DictLLM: Post-training Compression of Large Language Model with Dictionary Kernels

Jinho Park
Columbia University
jp4327@columbia.edu

Se Young Chun
Seoul National University
sychun@snu.ac.kr

Mingoo Seok
Columbia University
ms4415@columbia.edu

jp4327.github.io/dictllm/

Abstract

Efficient deployment of large language models (LLMs) on mobile devices requires balancing performance with compute and memory constraints. Post-training compression is an effective approach to managing this trade-off, while requiring much less computation than in-training counterparts. We propose DictLLM, a dictionary-based compression scheme in which weight matrices within grouped layers are decomposed into a shared dictionary and layer-specific sparse coefficients, directly obtained from a pre-trained model without further training. Specifically, we cast DictLLM as a four-step post-training optimization. First, we obtain shared dictionaries via singular value decomposition (SVD) of input-scaled weight matrices, amortizing rank across grouped layers; second, we sparsify the layer-specific coefficients to the target compression ratio using a dictionary-aware Hessian; and third, we refine the dictionaries given the resulting sparse coefficients. Finally, we aggressively quantize both components using factor-specific Hessian information: input covariance for the dictionaries and the dictionary-aware Hessian for the coefficients. The resulting representation reduces memory footprint by storing a single shared dictionary per layer group, along with sparse layer-specific coefficients. On language modeling and zero-shot reasoning benchmarks, DictLLM consistently outperforms existing post-training low-rank compression methods and pushes the Pareto frontier between model size and performance. DictLLM compresses the original LLaMA-7B model by 5.1× with only a 2.9-point increase in WikiText-2 perplexity, achieving a 3.6× higher compression ratio than the state-of-the-art low-rank method.

1 Introduction

Deploying large language models (LLMs) on mobile hardware is fundamentally limited by memory, not computation [1–3]. Their billions of parameters exceed device capacity and dominate energy cost through repeated weight memory transfers. Mobile devices do not have enough memory to store these models, and loading parameters to and from memory consumes significantly more energy than computations required for attention and feedforward (FFN) operations [1].

In light of this, LLM compression has emerged as a promising approach for deploying LLMs on mobile graphics processing units (GPUs) with limited memory. In particular, post-training compression has attracted significant research attention, as it requires far fewer computational resources than its in-training counterparts. Model compression offers two key benefits: (1) it enables deployment of state-of-the-art LLMs on a mobile device, and (2) it alleviates memory-bandwidth bottlenecks, as characterized by the roofline model [3, 4]. While compression also reduces the number of multiply–accumulate (MAC) operations, the dominant performance gains typically arise from reduced memory traffic rather than decreased computation [4, 5].

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

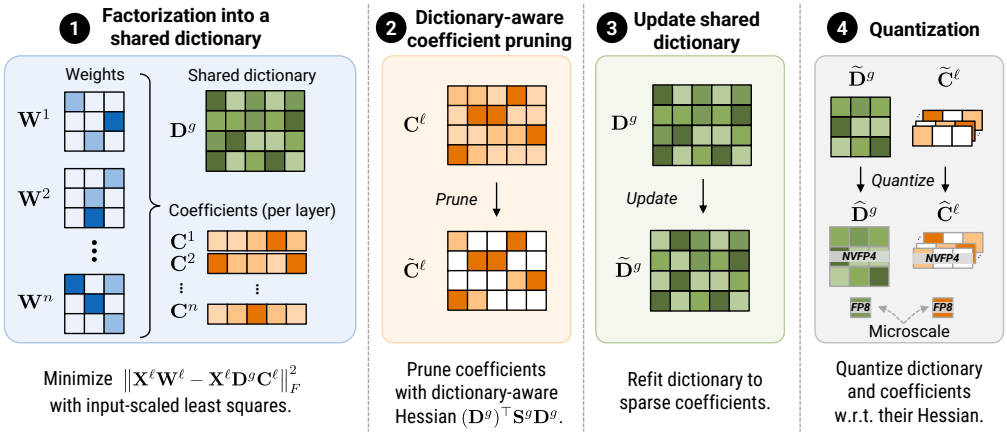

Figure 1: **DictLLM: Dictionary-based compression method.** DictLLM resolves the weight memory bottleneck by compressing transformer weights into a shareable dictionary. It compounds four compression steps. ❶ The first step factorizes input-scaled weight matrices of grouped layers into a shared dictionary and layer-specific coefficients. ❷ We then prune the layer-specific coefficients using a dictionary-aware Hessian. ❸ We refit the shared dictionary to the resulting sparse coefficients. ❹ We quantize both the dictionary and the coefficients into NVFP4 using input statistics for the dictionary and dictionary-aware Hessians for the coefficients. The resulting compressed representation enables efficient projections by transferring a small set of shared dictionaries.

Post-training model compression methods have sought to reduce model footprint along three independent axes: low-rank approximation shrinks matrix dimensions, pruning removes individual connections, and quantization lowers per-element bit width. Low-rank approximation uses singular value decomposition (SVD) [6] to factorize weight matrices into smaller ones [7–12]. However, meaningful compression requires the retained rank to be substantially smaller than the original matrix dimensions. Since non-negligible spectral components can remain outside the retained subspace, aggressive low-rank approximation often degrades performance. On the other hand, pruning [13] allows sparsification of weight matrices while preserving their shape [14, 15]. Since neural processing units (NPUs) and GPUs support sparse matrix-matrix multiplication, pruned models can benefit from improved energy efficiency. Orthogonal to these methods, quantization [16–23] reduces the bit width of each element while keeping the number of model parameters constant.

Parameter sharing [24–27] takes a different approach by reusing attention and feedforward weights across layers, enabling deeper models without additional parameters. On the other hand, dictionary-based transformers relax strict sharing by representing weights as combinations of shared atoms and layer-specific coefficients [25, 28]. This idea is closely related to classical dictionary learning, where algorithms such as the Method of Optimal Directions (MOD) [29] and k-SVD [30] reconstruct signals such as images from an overcomplete dictionary and sparse representation coefficients. Although parameter sharing approaches can greatly reduce the number of parameters and memory traffic, they often require training the shared representation from scratch. This in-training requirement of the method makes it difficult to apply directly as a post-training compressor for billion-scale LLMs.

Existing post-training methods [7–12, 14–21] and sharing-based architectures [24–26] therefore exploit complementary axes of redundancy, but they are typically applied in isolation. A more effective post-training compressor should combine these benefits by sharing a reusable structure across layers while preserving layer-specific adaptation. However, stacking these methods is nontrivial since each compression stage changes the weight and activation statistics used by the next stage, and a naïve composition can severely degrade performance.

We propose DictLLM, a post-training compression method that jointly achieves layer sharing, Hessian-aware sparsification, and 4-bit block floating-point (FP4) quantization via a single shared dictionary and sparse layer-specific coefficients within grouped layers (Fig. 1). We develop a 3+1 approach that shows shared dictionary-based compression can be constructed via a three-step post-training process: input-scaled factorization, dictionary-aware Hessian coefficient pruning, and dictionary refitting. This distinguishes our approach from prior work that constructs dictionary-based compression from scratch [25] or requires principal component-based analysis and fine-tuning [31]. In the last step, we show that the compressed model remains amenable to FP4 [16] quantization.

We highlight our contributions as follows:

- We propose dictionary-based parameter sharing, which decomposes large weight matrices into shared dictionary kernels and layer-wise sparse representation coefficients. Our zero-shot *post-training* compression method enables compression of a billion-parameter LLM.
- We combine activation-aware dictionary factorization and dictionary-induced Hessian pruning of the coefficients, then refit the shared dictionary to the sparse representation coefficients, improving reconstruction without changing sparsity or model size.
- We perform Hessian-aware FP4 quantization using the input covariance for the dictionary and the dictionary-based saliency for the coefficients.
- DictLLM reduces the original LLaMA-7B size by 5.1× with only a 2.9-point increase in WikiText-2 perplexity, improving over the state-of-the-art low-rank method by 3.6×.

2 Related works

2.1 Post-training model compression

Low-rank compression. Low-rank approximation is a natural approach for compressing transformer weights because attention and feedforward layers are dominated by large matrix multiplications. Existing SVD methods reduce each weight matrix into two smaller factors, but aggressive rank truncation often causes substantial degradation in LLMs. SVD-LLM [8] improves low-rank compression by using truncation-aware decomposition and activation information. ASVD [11] and Dobi-SVD [10] also use activation-aware scaling to reduce the error from low-rank approximation.

Pruning. Pruning reduces model size by removing redundant weights while preserving the original matrix shapes. Early pruning methods used magnitude or second-order saliency criteria to identify parameters to remove [13, 32]. For LLMs, SparseGPT [14] introduced an efficient post-training pruning method that uses calibration activations as a Hessian proxy and compensates for pruning errors in a blockwise manner. Wanda [15] showed that combining weight magnitude with activation statistics can provide an effective pruning criterion. These methods directly prune dense weight matrices [33]. In contrast, our method prunes the layer-specific coefficient matrices after dictionary factorization and projects the activation covariance into the coefficient space induced by the shared dictionary.

Low-bit quantization. Quantization reduces memory footprint and bandwidth by representing model weights with fewer bits [18, 19, 21, 34]. Post-training quantization methods, such as GPTQ [19], use calibration activations to compensate for quantization errors without retraining. Recent hardware-oriented formats further push weight precision below INT8 by using floating-point quantization with shared scales. NVFP4 [16] represents groups of FP4 values with micro-scaling factors, enabling efficient low-bit inference on modern accelerators. Microscaling FP4 methods have shown that careful scaling and error compensation are important for closing the gap between theoretical compression and practical accuracy [17]. DictLLM quantizes both the dictionary and the coefficients with NVFP4. It compensates for dictionary quantization errors using the input covariance and for coefficient quantization errors using the dictionary-induced Hessian.

2.2 Parameter sharing and dictionary learning

Parameter sharing. Parameter sharing reduces model size by reusing weights across layers or recurrent computation steps. ALBERT [24] shares transformer parameters across layers, showing that substantial parameter reduction is possible without changing the high-level architecture. Deep equilibrium models [35] take a related perspective by replacing explicit layer stacks with a recurrent computation that converges to a fixed point. More recent methods relax strict sharing by adding layer-specific residual or low-rank components [26, 36–39].

Basis Sharing [9] further improves the compression trade-off by sharing basis matrices across layers instead of factorizing each layer independently. DictLLM builds on this direction by recovering shared dictionaries via activation-aware factorization, while further sparsifying the layer-specific coefficients using a dictionary-aware Hessian and quantizing both factors with NVFP4.

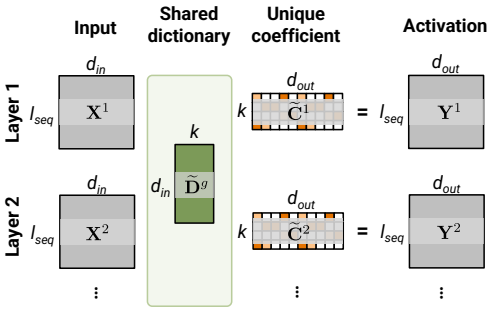

Figure 2: **Dictionary-based forward operation.** Each weight matrix $\mathbf{W}^\ell$ is replaced by the product of a shared dictionary $\widetilde{\mathbf{D}}^g$ and a layer-specific sparse coefficient matrix $\widetilde{\mathbf{C}}^\ell$. The forward pass computes $\mathbf{Y}^\ell = \mathbf{X}^\ell \widetilde{\mathbf{D}}^g \widetilde{\mathbf{C}}^\ell$.

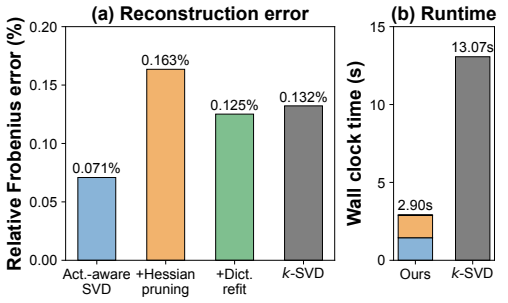

Figure 3: **Motivation for staged dictionary refinement.** We compare (a) activation reconstruction error and (b) runtime of our factorize–prune–update pipeline against k-SVD on a single LLaMA-7B query projection.

