampling on Llama3.2-1B with 20% compression. Results (Table 24) show low variance for most metrics (e.g., Avg. Acc.: 50.2 ± 0.3), indicating robustness to sampling variation. Lambada perplexity exhibits higher sensitivity (13.51 ± 0.63), likely due to its dependence on long-range context; however, all runs remain within a competitive range relative to baselines.

Table 24: Results across different data sampling seeds. We report mean ± standard deviation over 5 independent runs.

Seed	Avg. Acc.↑	Perplexity↓	
		WikiText	LAMBADA
2023	50.1	20.71	13.97
43	50.2	20.51	13.08
44	50.7	20.56	12.67
45	50.0	20.56	14.20
46	49.9	20.77	13.65
Mean ± Std	**50.2 ± 0.3**	**20.62 ± 0.11**	**13.51 ± 0.63**
Min–Max	**49.9–50.7**	**20.51–20.77**	**12.67–14.20**

A.13 Hardware setup

For all experiments if not specified otherwise ran on a server with 256-core AMD EPYC 7742 CPU and 4× NVIDIA A100-SXM4-40GB GPUs. In some cases we utilized only a single GPU (e.g. for time measurements)

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the paper's main contributions and scope: a training-free Transformer compression method based on orthogonal dictionary factorization, closed-form Procrustes and sparse-coding updates, one-shot dynamic compression allocation, and compatibility with post-training quantization. These claims are supported by method description and experimental results in the main text, and additional evaluations and ablations in the Appendix.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses limitations in Section 5, including dependence on representative calibration data, sensitivity of the whitening step to ill-conditioned Gram/covariance matrices, and the current use of a fixed sparsity pattern. The section also outlines possible extensions such as stabilized whitening and learning the sparsity structure.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The theoretical results are stated with their assumptions in Appendix A.1.5: the dictionary is constrained to the Stiefel set, sparse codes are column-wise s-sparse, and both subproblems are solved exactly. The appendix provides complete proofs for the closed-form sparse-coding update, the orthogonal Procrustes dictionary update, and monotone convergence of the COMPOT objective with corresponding references to the prior art.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides the main information needed to reproduce the experimental results in Section 4 and the appendix, including model families, compressed projection types, compression ratios, calibration datasets and sequence lengths, evaluation benchmarks, metrics, baseline protocols, implementation details, allocation procedures, and hyperparameter choices.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.

(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code is submitted with the paper, together with instructions for reproducing the main experimental results. The experiments use publicly available models, calibration datasets, and evaluation benchmarks, with data sources and experimental protocols described in Section 4 and the appendix.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the experimental setup in Section 4 and the appendix, including calibration datasets, sequence lengths, compressed projection types, compression ratios, evaluation benchmarks, metrics, and key hyperparameters such as dictionary size, sparsity ratio, initialization, and number of alternating updates. Since COMPOT is training-free, no optimizer or training schedule is used.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.

- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports statistical variability in the appendix using repeated COMPOT runs with different random seeds. We provide the corresponding mean and standard deviation for the relevant metrics, enabling assessment of run-to-run variability and supporting the statistical reliability of the reported improvements.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports the compute setup and runtime information needed to reproduce the experiments in the appendix, including GPU type, memory-relevant model scales, wall-clock compression time, and implementation details for the main compression runs. The reported timing experiments describe the hardware used and the per-layer optimization time, while the experimental setup specifies the models, calibration sizes, and evaluation protocols.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: The research conforms to the NeurIPS Code of Ethics. The paper focuses on post-training compression of existing Transformer models, uses publicly available models and datasets, and does not involve human-subject experiments, private data, or data collection from vulnerable populations.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The paper discusses broader impacts in Section 5. It notes potential positive impacts from reducing memory, compute, deployment cost, and energy use, which can improve accessibility. It also discusses possible negative impacts from lowering the barrier to deploying powerful models in settings where risks of the underlying models remain, including bias, privacy leakage, misinformation, or misuse, and recommends using compressed models with appropriate safeguards and monitoring.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release new pretrained models, generated content, scraped datasets, or other high-risk assets. The released code implements a compression method for existing models, and any compressed models inherit the usage restrictions, safeguards, and licenses of the corresponding base models and datasets.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper credits the original creators of the publicly available models, datasets, benchmarks, and baseline codebases used in the experiments through citations in the main text and appendix. All assets are used according to their corresponding licenses and terms of use. Any compressed model produced by COMPOT is a derivative of the corresponding base model and therefore inherits the original model's license, usage restrictions, and attribution requirements.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper introduces and releases the COMPOT code as a new asset. The submitted code package includes documentation for running the method, reproducing the main experiments, key hyperparameters, expected inputs and outputs, and licensing information. The method does not introduce new datasets or pretrained base models; compressed models are derivatives of existing models and inherit their corresponding licenses and usage restrictions.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, human-subject experiments, user studies, or new human data collection.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, human-subject experiments, user studies, or new human data collection.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

Justification: The core method development in this research does not use LLMs as an important, original, or non-standard component. Any LLM use, if applicable, was limited to writing, editing, or formatting assistance and did not affect the methodology, experiments, scientific rigor, or originality of the work.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.