Dictionary learning. Dictionary-based models represent weights or signals as combinations of shared atoms and sparse coefficients. This idea is rooted in classical dictionary learning, where methods such as MOD [29] and k-SVD [30, 40] alternate between sparse coding and dictionary updates to reconstruct signals from an overcomplete basis. DictFormer [25] applies this idea to transformers and substantially reduces model size by sharing dictionary atoms across layers. T-REX [28] further shows that keeping the shared dictionary on chip can reduce external memory access and improve energy efficiency. However, existing dictionary-based transformer approaches typically require training the dictionary representation from scratch, making them difficult to apply to billion-scale pretrained LLMs. DictLLM instead constructs a dictionary representation, prunes coefficients with a dictionary-induced Hessian, and refits the dictionary to the sparse coefficients.

3 Proposed methods

3.1 Shared dictionary representation

We first describe the shared-dictionary representation and the forward operation. For a layer group g with layers $\ell \in g$ that share a dictionary, we define input dimension d_{in}, output dimension d_{out}, and dictionary size k (Fig. 2). A shared dictionary $\widetilde{\mathbf{D}}^g \in \mathbb{R}^{d_{\text{in}} \times k}$ is reused by every layer in the group, while the coefficient matrix $\widetilde{\mathbf{C}}^\ell \in \mathbb{R}^{k \times d_{\text{out}}}$ of each layer is not shared. The original weight matrix $\mathbf{W}^\ell \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ of layer ℓ is approximated column-wise as

$$\widetilde{\mathbf{W}}^\ell_{:,j} = \sum_{m \in \mathcal{S}^\ell_j} \widetilde{\mathbf{D}}^g_{:,m} \, \widetilde{\mathbf{C}}^\ell_{m,j}, \tag{1}$$

where $\mathcal{S}^\ell_j \subseteq [k]$ denotes the indices of active dictionary atoms. The coefficient sparsity is defined as $p = 1 - |\mathcal{S}^\ell_j|/k$. Equation 1 can be written in matrix form as $\widetilde{\mathbf{W}}^\ell = \widetilde{\mathbf{D}}^g \widetilde{\mathbf{C}}^\ell$, where $\widetilde{\mathbf{C}}^\ell$ is sparse.

3.2 Dictionary-based compression objective

We compute the dictionary and sparse coefficients sequentially because joint optimization would require solving a large discrete-continuous problem over shared factors and sparsity masks. DictLLM solves an activation-weighted reconstruction problem,

$$\min_{\mathbf{D}^g, \{\mathbf{C}^\ell\}_{\ell \in g}} \sum_{\ell \in g} \left\| \mathbf{X}^\ell \mathbf{W}^\ell - \mathbf{X}^\ell \mathbf{D}^g \mathbf{C}^\ell \right\|_F^2 \tag{2}$$

where $\| \cdot \|_F$ denotes the Frobenius norm. Algorithm 1 delineates the decomposition steps. We first decompose grouped model weights into a shared dictionary $\mathbf{D}^g$ and dense layer-specific coefficients $\{\mathbf{C}^\ell\}_{\ell \in g}$ using activation-aware low-rank factorization. Next, we sparsify the coefficients using a dictionary-aware Hessian criterion with the dictionary fixed. The final step refits the shared dictionary based on the resulting coefficients.

Algorithm 1: Dictionary-based compression

Input: Pretrained weights $\{\mathbf{W}_t^\ell\}_{\ell=1}^L$, grouped layer index sets $\mathcal{G}$, calibration set $\mathcal{X}$, sparsity p

Output: Compressed parameters $\{\widehat{\mathbf{D}}, \widehat{\mathbf{C}}\}$

$\{\mathbf{S}_t^\ell\}_{\ell,t} \leftarrow \mathrm{CollectCovariance}(f_{\mathbf{W}}, \mathcal{X})$;
foreach *weight type* t **do**
  foreach *layer group* $g \in \mathcal{G}^t$ **do**
    $\mathcal{W}_t^g \leftarrow \{\mathbf{W}_t^\ell : \ell \in g\}$;
    $\mathbf{S}_t^g \leftarrow \sum_{\ell \in g} \mathbf{S}_t^\ell$;
    $k_t^g \leftarrow \mathrm{Rank}(\mathcal{W}_t^g)$;
    $\mathbf{D}_t^g, \{\mathbf{C}_t^\ell\}_{\ell \in g} \leftarrow \mathrm{Factorize}(\mathcal{W}_t^g, \mathbf{S}_t^g, k_t^g)$;
    $\mathbf{H}_t^{\mathbf{C},g} \leftarrow (\mathbf{D}_t^g)^\top \mathbf{S}_t^g \mathbf{D}_t^g$;
    foreach $\ell \in g$ **do**
      $\widetilde{\mathbf{C}}_t^\ell \leftarrow \mathrm{Prune}(\mathbf{C}_t^\ell, \mathbf{H}_t^{\mathbf{C},g}, p)$;
    $\widetilde{\mathbf{D}}_t^g \leftarrow \mathrm{Refit}(\mathcal{W}_t^g, \{\widetilde{\mathbf{C}}_t^\ell\}_{\ell \in g})$;
    $\widehat{\mathbf{D}}_t^g \leftarrow \mathrm{Quantize}(\widetilde{\mathbf{D}}_t^g, \mathbf{S}_t^g)$;
    foreach $\ell \in g$ **do**
      $\widehat{\mathbf{C}}_t^\ell \leftarrow \mathrm{Quantize}(\widetilde{\mathbf{C}}_t^\ell, \mathbf{H}_t^{\mathbf{C},g})$;

Algorithm 2: Dictionary-aware coefficient pruning

Input: Coefficient matrix $\mathbf{C}$, dictionary $\mathbf{D}$, input covariance $\mathbf{S}$, sparsity p, damping λ, block size B

Output: Pruned coefficient matrix $\widetilde{\mathbf{C}}$

$\mathbf{H}^{-1} \leftarrow (\mathbf{D}^\top \mathbf{S} \mathbf{D} + \lambda \mathbf{I})^{-1}$;
$\mathbf{d} \leftarrow \mathrm{diag}(\mathbf{H}^{-1})$;
$\widetilde{\mathbf{C}} \leftarrow \mathbf{C}$;
for $j \leftarrow 0,\ B,\ 2B,\ \ldots$ **do**
  $\mathbf{R}_{:,\,j:j+B} \leftarrow \mathrm{SaliencyMap}(\widetilde{\mathbf{C}}_{:,j:j+B}, \mathbf{d})$;
  for $i \leftarrow 0$ **to** $\lfloor p \cdot k \rfloor - 1$ **do**
    $\mathbf{c} \leftarrow \mathbf{R}_{i,\,j:j+B}$;
    `// Least salient rows`
    $\mathbf{e} \leftarrow \widetilde{\mathbf{C}}_{[\mathbf{c},\,j:j+B]} \odot \mathbf{d}_{[\mathbf{c}]}^{-1}$;
    `// Corresponding error`
    $\widetilde{\mathbf{C}}_{:,\,j:j+B} \leftarrow \widetilde{\mathbf{C}}_{:,\,j:j+B} - \mathbf{H}^{-1}_{:,[\mathbf{c}]}\,\mathrm{diag}(\mathbf{e})$;
  $\widetilde{\mathbf{C}}_{:,\,j:j+B} \leftarrow \widetilde{\mathbf{C}}_{:,\,j:j+B} \odot \mathbf{M}_{:,\,j:j+B}$
    `// Apply pruning`

Motivation for staged dictionary refinement. To motivate our post-training approximation, we compare our method, consisting of activation-aware SVD, Hessian coefficient pruning, and post-pruning dictionary update, against k-SVD with an undercomplete dictionary ($k < d_{\text{in}}$) on the first layer of LLaMA-7B query projection (Fig. 3). k-SVD is an alternative to our sequential pipeline for learning a sparse dictionary representation from scratch. However, full sparse dictionary learning is expensive at the LLM scale because it requires repeated atom-wise and coefficient updates over a large number of weight columns. We measure the relative activation reconstruction error $\|\mathbf{X}\mathbf{W} - \mathbf{X}\widetilde{\mathbf{W}}\|_F / \|\mathbf{X}\mathbf{W}\|_F$ using 256 calibration samples from WikiText-2. Our three-stage pipeline achieves lower reconstruction error than k-SVD while running 3.4× faster. We discuss the formulation of k-SVD in Appendix A1.1.

3.3 Dictionary-based compression pipeline: factorize, prune, refit, and quantize

Step 1: Activation-aware dictionary factorization. We first factorize each group of weights into a shared dictionary and layer-specific coefficients using activation-aware SVD. For a layer group $g = \{\ell_1, \ldots, \ell_n\}$, we first obtain the concatenated weight $\mathbf{W}^g = [\mathbf{W}^{\ell_1} | \mathbf{W}^{\ell_2} | \cdots | \mathbf{W}^{\ell_n}]$ for n weights in g, similar to [9]. Given calibration examples $\mathcal{X}$, we run the original model $f_{\mathbf{W}}$ to collect each linear module's input activations $\mathbf{X}^\ell$ and aggregate it as $\mathbf{S}^g = \sum_{\ell \in g} \mathbf{S}^\ell = \sum_{\ell \in g} (\mathbf{X}^\ell)^\top \mathbf{X}^\ell \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$. This is equivalent to computing the covariance after vertically concatenating the inputs from the grouped layers. We then perform rank-k SVD on the input-scaled concatenated weight $(\mathbf{S}^g)^{1/2}\mathbf{W}^g \approx \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top$, where $\mathbf{U}_k \in \mathbb{R}^{d_{\text{in}} \times k}$, $\mathbf{\Sigma}_k \in \mathbb{R}^{k \times k}$, and $\mathbf{V}_k \in \mathbb{R}^{n d_{\text{out}} \times k}$. The shared dictionary is attained as $\mathbf{D}^g = (\mathbf{S}^g)^{-1/2}\mathbf{U}_k \mathbf{\Sigma}_k$, and $\mathbf{V}_k^\top$ is equivalent to layer-specific coefficients $\{\mathbf{C}^\ell\}_{\ell \in g}$. Appendix A1.2 includes details on rank k in relation to the compression ratio.

Step 2: Dictionary-aware coefficient pruning. After dictionary factorization, we prune the layer-specific coefficients $\{\mathbf{C}^\ell\}_{\ell \in g}$, keeping the shared dictionary $\mathbf{D}^g$ fixed. Unlike SparseGPT [14], which prunes the dense weight $\mathbf{W}^\ell$ using the input covariance $\mathbf{S}^\ell = (\mathbf{X}^\ell)^\top \mathbf{X}^\ell \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$, DictLLM prunes in the coefficient space defined by the shared dictionary. Pruning the coefficient induces a structured weight perturbation $\Delta\mathbf{W}^\ell = \mathbf{D}^g \Delta\mathbf{C}^\ell$. The local reconstruction error from coefficient pruning is

$$\|\mathbf{X}^\ell \mathbf{D}^g \Delta\mathbf{C}^\ell\|_F^2. \tag{3}$$

This yields the dictionary-aware coefficient Hessian $\mathbf{H}^{\mathbf{C},g} = (\mathbf{D}^g)^\top \mathbf{S}^g \mathbf{D}^g \in \mathbb{R}^{k\times k}$, where $\mathbf{S}^g$ is the grouped input covariance from calibration examples. We derive this coefficient-space Hessian from the local reconstruction objective in Appendix A1.3.

Algorithm 2 summarizes the pruning procedure. $\mathrm{SaliencyMap}(\widetilde{\mathbf{C}}_{:,j:j+B}, \mathbf{d})$ returns $\mathbf{R}_{:,j:j+B} \in \mathbb{Z}^{k\times B}$ for each block of B columns. Each column of $\mathbf{R}$ sorts the k row indices by ascending saliency calculated by $\widetilde{\mathbf{C}}^2_{i,j}/\mathbf{d}_i$ where $\mathbf{d} \in \mathbb{R}^k$. At pruning step i, we find the least salient row index per column for the B columns in the block as $\mathbf{c} \in \mathbb{Z}^B$, and compute the error used to compensate the remaining rows as $\mathbf{e} \in \mathbb{R}^B$. After $\lfloor p \cdot k \rfloor$ steps, the binary mask $\mathbf{M}$ prunes the rows indexed by $\mathbf{R}_{0:\lfloor p\cdot k\rfloor,:}$. The key efficiency gain comes from operating in the coefficient space. Since $k < d_{\text{in}}$, inverting $\mathbf{H}^{\mathbf{C},g} \in \mathbb{R}^{k\times k}$ costs $\mathcal{O}(k^3)$ rather than $\mathcal{O}(d_{\text{in}}^3)$ of [14, 32]; the inverse is shared across all coefficient columns, reducing each column update to $\mathcal{O}(k^2)$.

Step 3: Updating shared dictionary with sparse coefficients. We refit the dictionary to the sparse coefficients while keeping the coefficient values and sparsity pattern fixed. This is necessary because the shared dictionary $\mathbf{D}^g$ is obtained before pruning and is therefore optimized for dense coefficients, not for the sparse support induced by $\widetilde{\mathbf{C}}^\ell$. We update the shared dictionary $\mathbf{D}^g \leftarrow \mathbf{W}^g(\widetilde{\mathbf{C}}^g)^\top(\widetilde{\mathbf{C}}^g(\widetilde{\mathbf{C}}^g)^\top + \lambda\mathbf{I})^{-1}$ where $\widetilde{\mathbf{C}}^g = [\widetilde{\mathbf{C}}^{\ell_1}|\cdots|\widetilde{\mathbf{C}}^{\ell_n}]$ for a layer group g. This is a solution to a least-squares problem, $\min_{\mathbf{D}^g} ||\mathbf{X}^g(\mathbf{W}^g - \mathbf{D}^g\widetilde{\mathbf{C}}^g)||_F^2$ where $\mathbf{X}^g = [(\mathbf{X}^{\ell_1})^\top, \ldots, (\mathbf{X}^{\ell_n})^\top]^\top$ denotes the vertical concatenation of calibration activations. Hence, $(\mathbf{X}^g)^\top\mathbf{X}^g = \sum_{\ell\in g}(\mathbf{X}^\ell)^\top\mathbf{X}^\ell = \mathbf{S}^g$. This update improves reconstruction under fixed sparse support without changing coefficient sparsity.

Step 4: Hessian-aware NVFP4 quantization. We quantize the dictionary and coefficients using respective Hessians. For the dictionary $\mathbf{D}^g$, we group input covariance as the Hessian since the module input is the original activation. For the coefficient $\mathbf{C}^\ell$, the Hessian proxy is $\mathbf{H}^{\mathbf{C},g} = (\mathbf{D}^g)^\top\mathbf{S}^g\mathbf{D}^g$, since the module input is $\mathbf{X}^\ell\mathbf{D}^g$. Thus, coefficient quantization uses the same dictionary-induced Hessian as coefficient pruning.

With these Hessians, we apply error compensation and quantize the dictionary and coefficients using NVFP4. During coefficient quantization, we preserve the pruning mask by fixing masked entries to zero and compensating for the resulting errors. We denote the NVFP4 quantizer as $\mathcal{Q}_{\mathrm{NVFP4}}(\cdot)$ and obtain the final compressed dictionary as $\widehat{\mathbf{D}}^g = \mathcal{Q}_{\mathrm{NVFP4}}(\mathbf{D}^g)$ and coefficient $\widehat{\mathbf{C}}^\ell = \mathcal{Q}_{\mathrm{NVFP4}}(\widetilde{\mathbf{C}}^\ell)$. Following the NVFP4 format [16], each block of values is represented by low-precision FP4 together with a shared microscale.

4 Experiments

4.1 Experimental setup

Models and datasets. We use DictLLM on LLaMA-7B and LLaMA2-7B [41], OPT-6.7B [42], and Mistral-7B [43] using the HuggingFace checkpoint. We evaluate language modeling performance on WikiText-2 [44], Penn TreeBank (PTB) [45], and Colossal Clean Crawled Corpus (C4) [46]. We also evaluate zero-shot reasoning performance on seven benchmarks: OpenBookQA [47], WinoGrande [48], HellaSwag [49], PIQA [50], MathQA [51], ARC-Easy, and ARC-Challenge [52]. All reasoning evaluations are performed in the zero-shot setting using the LM-Evaluation-Harness [53].

Baselines and metrics. We compare DictLLM against post-training low-rank compression methods. ASVD [11] applies activation-aware scaling before low-rank decomposition. SVD-LLM [8] improves low-rank compression with truncation-aware singular value decomposition. Basis Sharing [9] shares low-rank bases across transformer layers and is the closest baseline to our dictionary factorization. We report perplexity on language modeling tasks and accuracy for reasoning tasks. We measure compression gain based on the number of non-zero entries in the dictionary and coefficient matrices.

Implementation details. We use models with checkpoints from HuggingFace Transformers. We collect calibration statistics from 256 WikiText-2 samples with a sequence length of 2048. All experiments are run on dual NVIDIA RTX PRO 6000 Blackwell GPUs. We apply NVFP4 quantization post hoc to both dictionary and coefficient factors based on QuTLASS [54].

Table 1: **Benchmark on language modeling and reasoning tasks.** We benchmark DictLLM against low-rank compression baselines on language modeling and commonsense reasoning tasks for LLaMA-7B at different compression ratios. We additionally report the FP4-quantized $\text{DictLLM}_{\text{FP4}}$ models at the bottom of the table. We measure perplexity on WikiText-2, Penn TreeBank (PTB), and C4, and zero-shot accuracy on OpenBookQA (OB), WinoGrande (WG), HellaSwag (HS), PIQA (PQ), MathQA (MQ), ARC-Easy (AE), and ARC-Challenge (AC).

Comp. gain	Method	Perplexity ↓			Zero-shot accuracy ↑							
		WikiText-2	PTB	C4	OB	WG	HS	PQ	MQ	AE	AC	Average
1×	Original	5.68	8.35	7.34	0.28	0.67	0.56	0.78	0.27	0.67	0.38	0.52
1.25×	ASVD [11]	11.14	16.55	15.93	0.25	0.64	0.41	0.68	0.24	0.53	0.27	0.43
	SVD-LLM [8]	7.94	18.05	15.93	0.22	0.63	0.43	0.69	0.24	0.58	0.29	0.44
	Basis Sharing [9]	7.74	17.35	15.03	0.28	0.66	0.46	0.71	0.25	0.66	**0.36**	0.48
	DictLLM (Ours)	**6.89**	**12.27**	**11.75**	**0.33**	**0.67**	**0.50**	**0.74**	**0.26**	**0.69**	**0.36**	**0.51**
1.43×	ASVD [11]	51	70	41	0.18	0.53	0.37	0.65	0.21	0.43	0.25	0.38
	SVD-LLM [8]	9.56	29.44	25.11	0.20	0.59	0.40	0.65	0.22	0.48	0.26	0.40
	Basis Sharing [9]	9.25	29.12	22.46	0.27	0.63	0.40	0.68	0.24	0.63	0.30	0.45
	DictLLM (Ours)	**7.79**	**17.62**	**15.20**	**0.29**	**0.66**	**0.46**	**0.71**	**0.24**	**0.66**	**0.35**	**0.48**
1.67×	ASVD [11]	1407	3292	1109	0.13	0.48	0.26	0.55	0.19	0.28	0.22	0.30
	SVD-LLM [8]	13.11	63.75	49.83	0.19	0.58	0.33	0.60	0.21	0.42	0.25	0.37
	Basis Sharing [9]	12.39	55.78	41.28	0.22	0.61	0.35	0.62	0.23	0.52	0.27	0.40
	DictLLM (Ours)	**9.37**	**29.75**	**23.08**	**0.27**	**0.64**	**0.40**	**0.68**	**0.23**	**0.63**	**0.30**	**0.45**
2×	ASVD [11]	15358	47690	27925	0.12	0.51	0.26	0.52	0.19	0.26	0.22	0.30
	SVD-LLM [8]	23.97	150.58	118.57	0.16	0.54	0.29	0.56	0.21	0.33	0.23	0.33
	Basis Sharing [9]	19.99	126.35	88.44	0.18	0.57	0.31	0.58	0.22	0.42	0.23	0.36
	DictLLM (Ours)	**10.59**	**35.67**	**29.12**	**0.24**	**0.63**	**0.38**	**0.65**	**0.23**	**0.61**	**0.29**	**0.43**
5×	$\text{DictLLM}_{\text{FP4}}$ (Ours)	8.48	21.40	17.04	0.29	0.65	0.44	0.70	0.24	0.64	0.33	0.47
6×		10.37	35.16	27.07	0.26	0.61	0.38	0.66	0.23	0.59	0.28	0.43
7×		11.92	43.14	33.90	0.24	0.59	0.35	0.63	0.22	0.54	0.26	0.40

4.2 Experimental results

Benchmark on generation and reasoning datasets. We report language modeling perplexity and zero-shot reasoning accuracy in Table 1. DictLLM consistently improves over low-rank baselines [8, 9] across compression ratios from 1.25× to 2×. For example, DictLLM achieves an average zero-shot accuracy of 0.43 at 2× compression, compared with 0.40 for Basis Sharing at 1.67× compression. We also report FP4-quantized variants, $\text{DictLLM}_{\text{FP4}}$, at the bottom of Table 1. Notably, $\text{DictLLM}_{\text{FP4}}$ achieves an average accuracy of 0.40 at 7× compression, outperforming Basis Sharing at 2× compression, which achieves 0.36 average accuracy. We provide the sparsity level to reach the desired compression ratio in Table A1.

Perplexity-model size trade-off. We evaluate the trade-off between WikiText-2 perplexity and model size ratio in Fig. 4. DictLLM achieves a 5.1× compression ratio on the original LLaMA model with only a 2.9-point increase in WikiText-2 perplexity, providing 3.6× higher compression than the state-of-the-art low-rank method. In addition, DictLLM reduces perplexity by 9.4 points compared to Basis Sharing [9] with 2× compression on WikiText-2. DictLLM provides a continuous compression curve by controlling the parameter count via two knobs: dictionary size and coefficient sparsity. As the target model size decreases, perplexity gradually increases, reflecting stronger compression applied to both shared dictionaries and layer-specific

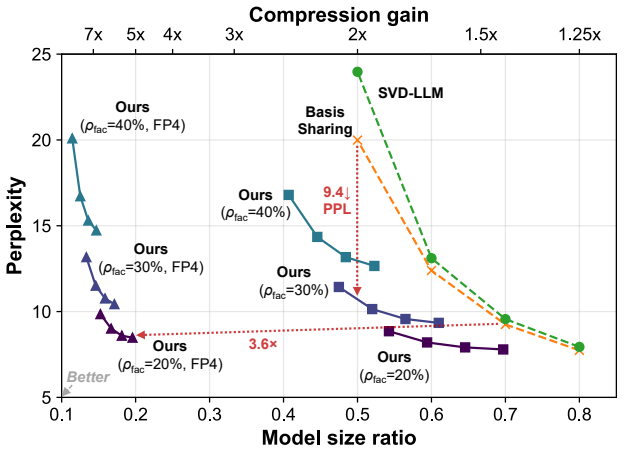

Figure 4: **Perplexity-model size trade-off.** WikiText-2 perplexity of compressed LLaMA-7B across factorization ratios and coefficient sparsities. DictLLM factorizes weights at rate ρ_{fac} and then prunes layer-specific coefficients with 20%–50% sparsity. We also report their NVFP4 versions.

Table 2: **Ablation of pruning target.** We ablate the pruning target with sparsity p: the dictionary $\mathbf{D}$, its coefficients $\mathbf{C}$, or both. Perplexity is measured on WikiText-2 using LLaMA-7B, yielding an overall compression gain of 1.43×.

Dict. $\mathbf{D}$ (p)	Coef. $\mathbf{C}$ (p)	Perplexity ↓
Dense (0%)	Dense (0%)	9.25
Dense (0%)	Sparse (19.4%)	**7.83**
Sparse (35.0%)	Dense (0%)	9.04
Sparse (12.5%)	Sparse (12.5%)	7.99

Table 3: **Choice of pruning method.** We prune the coefficients of LLaMA-7B based on magnitude, full Hessian inversion, and Hessian with lazy updates. We measure the perplexity on WikiText-2 with 50% sparsity on a 20% factorized model.

Method	Perplexity ↓	Time ↓
Magnitude	47.43	**44.6 min**
Full inversion	**9.89**	13.49 h
Lazy update (Ours)	**9.89**	55.6 min

coefficients. We first determine the rank k from the factorization compression rate ρ_{fac}, then apply 20%–50% sparsity to the per-layer coefficients. The final compression gain is therefore determined jointly by ρ_{fac} and the coefficient sparsity p.

Ablation of pruning target. Table 2 examines whether sparsity should be applied to the shared dictionary, the layer-specific coefficients, or both. We control the sparsity of the pruning target to reach a 30% compression ratio. The most interesting finding is that pruning the coefficients gives the lowest perplexity of all combinations. This suggests that the dictionary stores shared information reused across layers, whereas the coefficients exhibit more layer-specific redundancy. Therefore, DictLLM keeps the dictionary dense and applies sparsity only to the coefficients.

Choice of pruning method. We compare three coefficient pruning strategies to isolate the effect of dictionary-aware Hessian compensation (Table 3). Magnitude pruning is fast but ignores activation sensitivity and the dictionary, leading to significant degradation in perplexity. Full Hessian inversion gives much better perplexity by compensating for pruning errors, but its cost is impractical for repeated compression. Our dictionary-aware lazy-update method preserves the same curvature information while avoiding the expensive full inverse update. As shown in Table 3, it matches the perplexity of the full inversion method while being 14.6× faster, with a duration similar to the magnitude-based method.

Ablation of dictionary update. We ablate the effect of updating the dictionary after coefficient pruning (Fig. 5). We keep the sparse coefficient matrices fixed and refit the shared dictionary across a coefficient sparsity range from 20% to 50%. This step improves the reconstruction obtained with the fixed sparse codes without changing the sparsity pattern or the model-size estimate. The dictionary update reduces perplexity compared to the pruned baseline. The interesting part is that perplexity reduction is higher at increased sparsity: 1.04 PPL at 50% sparsity, whereas 0.06 PPL at 20% sparsity. This suggests that coefficient pruning yields a suboptimal dictionary for the remaining active atoms, and that the dictionary refit alleviates some of the resulting increase in perplexity.

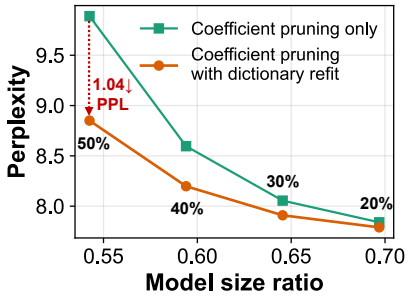

Figure 5: **Ablation of dictionary update.** WikiText-2 perplexity of LLaMA-7B across different sparsity levels, with and without dictionary update.

Different quantization format. We investigate how different precision choices for the dictionary and coefficients affect the final compression trade-off (Table 4). The FP16/FP16 baseline corresponds to the factorized model with a 30% reduction in factorizations and 50% coefficient sparsity before quantization. Quantizing only the dictionary to NVFP4 yields a clear storage gain with a small increase in perplexity, suggesting that the shared dictionary is relatively robust to low-bit quantization. When both the dictionary and coefficients are quantized to NVFP4, perplexity remains close to the INT8-coefficient setting while using almost half the precision. Although NVFP4/FP8 mixed quantization exhibits a decent perplexity-size trade-off, our all-NVFP4 setting yields the most compact representation and pushes the model-size frontier further to the left.

Table 4: **Different quantization format.** We compare quantization choices for the shared dictionary $\mathbf{D}$ and layer-specific coefficients $\mathbf{C}$. Perplexity is measured on WikiText-2 using LLaMA-7B, and compression gain denotes the reduction in model size. The FP16/FP16 baseline corresponds to the factorized model with a 30% factorization reduction and 50% coefficient sparsity before quantization.

Method	Dictionary $\mathbf{D}$	Coefficient $\mathbf{C}$	PPL ↓	Comp. gain ↑
Unquantized baseline	FP16	FP16	12.98	1.00×
Mixed precision 1	NVFP4	FP16	**13.66**	1.61×
Mixed precision 2	NVFP4	FP8	14.71	2.60×
Mixed precision 3	NVFP4	INT8	16.05	2.60×
All NVFP4 (Ours)	NVFP4	NVFP4	16.82	**3.56**×

Table 5: **Generalization across model families.** Perplexity (↓) of three different LLMs, e.g., LLaMA 2-7B, OPT-6.7B, and Mistral-7B, under 1.25× compression on WikiText-2.

Method	LLaMA2-7B	OPT-6.7B	Mistral-7B
ASVD [11]	10.10	82.0	13.72
SVD-LLM [8]	8.5	16.04	10.21
Basis Sharing [9]	7.70	11.79	7.57
DictLLM (Ours)	**6.93**	**11.36**	**6.46**

Table 6: **Effect of LoRA fine-tuning.** WikiText-2 perplexity (↓) of LLaMA-7B with post-compression LoRA fine-tuning across compression gains.

Comp. gain	Basis Sharing [9]	DictLLM
1.25×	7.09	**6.42**
1.43×	8.21	**7.54**
1.67×	10.44	**7.91**
2×	10.95	**8.71**

Generalization across model families. Table 5 reports WikiText-2 perplexity at 1.25× compression for OPT-6.7B, LLaMA 2-7B, and Mistral-7B. DictLLM consistently outperforms ASVD, SVD-LLM, and Basis Sharing across all three architectures. These results suggest that dictionary factorization and dictionary-aware Hessian pruning generalize across model families.

Effect of LoRA fine-tuning. Table 6 reports the effect of applying low-rank adaptation (LoRA) fine-tuning [55] after dictionary factorization and coefficient pruning. LoRA fine-tuning consistently recovers a portion of the perplexity lost during compression, reducing WikiText-2 perplexity by up to 1.9 points. The LoRA fine-tuning is more beneficial at higher compression ratios. We report LoRA fine-tuning for both Basis Sharing and DictLLM updating only query and value weights.

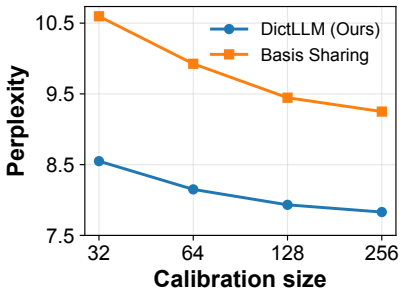

Figure 6: **Effect of calibration size.** WikiText-2 perplexity of LLaMA-7B as a function of calibration set size.

Effect of calibration set size. We vary the number of calibration samples from 32 to 256 and measure WikiText-2 perplexity of LLaMA-7B at 1.43× compression (Fig. 6). DictLLM consistently outperforms Basis Sharing across all calibration sizes, while both DictLLM and Basis Sharing improve as more calibration data is available. Notably, DictLLM, using only 32 samples, outperforms Basis Sharing with 256 samples.

5 Conclusion

We present DictLLM, a post-training compression method that represents transformer weights with shared dictionary kernels and layer-specific sparse coefficients. DictLLM first recovers shared dictionaries via activation-aware factorization, then prunes the coefficients using a dictionary-aware Hessian that projects the calibration covariance into the coefficient space. We then refit the shared dictionary by using the resulting coefficients. We further apply NVFP4 quantization with Hessian-based error compensation to both dictionary and coefficient factors, reducing storage while maintaining the factorized structure. Comprehensive experiments show that DictLLM improves the perplexity-model size trade-off and preserves performance relative to state-of-the-art low-rank compression methods across language modeling and zero-shot reasoning benchmarks.

Acknowledgments and disclosure of funding

This work was supported in part by COGNISENSE, one of seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. The work of SY Chun was supported by IITP grants funded by the Korea government(MSIT) [No.RS-2021-II211343, Artificial Intelligence Graduate School Program (Seoul National University) / No.RS-2025-02314125, Effective Human-Machine Teaming With Multimodal Hazy Oracle Models].

References

[1] Mark Horowitz. 1.1 Computing's Energy Problem (and what we can do about it). In *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*, pages 10–14. IEEE, 2014.

[2] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits*, 52(1): 127–138, 2016.

[3] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[4] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[5] Jaewon Lee, Gain Kim, Jinho Park, and Hyeon-Min Bae. Link bit-error-rate requirement analysis for deep neural network accelerators. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[6] Carl Eckart and Gale Young. The Approximation of One Matrix by Another of Lower Rank. *Psychometrika*, 1(3):211–218, 1936.

[7] Changwoo Lee, Soo Min Kwon, Qing Qu, and Hun-Seok Kim. BLAST: Block-Level Adaptive Structured Matrices for Efficient Deep Neural Network Inference. *Advances in Neural Information Processing Systems*, 37:14996–15027, 2024.

[8] Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. SVD-LLM: Truncation-aware Singular Value Decomposition for Large Language Model Compression. In *The Thirteenth International Conference on Learning Representations*, 2025.

[9] Jingcun Wang, Yu Guang Chen, Chao Lin, Bing Li, and Grace Li Zhang. Basis Sharing: Cross-Layer Parameter Sharing for Large Language Model Compression. In *The Thirteenth International Conference on Learning Representations*, 2025.

[10] Qinsi Wang, Jinghan Ke, Masayoshi Tomizuka, Kurt Keutzer, and Chenfeng Xu. Dobi-SVD: Differentiable SVD for LLM Compression and Some New Perspectives. In *The Thirteenth International Conference on Learning Representations*, 2025.

[11] Zhihang Yuan, Yuzhang Shang, Yue Song, Dawei Yang, Qiang Wu, Yan Yan, and Guangyu Sun. ASVD: Activation-aware Singular Value Decomposition for Compressing Large Language Models. *arXiv preprint arXiv:2312.05821*, 2023.

[12] Yu-Chen Lu, Sheng-Feng Yu, Hui-Hsien Weng, Pei-Shuo Wang, Yu-Fang Hu, Liang Hung-Chun, Hung-Yueh Chiang, and Kai-Chiang Wu. SkipCat: Rank-Maximized Low-Rank Compression of Large Language Models via Shared Projection and Block Skipping. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 24124–24132, 2026.

[13] Song Han, Huizi Mao, and William J Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv preprint arXiv:1510.00149*, 2015.

[14] Elias Frantar and Dan Alistarh. SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR, 2023.

[15] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A Simple and Effective Pruning Approach for Large Language Models. *Proceedings of the International Conference on Learning Representations*, 2024.

[16] NVIDIA. NVIDIA Blackwell Architecture Technical Brief, 2024. URL `https://resources.nvidia.com/en-us-blackwell-architecture`.

[17] Vage Egiazarian, Roberto L Castro, Denis Kuznedelev, Andrei Panferov, Eldar Kurtic, Shubhra Pandit, Alexandre Marques, Mark Kurtz, Saleh Ashkboos, Torsten Hoefler, et al. Bridging the Gap Between Promise and Performance for Microscaling FP4 Quantization. *Proceedings of the International Conference on Learning Representations*, 2026.

[18] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. QuIP: 2-Bit Quantization of Large Language Models With Guarantees. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[19] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. *Proceedings of the International Conference on Learning Representations*, 2023.

[20] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[21] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-Free 4-Bit Inference in Rotated LLMs. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

[22] Jinho Park, Jaewon Lee, Gain Kim, and Hyeon-Min Bae. Bin-Specific Quantization in Spectral-Domain Convolutional Neural Network Accelerators. In *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 407–410. IEEE, 2022.

[23] Seunghyun Moon, Jinho Park, Chuan-Tung Lin, SeongHwan Cho, Gregory Chen, Phil Knag, Ram Krishnamurthy, and Mingoo Seok. PersASR: A 2.3-μJ/frame 96.8%-accurate TinyML Automatic Speech Recognition Processor with Data Augmentation-Driven Personalization in 16-nm FinFET. In *2026 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4. IEEE, 2026.

[24] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *International Conference on Learning Representations*, 2020.

[25] Qian Lou, Ting Hua, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. DictFormer: Tiny Transformer with Shared Dictionary. In *International Conference on Learning Representations*, 2022.

[26] Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed Recursive Transformers: Effective Parameter Sharing with Layer-wise LoRA. *Proceedings of the International Conference on Learning Representations*, 2025.

[27] Tamir David Hay and Lior Wolf. Dynamic Layer Tying for Parameter-Efficient Transformers. In *The Twelfth International Conference on Learning Representations*, 2024.

[28] Seunghyun Moon, Mao Li, Gregory K Chen, Phil C Knag, Ram Kumar Krishnamurthy, and Mingoo Seok. T-REX: A 68-to-567µs/Token 0.41-to-3.95 µJ/Token Transformer Accelerator with Reduced External Memory Access and Enhanced Hardware Utilization in 16nm FinFET. In *2025 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 68, pages 406–408. IEEE, 2025.

[29] Kjersti Engan, Sven Ole Aase, and J Hakon Husoy. Method of optimal directions for frame design. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, volume 5, pages 2443–2446. IEEE, 1999.

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

[31] Magauiya Zhussip, Dmitriy Shopkhoev, Ammar Ali, and Stamatios Lefkimmiatis. Share Your Attention: Transformer Weight Sharing via Matrix-based Dictionary Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 29260–29268, 2026.

[32] Yann LeCun, John Denker, and Sara Solla. Optimal Brain Damage. *Advances in Neural Information Processing Systems*, 2, 1989.

[33] Kwanhee Lee, Hyeondo Jang, Dongyeop Lee, Dan Alistarh, and Namhoon Lee. The Unseen Frontier: Pushing the Limits of LLM Sparsity with Surrogate-Free ADMM. In *The Fourteenth International Conference on Learning Representations*, 2026.

[34] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

[35] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep Equilibrium Models. *Advances in Neural Information Processing Systems*, 32, 2019.

[36] Yuesen Liao, Zhiwei Li, Binrui Wu, Zihao Cheng, Su Zhao, Shuai Chen, and Weizhong Zhang. Compress Large Language Models via Collaboration Between Learning and Matrix Approximation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[37] Changhai Zhou, Qian Qiao, Yuhua Zhou, Yuxin Wu, Shichao Weng, Weizhong Zhang, and Cheng Jin. Large Language Model Compression with Global Rank and Sparsity Optimization. In *The Fourteenth International Conference on Learning Representations*, 2026.

[38] Jialin Zhao, Yingtao Zhang, and Carlo Vittorio Cannistraci. Pivoting Factorization: A Compact Meta Low-Rank Representation of Sparsity for Efficient Inference in Large Language Models. In *Forty-second International Conference on Machine Learning*, 2025.

[39] Yi Li, Zhichun Guo, Miao Yin, and Bingzhe Li. LeSTD: LLM Compression via Learning-based Sparse Tensor Decomposition. In *The Fourteenth International Conference on Learning Representations*, 2026.

[40] Denis Makhov, Dmitriy Shopkhoev, Magauiya Zhussip, Ammar Ali, and Stamatios Lefkimmiatis. CoSpaDi: Compressing LLMs via Calibration-Guided Sparse Dictionary Learning. *arXiv preprint arXiv:2509.22075*, 2025.

[41] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*, 2023.

[42] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open Pre-trained Transformer Language Models. *arXiv preprint arXiv:2205.01068*, 2022.

[43] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

[44] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. In *International Conference on Learning Representations*, 2017.

[45] Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330, 1993.

[46] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[47] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2381–2391, 2018.

[48] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740, 2020.

[49] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800, 2019.

[50] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. PIQA: Reasoning about Physical Commonsense in Natural Language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7432–7439, 2020.

[51] Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367, 2019.

[52] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[53] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The Language Model Evaluation Harness, 07 2024. URL `https://zenodo.org/records/12608602`.

[54] Roberto L. Castro and Dan Alistarh. QuTLASS: CUTLASS-Powered Quantized BLAS for Deep Learning. `https://github.com/IST-DASLab/qutlass`, 2025.

[55] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. LoRA: Low-Rank Adaptation of Large Language Models. *Proceedings of the International Conference on Learning Representations*, 2022.

[56] Xin Wang, Samiul Alam, Zhongwei Wan, Hui Shen, and Mi Zhang. SVD-LLM V2: Optimizing Singular Value Truncation for Large Language Model Compression. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4287–4296, 2025.

[57] Vladimír Macko and Vladimír Boža. MACKO: Sparse Matrix-Vector Multiplication for Low Sparsity. In *Forty-third International Conference on Machine Learning*, 2025.

[58] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025.

Appendix

Appendix contents

A1 Additional details 14
A1.1 Additional details on dictionary learning 14
A1.2 Compression and number of parameters 15
A1.3 Dictionary-induced coefficient Hessian. 15
A2 Additional experimental details 16
A3 Generalization across model families and scales 17
A4 Componentwise ablation 18
A5 Spectral analysis and motivation for cross-layer weight sharing 18
A6 Aggressive layer sharing with LoRA fine-tuning. 20
A7 Comparison with additional baselines 20
A8 End-to-end decoding throughput and memory 22
A9 Assets and compute resources 23
A9.1 Assets 23
A9.2 Computation resources. 23
A10 Discussions 23

A1 Additional details

A1.1 Additional details on dictionary learning

Relation to *k*-SVD. DictLLM is inspired by sparse dictionary learning, but its objective differs from classical k-SVD [30]. Given training signals $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]$, k-SVD learns an overcomplete dictionary $\mathbf{D} \in \mathbb{R}^{d \times k}$ with $k > d$ and sparse codes $\mathbf{A}$ by solving

$$\min_{\mathbf{D},\,\mathbf{A}} \|\mathbf{Y} - \mathbf{D}\mathbf{A}\|_F^2 \quad \text{s.t.} \quad \|\mathbf{a}_i\|_0 \le s,\ \forall i. \tag{A1}$$

Although k-SVD is often used with an overcomplete dictionary, our comparison uses an undercomplete dictionary ($k < d_{\text{in}}$) to match the compression setting. k-SVD is computationally expensive because it updates dictionary atoms and their associated sparse coefficients one at a time by repeatedly forming residuals and solving local rank-one approximation problems.

In contrast, DictLLM does not learn a dictionary from raw data signals. Instead, it starts from pretrained weights and minimizes the activation-space reconstruction error induced by replacing each dense weight with a shared dictionary and sparse coefficients:

$$\min_{\mathbf{D}^g,\,\{\mathbf{C}^\ell\}_{\ell \in g}} \sum_{\ell \in g} \left\|\mathbf{X}^\ell \mathbf{W}^\ell - \mathbf{X}^\ell \mathbf{D}^g \mathbf{C}^\ell\right\|_F^2 \quad \text{s.t.} \quad \|\mathbf{C}^\ell_{:,j}\|_0 \le (1-p)k. \tag{A2}$$

Here, $\mathbf{D}^g \in \mathbb{R}^{d_{\text{in}} \times k}$ is a compression dictionary whose size k is chosen from the target parameter budget, rather than an overcomplete dictionary. This makes the problem closer to post-training model compression than classical sparse coding. Accordingly, DictLLM uses a scalable staged approximation: activation-aware SVD initializes the dictionary, Hessian-aware pruning chooses the sparse coefficient support, and a MOD-style update refits the dictionary to the fixed sparse coefficients.

Relation to Method of Optimal Directions. The Method of Optimal Directions (MOD) [29] is a classical dictionary-learning algorithm that alternates between estimating sparse codes and updating the dictionary to better reconstruct training signals. Given fixed sparse codes, MOD updates all dictionary atoms at once by solving a least-squares problem. Our refitting step uses the same fixed-code dictionary update idea, but in a post-training model-compression setting. Instead of learning a

dictionary from raw signals, we start from pretrained transformer weights, keep the pruned coefficient matrices fixed, and refit the shared dictionary to reduce the reconstruction error of the original weights. Thus, our update is a lightweight post-pruning refinement rather than a full alternating dictionary-learning procedure.

A1.2 Compression and number of parameters

Modeling number of parameters in LLMs with dictionary and coefficients. Let $\mathcal{T}$ denote the set of weight types, such as attention and MLP projection matrices. Assuming that each weight $\mathbf{W}_t^\ell \in \mathbb{R}^{d_{\text{in},t} \times d_{\text{out},t}}$, the total number of parameters in the original model is

$$N_{\text{original}} = \sum_{\ell=1}^{L} \sum_{t \in \mathcal{T}} \left| \mathbf{W}_t^\ell \right| = \sum_{\ell=1}^{L} \sum_{t \in \mathcal{T}} d_{\text{in},t}^\ell d_{\text{out},t}^\ell, \tag{A3}$$

where $\left|\mathbf{W}_t^\ell\right|$ denotes the number of scalar parameters in weight matrix $\mathbf{W}_t^\ell$. This count includes only the transformer projection matrices compressed by DictLLM. It excludes the tokenizer, embedding tables, normalization parameters, biases, and the language modeling head, which are kept unchanged. DictLLM's grouped dictionary parameter count can naturally follow as

$$N_{\text{DictLLM}} = \sum_{t \in \mathcal{T}} \sum_{g \in \mathcal{G}_t} \left(d_{\text{in},t}^g k_t^g + (1 - p_t^g) k_t^g \sum_{\ell \in g} d_{\text{out},t}^\ell \right). \tag{A4}$$

Compression ratio and sparsity. We first describe how factorization determines the dictionary rank for one group of weights. Consider n matrices with the same shape $d_{\text{in}} \times d_{\text{out}}$ that share one dictionary of rank k. Before coefficient pruning, i.e., with sparsity $p = 0$, the factorization compression ratio is

$$\rho_{\text{fac}} = 1 - \frac{d_{\text{in}} k + n k d_{\text{out}}}{n d_{\text{in}} d_{\text{out}}}. \tag{A5}$$

Solving for k gives

$$k = \frac{n(1 - \rho_{\text{fac}}) d_{\text{in}} d_{\text{out}}}{d_{\text{in}} + n d_{\text{out}}}. \tag{A6}$$

For example, the query projection in LLaMA-7B has $d_{\text{in}} = d_{\text{out}} = 4096$. With $\rho_{\text{fac}} = 20\%$ and a sharing group of $n = 2$ layers, this gives $k = 2184$. In contrast, the down projection has shape 11008×4096 and is not shared across layers following Basis Sharing, so $n = 1$ and $k = 2388$. We choose the dictionary rank separately for each weight type so that the factorization compression ratio is matched across weight types.

After the rank is fixed, we apply sparsity level p to the coefficients, leaving only a $(1 - p)$ fraction of coefficient entries nonzero. The resulting final compression ratio is

$$\rho = 1 - \frac{d_{\text{in}} k + k d_{\text{out}} n (1 - p)}{d_{\text{in}} d_{\text{out}} n}. \tag{A7}$$

We apply the same coefficient sparsity across all weight types, so the resulting compression ratio can vary by weight type. For instance, when pruning a LLaMA-7B model factorized with a 20% reduction and layer sharing group size $n = 2$, we use coefficient sparsity $p = 0.194$ to reach 30% overall compression. Under this setting, the effective compression by weight type is 30.36% for Q, K, and V projections, 33.10% for up and gate projections, 27.78% for output projections, and 24.21% for down projections.

A1.3 Dictionary-induced coefficient Hessian.

Let $\widetilde{\mathbf{C}}^\ell = \mathbf{C}^\ell + \Delta\mathbf{C}^\ell$ be the pruned coefficient. Since the dictionary is fixed, the induced weight perturbation is $\Delta\mathbf{W}^\ell = \mathbf{D}^g \Delta\mathbf{C}^\ell$. Under the calibration reconstruction objective, the local pruning loss is

$$\left\| \mathbf{X}^\ell \Delta\mathbf{W}^\ell \right\|_F^2 = \left\| \mathbf{X}^\ell \mathbf{D}^g \Delta\mathbf{C}^\ell \right\|_F^2 = \operatorname{Tr}\left((\Delta\mathbf{C}^\ell)^\top (\mathbf{D}^g)^\top (\mathbf{X}^\ell)^\top \mathbf{X}^\ell \mathbf{D}^g \Delta\mathbf{C}^\ell \right). \tag{A8}$$

For a single layer, this derivation gives the coefficient-space Hessian

$$\mathbf{H}^{\mathbf{C},\ell} = (\mathbf{D}^g)^\top \mathbf{S}^\ell \mathbf{D}^g. \tag{A9}$$

In practice, we use a shared group-level Hessian proxy $\mathbf{H}^{\mathbf{C},g} = (\mathbf{D}^g)^\top \mathbf{S}^g \mathbf{D}^g$, where $\mathbf{S}^g = \sum_{\ell \in g} \mathbf{S}^\ell$. This pooled covariance approximation avoids computing and storing separate Hessian inverses for every layer while preserving activation-aware sensitivity at the group level.

A2 Additional experimental details

Our comparison excludes language modeling heads and embeddings, following the protocol of prior work; we count the number of non-zero entries in the coefficient matrices. Unless otherwise specified, we use damping $\lambda = 10^{-4}$ when regularizing covariance and Hessian matrices before inversion or Cholesky decomposition. We use a relative Hessian damping factor of 0.1 for masked quantization. For LM Harness [53] evaluation, we report `acc` as the accuracy of all reasoning tasks instead of `acc_norm`. For PTB [45] evaluation, we strip leading and trailing whitespace from each test sentence and join sentences with double newlines before tokenization, following our original evaluation protocol

Table A1: **Factorization and coefficient sparsity settings for LLaMA-7B.** ρ_{fac} denotes the parameter reduction from factorization, and sparsity denotes the fraction of pruned coefficient entries.

Compression gain	Factorization ratio ρ_{fac}	Sparsity p
1.25×	10%	17.25%
1.43×	20%	19.40%
1.67×	30%	22.17%
2×	30%	44.38%

Table A2: **Coefficient sparsity settings across LLaMA2-7B, OPT-6.7B, and Mistral-7B.** All models have a factorization ratio $\rho_{\text{fac}} = 10\%$ and corresponding coefficient sparsities to achieve 1.25× compression gain.

Models	LLaMA2-7B	OPT-6.7B	Mistral-7B
Sparsity p	17.25%	19.42%	17.66%

For the DictLLM results in Table 1, we report the reduction from factorization and coefficient sparsity in Table A1. All compressed models are calibrated on WikiText-2 and evaluated without task-specific fine-tuning. Quantized models DictLLM$_{\text{FP4}}$ reported at the bottom of Table 1 have $\rho_{\text{fac}} = \{20\%, 30\%, 30\%\}$ with sparsity $p = \{19.40\%, 22.17\%, 44.38\%\}$, achieving compression gains of 5.08×, 5.94×, and 7.12×.

In Table 2, the dense-dense setting reaches 1.43× compression using factorization alone. For the remaining settings, we start from a factorized model with $\rho_{\text{fac}} = 20\%$ and apply pruning to the dictionary, the coefficients, or both, using the sparsity levels p listed in the table.

In Table 5, we use models factorized with $\rho_{\text{fac}} = 10\%$ and sparsify the layer-specific coefficients with sparsity of 17.25% for LLaMA2-7B, 19.42% for OPT-6.7B, and 17.66% for Mistral-7B as described in Table A2. To reach 1.25× compression (or 20% compression rate), the effective compression rate of LLaMA2-7B by weight type is 20.37% for Q, K, and V projections, 23.09% for up and gate projections, 17.77% for output projections, and 14.23% for down projections. On the other hand, OPT-6.7B has per-weight type compression rates of 21.67% for Q, K, and V projections, 25.56% for the up projection, 18.75% for the output projections, and 13.50% for the down projections. Mistral-7B compressed by 1.25× has compression rates of 20.61% for Q, 13.21% for K and V, 17.96% for the output, 23.92% for the gate and up, and 13.54% for the down projections.

For the LoRA fine-tuning results in Table 6, we fine-tune DictLLM models at final compression gains from 1.25× to 2×, using the same factorization and pruning hyperparameters reported in Table 1. We use rank $r = 8$ adapters with a scaling factor $\alpha = 32$, yielding an effective LoRA update scale of $\alpha/r = 4$. We train only the LoRA adapter parameters while keeping the compressed backbone fixed. Fine-tuning is performed with a batch size of 1, a learning rate of 10^{-4}, and 2 training epochs. These experiments use the factorized and pruned models without dictionary refitting.

In Figure 5, we start from a model with a 20% factorization reduction, apply coefficient pruning with sparsity levels from 20% to 50%, and then apply dictionary refitting to each setting.

For the calibration set length sweep in Figure 6, we use the 1.43× compressed DictLLM model with factorization rate $\rho_{\text{fac}} = 20\%$, coefficient sparsity $p = 19.40\%$, and no dictionary refitting.

A3 Generalization across model families and scales

Table A3: **Benchmark on language modeling and reasoning tasks with Qwen3-8B.** We compare DictLLM against low-rank compression baselines at compression gains from 1.25× to 2×. We report perplexity on WikiText-2 and C4, and zero-shot accuracy on OpenBookQA (OB), WinoGrande (WG), HellaSwag (HS), PIQA (PQ), MathQA (MQ), ARC-Easy (AE), and ARC-Challenge (AC).

Comp. gain	Method	Perplexity ↓		Zero-shot accuracy (%) ↑							
		WikiText-2	C4	OB	WG	HS	PQ	MQ	AE	AC	Average
1×	Original	7.09	11.87	32.20	72.53	59.03	79.16	55.41	81.61	53.24	61.88
1.25×	SVD-LLM [8]	11.52	32.16	29.00	66.22	44.16	71.55	31.56	67.51	38.31	49.76
	Basis Sharing [9]	11.14	29.37	**31.40**	67.56	45.77	71.93	32.26	67.21	38.99	50.73
	DictLLM (Ours)	**9.90**	**22.59**	31.00	**70.09**	**50.07**	**74.86**	**35.78**	**70.54**	**41.89**	**53.46**
1.43×	SVD-LLM [8]	14.30	51.96	26.20	64.33	37.91	65.02	26.77	55.51	28.50	43.46
	Basis Sharing [9]	13.29	44.72	26.60	62.98	39.76	67.74	28.68	58.08	33.11	45.28
	DictLLM (Ours)	**11.26**	**30.09**	**31.60**	**67.64**	**45.21**	**72.36**	**32.56**	**65.66**	**38.82**	**50.55**
1.67×	SVD-LLM [8]	20.81	103.87	20.00	56.27	31.77	60.17	23.15	39.02	20.56	35.85
	Basis Sharing [9]	17.56	84.85	21.60	58.25	33.72	61.32	24.02	47.81	24.83	38.79
	DictLLM (Ours)	**13.59**	**47.79**	**26.40**	**63.22**	**39.05**	**67.03**	**27.71**	**58.08**	**32.94**	**44.92**
2×	SVD-LLM [8]	35.07	215.72	14.40	55.17	28.36	56.47	22.48	30.81	20.56	32.61
	Basis Sharing [9]	29.61	191.90	15.60	55.72	29.25	58.43	22.81	37.16	21.16	34.31
	DictLLM (Ours)	**16.58**	**73.52**	**24.20**	**60.14**	**35.38**	**63.06**	**26.37**	**54.04**	**28.24**	**41.63**

Table A4: **Qwen3-1.7B language modeling and reasoning results.** Methods are compared at the reported compression gains.

Comp. gain	Method	Perplexity ↓		Zero-shot accuracy (%) ↑							
		WikiText-2	C4	OB	WG	HS	PQ	MQ	AE	AC	Average
1×	Original	9.51	15.34	29.80	63.54	49.29	75.73	45.19	73.74	41.04	54.05
1.25×	SVD-LLM [8]	23.20	96.37	22.80	56.83	34.98	61.64	24.15	44.15	24.66	38.46
	Basis Sharing [9]	22.93	95.07	22.60	56.20	36.66	62.02	26.16	48.82	27.13	39.94
	DictLLM (Ours)	**17.66**	**52.80**	**25.80**	**59.91**	**40.63**	**67.85**	**27.84**	**53.37**	**28.75**	**43.45**
1.43×	SVD-LLM [8]	34.60	190.49	19.00	55.01	31.08	58.32	23.75	38.13	22.95	35.46
	Basis Sharing [9]	33.37	206.33	21.00	55.09	32.44	58.60	23.79	42.63	23.38	36.70
	DictLLM (Ours)	**23.45**	**99.27**	**21.80**	**56.27**	**36.39**	**62.30**	**25.83**	**47.52**	**25.51**	**39.37**

Table A5: **Qwen3-14B language modeling and reasoning results.** The improvements over other baselines extend to the largest model evaluated in language modeling and reasoning tasks.

Comp. gain	Method	Perplexity ↓		Zero-shot accuracy (%) ↑							
		WikiText-2	C4	OB	WG	HS	PQ	MQ	AE	AC	Average
1×	Original	6.46	10.95	35.40	74.19	61.75	80.41	61.78	83.33	55.63	64.64
1.25×	SVD-LLM [8]	9.49	27.15	28.80	68.98	47.19	72.69	34.34	67.00	38.48	51.07
	Basis Sharing [9]	9.18	25.93	30.20	70.24	49.32	74.32	36.98	66.08	39.33	52.35
	DictLLM (Ours)	**8.29**	**19.69**	**32.20**	**70.32**	**54.48**	**76.22**	**39.90**	**73.02**	**43.60**	**55.68**
1.43×	SVD-LLM [8]	11.51	44.54	26.00	65.11	39.29	67.08	29.95	62.25	31.57	45.89
	Basis Sharing [9]	10.81	40.77	28.80	68.67	42.67	70.24	32.83	60.35	35.07	48.38
	DictLLM (Ours)	**9.28**	**26.46**	**30.00**	**70.32**	**48.87**	**73.99**	**36.05**	**66.08**	**38.91**	**52.03**

Tables A3, A4, and A5 compare DictLLM with SVD-LLM [8] and Basis Sharing [9]. We evaluate 1.25× and 1.43× compression for all three scales, and extend the Qwen3-8B sweep to 1.67× and 2×. At every evaluated compression point, DictLLM achieves the lowest WikiText-2 and C4 perplexities and the highest average reasoning accuracy among these compression methods. The results of SVD-LLM [8] and Basis Sharing [9] are reproduced by us.

At 1.43× compression, DictLLM improves average accuracy over Basis Sharing from 36.70% to 39.37% on Qwen3-1.7B, from 45.28% to 50.55% on Qwen3-8B, and from 48.38% to 52.03% on Qwen3-14B. On Qwen3-8B at 2× compression, it reduces WikiText-2 perplexity from 29.61 to 16.58 and increases average accuracy from 34.31% to 41.63% relative to Basis Sharing. These results extend the performance-model size trade-off to a newer model family and multiple parameter scales.

Table A6: **Factorization and coefficient sparsity settings for Qwen3 model family.** We report the factorization ratio ρ_{fac} and Sparsity p required to achieve a given compression gain for Qwen3-8B, 1.7B, and 14B models.

Compression gain	**Factorization ratio** ρ_{fac}	**Sparsity** p		
		Qwen3-8B	Qwen3-1.7B	Qwen3-14B
1.25×	10%	17.25%	17.57%	17.52%
1.43×	20%	19.40%	19.79%	19.71%
1.67×	30%	22.17%	-	-
2×	30%	44.38%	-	-

We show the factorization ratio and coefficient sparsity to achieve a given compression gain for Qwen3 model families in Table A6

A4 Componentwise ablation

Table A7: **Componentwise ablation of compression stages.** 'F', 'P', 'R', and 'Q' respectively denote factorization, Hessian pruning, dictionary refitting, and quantization. A checkmark indicates an active component. We measure language and reasoning performance on LLaMA-7B.

Stage	**Comp. gain**	**Components**				**Perplexity** ↓		**Zero-shot accuracy** ↑							
		F	P	R	Q	Wiki-2	C4	OB	WG	HS	PQ	MQ	AE	AC	Avg.
Original	1×	–	–	–	–	5.68	7.34	0.28	0.67	0.56	0.78	0.27	0.67	0.38	0.52
+ Factorization	1.43×	✓	–	–	–	9.25	22.46	0.27	0.63	0.40	0.68	0.24	0.63	0.30	0.45
+ Hessian pruning	2×	✓	✓	–	–	11.53	33.82	0.25	0.62	0.36	0.64	0.23	0.57	0.27	0.42
+ Dictionary refit	2×	✓	✓	✓	–	10.59	29.12	0.24	0.63	0.38	0.65	0.23	0.61	0.29	0.43
+ Quantization	7×	✓	✓	✓	✓	11.92	33.90	0.24	0.59	0.35	0.63	0.22	0.54	0.26	0.40

Table A7 evaluates the successive stages on LLaMA-7B under the same calibration and evaluation protocol. Starting from factorization at $\rho_{\text{fac}} = 30\%$, coefficient-space Hessian pruning at approximately 44% sparsity increases the reported compression gain from 1.43× to 2×, while average reasoning accuracy decreases from 0.45 to 0.42. Dictionary refitting then raises accuracy to 0.43 and reduces WikiText-2 perplexity from 11.53 to 10.59 without changing the dictionary dimensions or coefficient sparsity. C4 perplexity also improves, from 33.82 to 29.12.

Refitting, therefore, provides a quality recovery step without additional parameter storage, although it requires an additional offline update. Subsequent quantization increases the reported compression gain to 7×, with WikiText-2 perplexity of 11.92 and an average accuracy of 0.40. This cumulative comparison exposes the storage–quality trade-off at each stage; it is not a leave-one-out experiment at a fixed final budget.

A5 Spectral analysis and motivation for cross-layer weight sharing

Motivation for input-scaled cross-layer factorization. We analyze the singular-value spectra of LLaMA-7B projection weights before and after cross-layer concatenation (A1). For each setting, we compare the raw weight spectrum with the input-scaled spectrum computed using the WikiText-2 calibration covariance. The input-scaled spectra accumulate faster, suggesting that fewer components are needed to preserve activation-weighted energy. This supports calibration-aware factorization: while the raw weights may not be strongly low-rank, their action on calibration inputs is concentrated in a smaller set of spectral directions.

While cross-layer SVD has a smaller slope than the single-layer counterpart, it enables the shared dictionary to capture information common across adjacent layers, thereby easing the rank requirement

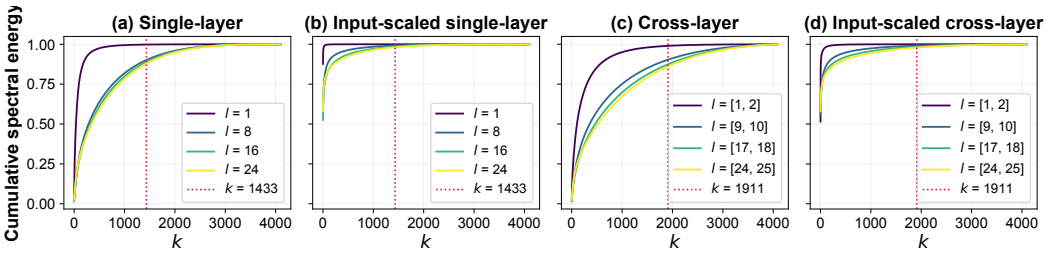

Figure A1: **Spectral concentration of LLaMA-7B query projection weights.** We plot cumulative spectral energy for single-layer SVD (a) with and (b) without input covariance scaling and cross-layer SVD (c) with and (d) without scaling across layers ℓ. Input-scaled spectra rise more rapidly, showing that the energy of input-scaled weights is more concentrated in low-rank components than that of the raw weight spectrum. Vertical dashed lines indicate the selected rank $k \in \{1433, 1911\}$ for the target factorization rate $\rho_{\text{fac}} = 30\%$ in single-layer and cross-layer settings.

$k = 1433 \rightarrow 1911$ for equal compression rate $\rho_{\text{fac}} = 30\%$. We also observe that the spectrum is more concentrated in the early layers: for example, layer $\ell = 1$ accumulates spectral energy rapidly at low ranks, whereas deeper layers, such as $\ell = 24$, exhibit slower spectral decay.

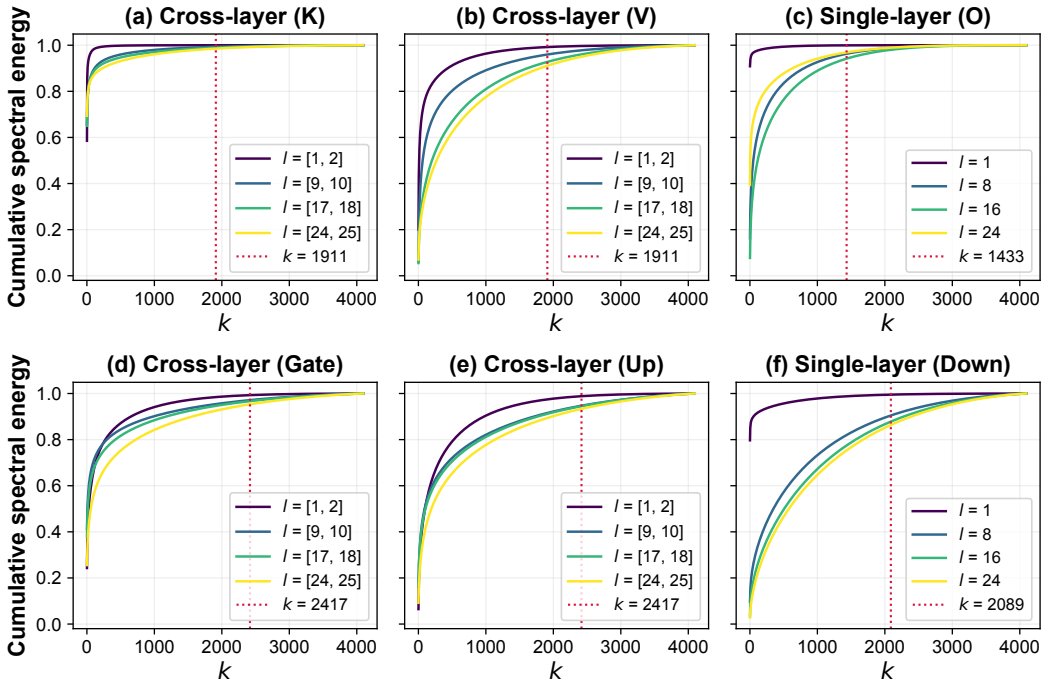

Figure A2: **Spectral concentration across different weight matrices in LLaMA-7B.** We plot cumulative spectral energy for the shared projections K, V, gate, and up using cross-layer SVD, and for the private projections O and down using single-layer SVD, following our factorization design. Vertical dashed lines indicate the selected rank k for the target factorization rate $\rho_{\text{fac}} = 30\%$ under the corresponding single-layer or cross-layer setting.

Spectral analysis of shared and private projections. We separately analyze the projection families according to our sharing design and report the findings in Fig. A2. For the shared projections K, V, gate, and up, we study the spectrum after cross-layer concatenation, since these matrices share a dictionary across layers. In contrast, O and down are private projections in our model, so we analyze them with single-layer SVD.

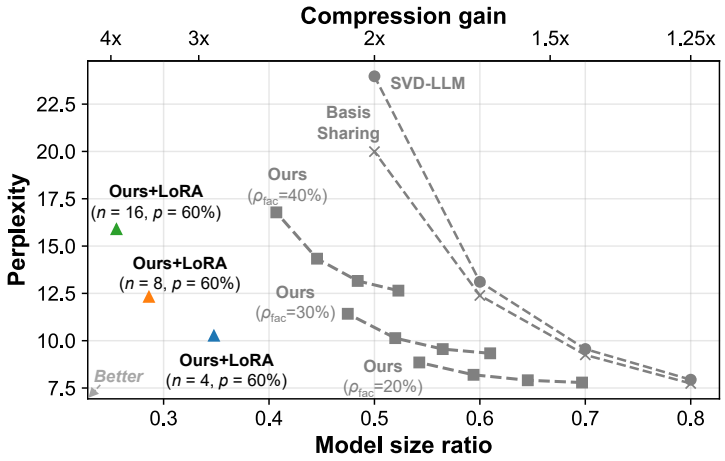

Figure A3: **Aggressive layer sharing with LoRA fine-tuning.** We increase the dictionary-sharing group size to $n = 4, 8, 16$ and apply 60% coefficient sparsity while keeping the rank of the shared dictionary fixed, then use LoRA fine-tuning to restore performance.

A6 Aggressive layer sharing with LoRA fine-tuning.

Fine-tuning allows more agressive layer-sharing. Dictionary sharing reduces parameter count by amortizing a single dictionary across a group of layers, but increasing the sharing group beyond two layers can degrade performance [9, Table 6]. To study a more aggressive compression setting, we also share the output and down projection matrices and increase the sharing group size to $n = 4, 8, 16$. We fix the shared dictionary ranks to $k_{\text{q,k,v,o}} = 2184$, $k_{\text{up,gate}} = 2762$, and $k_{\text{down}} = 3757$, corresponding to the ranks used at $\rho_{\text{fac}} = 20\%$. We then prune the coefficients with sparsity $p = 60\%$. To recover the resulting perplexity degradation, we apply LoRA fine-tuning to all projection matrices using the same hyperparameters as in Table 6. The three settings achieve model-size reductions of 2.9×, 3.5×, and 3.9×, with WikiText-2 perplexities of 10.26, 12.32, and 15.89, respectively.

Table A8: **Calibration size for NVFP4 quantization.** Post-quantization WikiText-2 perplexity of LLaMA-7B. Using 16 calibration samples provides a strong trade-off between perplexity and runtime.

Calibration size	Perplexity ↓
16	9.06
32	8.98
64	8.90
128	**8.88**

Ablation of calibration size on quantization. We apply NVFP4 quantization using subsets of the WikiText-2 training set for calibration. Table A8 shows that 16 calibration sequences provide an efficient accuracy-runtime trade-off. Using 16 sequences achieves a perplexity of 9.06 in slightly over an hour, while increasing the calibration size to 128 takes 8× longer and improves perplexity by only 0.18 points.

A7 Comparison with additional baselines

Pruning-only baselines at high sparsity. Table A9 compares SparseGPT [14] and Wanda [15] at 80.32% unstructured sparsity with 4-bit DictLLM at a similar reported 5× gain. DictLLM combines factorization at $\rho_{\text{fac}} = 20\%$ and approximately 19% coefficient sparsity with quantization. At this aggressive pruning level, SparseGPT and Wanda achieve average accuracies of 29.25% and 29.52%, respectively, whereas DictLLM achieves 46.92%. The language modeling results also favor DictLLM.

Table A9: **Comparison with pruning-only methods.** SparseGPT and Wanda are evaluated on LLaMA-7B at 80.32% unstructured sparsity. All results are reproduced by us.

Comp. gain	Method	Perplexity ↓		Zero-shot accuracy (%) ↑							
		WikiText-2	C4	OB	WG	HS	PQ	MQ	AE	AC	Average
1×	Original	5.68	7.34	30.80	69.38	56.36	78.24	26.57	74.66	41.55	53.94
5×	SparseGPT [14]	86.72	257.96	14.00	48.46	25.38	51.41	18.79	25.63	21.08	29.25
	Wanda [15]	2.8×10^3	2.8×10^3	11.60	48.30	25.78	53.10	21.21	26.35	20.31	29.52
	DictLLM$_{\text{FP4}}$	**8.48**	**17.04**	**29.20**	**64.64**	**43.67**	**70.18**	**23.99**	**64.02**	**32.76**	**46.92**

This comparison illustrates the benefit of distributing compression across factorization, sparsity, and precision, rather than relying solely on pruning to meet the budget.

Table A10: **Comparison with quantization-only baselines.** We compare methods at 7× compression gain on LLaMA-7B. GPTQ [19] and AWQ [34] use two-bit weights, whereas DictLLM combines factorization and coefficient sparsity with four-bit quantization. All results are reproduced by us.

Comp. gain	Method	Perplexity ↓		Zero-shot accuracy (%) ↑							
		WikiText-2	C4	OB	WG	HS	PQ	MQ	AE	AC	Average
1×	Original	5.68	7.34	30.80	69.38	56.36	78.24	26.57	74.66	41.55	53.94
7×	GPTQ [19] (2-bit)	17.59	36.56	13.80	48.38	30.47	57.13	20.84	32.03	21.33	32.00
	AWQ [34] (2-bit)	2.6×10^5	2.9×10^5	13.60	49.33	25.40	52.83	21.57	25.38	22.35	30.07
	DictLLM$_{\text{FP4}}$	**11.92**	**33.90**	**23.60**	**59.19**	**35.32**	**63.17**	**22.21**	**53.62**	**25.60**	**40.39**

Quantization-only baselines at a larger compression gain. Table A10 examines the more aggressive reported 7× budget. NVFP4 supplies approximately a 3.56× reduction, including its scale overhead, which compounds with the approximately 2× gain from factorization and pruning at $\rho_{\text{fac}} = 30\%$ and approximately 44% coefficient sparsity. We therefore compare against 2-bit GPTQ [19] and AWQ [34], rather than 4-bit quantization alone, to approach this larger compression gain.

DictLLM achieves an average accuracy of 40.39%, compared with 32.00% for GPTQ [19] and 30.07% for AWQ [34]. It also has lower WikiText-2 perplexity than either quantization-only baseline.

Table A11: **Comparison with low rank methods including SVD-LLM V2.** We measure WikiText-2 perplexity at 20% compression.

Method	LLaMA-7B	OPT-6.7B
ASVD [11]	11.14	82.0
SVD-LLM [8]	7.94	16.04
SVD-LLM V2 [56]	7.12	13.46
Basis Sharing [9]	7.74	11.79
DictLLM (Ours)	**6.89**	**11.36**

Comparison against SVD-LLM V2. We supplement our comparisons with the results of SVD-LLM V2 [56]. Table A11 reports WikiText-2 perplexities at 20% compression on LLaMA-7B and OPT-6.7B. DictLLM obtains perplexities of 6.89 and 11.36, compared with 7.12 and 13.46 for SVD-LLM V2.

Basis Sharing followed by quantization. We apply 4-bit GPTQ [19] and AWQ [34] to a Basis Sharing model factorized at $\rho_{\text{fac}} = 30\%$. Table A12 compares these combinations with 4-bit DictLLM at the same reported 5× compression gain. DictLLM uses $\rho_{\text{fac}} = 20\%$ and approximately 19% coefficient sparsity before quantization. It achieves WikiText-2 perplexity of 8.48, compared with 13.99 for Basis Sharing + GPTQ and 11.38 for Basis Sharing + AWQ. Its average accuracy is 46.92%, compared with 37.80% and 41.22%, respectively. These results show that quantizing

Table A12: **Effect of compounding quantization on Basis Sharing.** 'BS' denotes Basis Sharing [9]. We measure language and reasoning performances on LLaMA-7B. All results are reproduced by us.

Comp. gain	Method	Perplexity ↓		Zero-shot accuracy (%) ↑							
		Wiki-2	C4	OB	WG	HS	PQ	MQ	AE	AC	Average
1×	Original	5.68	7.34	30.80	69.38	56.36	78.24	26.57	74.66	41.55	53.94
5×	BS [9] + GPTQ [19] (4-bit)	13.99	34.01	24.00	56.83	34.72	57.89	20.74	46.93	23.46	37.80
	BS [9] + AWQ [34] (4-bit)	11.38	30.95	24.40	59.91	36.09	64.09	21.94	55.35	26.79	41.22
	$\text{DictLLM}_{\text{FP4}}$	**8.48**	**17.04**	**29.20**	**64.64**	**43.67**	**70.18**	**23.99**	**64.02**	**32.76**	**46.92**

a factorized model alone does not match the quality of the DictLLM configuration at this reported budget.

Table A13: **Effect of compounding pruning and quantization on Basis Sharing.** 'BS' denotes Basis Sharing [9]. Magnitude pruning acts on the coefficient matrix. We measure language and reasoning performance on LLaMA-7B. All results are reproduced by us.

Comp. gain	Method	Perplexity ↓		Zero-shot accuracy (%) ↑							
		Wiki-2	C4	OB	WG	HS	PQ	MQ	AE	AC	Average
1×	Original	5.68	7.34	30.80	69.38	56.36	78.24	26.57	74.66	41.55	53.94
1.43×	BS [9] + magnitude	8.53	17.28	**29.60**	65.35	43.67	70.40	23.28	65.32	**35.07**	47.53
	DictLLM	**7.79**	**15.20**	29.00	**66.38**	**45.67**	**71.27**	**24.25**	**65.53**	**35.07**	**48.17**
5×	BS [9] + magnitude + AWQ [34] (4-bit)	10.39	23.50	23.80	62.98	38.64	66.05	22.28	57.32	25.51	42.37
	$\text{DictLLM}_{\text{FP4}}$	**8.48**	**17.04**	**29.20**	**64.64**	**43.67**	**70.18**	**23.99**	**64.02**	**32.76**	**46.92**

Basis Sharing followed by magnitude pruning. To compare coefficient pruning strategies, we apply 19.4% coefficient sparsity to a Basis Sharing model with $\rho_{\text{fac}} = 20\%$, and optionally apply 4-bit AWQ afterward. As shown in Table A13, DictLLM improves WikiText-2 perplexity from 8.53 to 7.79 and average accuracy from 47.53% to 48.17% before quantization. With quantization, the corresponding values improve from 10.39 to 8.48 and from 42.37% to 46.92%. The unquantized comparison supports the benefit of Hessian-based coefficient pruning together with dictionary refitting over magnitude-only coefficient pruning. The quantized comparison also differs in its quantization method; therefore, it measures the combined compression pipeline rather than isolating pruning alone.

A8 End-to-end decoding throughput and memory

Table A14: **End-to-end decoding on LLaMA-7B.** DictLLM uses MACKO for sparse coefficients. The measurement excludes the prompt prefill stage.

Configuration	Weight memory [GB]	Peak memory [GB]	Speed [tokens/s]
Original	13.48	13.66	102.35
Basis Sharing [9] (dense $\mathbf{D}$, dense $\mathbf{C}$)	9.59	9.83	113.59
DictLLM (dense $\mathbf{D}$, sparse $\mathbf{C}$)	**7.82**	**8.02**	**126.81**

Benchmark setup. We use MACKO [57] to perform sparse-coefficient operations and to measure whole-model decoding, rather than isolated matrix multiplication. All configurations use FP16 precision on an NVIDIA RTX PRO 6000 Blackwell Max-Q GPU with a batch size of 1. Each run uses a 128-token prompt and 100 single-token autoregressive decoding steps. We report results over 10 measured runs following three warm-up runs. Prompt prefill is excluded from the timed interval.

Memory saving and speed-up. Table A14 compares the dense LLaMA-7B model, Basis Sharing with $\rho_{\text{fac}} = 30\%$, and DictLLM with the same factorization rate and approximately 44% coefficient sparsity. DictLLM reaches 126.81 tokens/s, a 1.24× increase over the original model and a 1.12× increase over the factorization-only model. Weight memory decreases from 13.48 to 7.82 GB (1.72× reduction), and peak memory decreases from 13.66 to 8.02 GB (1.70× reduction). These results

demonstrate a realized decoding benefit for the tested hardware and sparse backend, beyond the representation-level size estimates.

A9 Assets and compute resources

A9.1 Assets

License for WikiText-2. The WikiText-2 [44] dataset is published under the CC-BY-SA-3.0 License, and all data[1] is released under the GNU Free Documentation License family.

License for PTB. The PTB [45] dataset[2] is released without an explicit license; however, the data is released under LDC User Agreement for Non-Members.

License for C4. The C4 [46] dataset[3] is released under the ODC-BY License.

License for LM Harness. The LM Harness framework [53] with code[4] is released under MIT License.

License for LLaMA-7B. The LLaMA-7B model [41] with a variant checkpoint[5] is licensed under License Non-commercial bespoke license.

License for LLaMA 2-7B. The LLaMA 2-7B model [41] with checkpoint[6] is licensed under Llama 2 Community License Agreement.

License for Mistral-7B. The Mistral-7B model [43] with checkpoint[7] is licensed under Apache License 2.0.

License for OPT-6.7B. The OPT-6.7B model [42] with checkpoint[8] is licensed under OPT-175B License Agreement.

License for Qwen3. The Qwen 3 models [58] with checkpoints for Qwen3-1.7B[9], -8B[10], and -14B[11] are licensed under the Apache License 2.0.

A9.2 Computation resources.

The experiments are conducted on a Red Hat Enterprise Linux 8 server equipped with dual NVIDIA RTX PRO 6000 Blackwell Max-Q GPUs, each equipped with 96GB VRAM, an AMD Ryzen Threadripper PRO 5975WX CPU @ 3.6-4.5GHz, and 256GB of RAM.

A10 Discussions

Limitations. A limitation of DictLLM is that coefficient pruning produces unstructured sparsity. While this reduces parameter storage and can be exploited with sparse kernels, its hardware efficiency depends on support for irregular sparse access patterns. Exploring structured coefficient sparsity, such as block sparsity, is a promising direction for improving hardware efficiency and co-design.

[1] https://huggingface.co/datasets/mindchain/wikitext2
[2] https://huggingface.co/datasets/ptb-text-only/ptb_text_only
[3] https://huggingface.co/datasets/allenai/c4
[4] https://github.com/EleutherAI/lm-evaluation-harness
[5] https://huggingface.co/jeffwan/llama-7b-hf
[6] https://huggingface.co/meta-llama/Llama-2-7b
[7] https://huggingface.co/mistralai/Mistral-7B-v0.1
[8] https://huggingface.co/facebook/opt-6.7b
[9] https://huggingface.co/Qwen/Qwen3-1.7B
[10] https://huggingface.co/Qwen/Qwen3-8B
[11] https://huggingface.co/Qwen/Qwen3-14B

Broader impact. DictLLM aims to reduce the memory footprint of large language models, making capable models more accessible on resource-constrained hardware and lowering the energy cost of inference. This may broaden access to LLM deployment beyond large datacenters and enable more private, local, and low-latency applications. At the same time, easier deployment of compressed LLMs can also lower the barrier to misuse, including spam generation, misinformation, or unauthorized automation. Since DictLLM preserves the behavior of the underlying pretrained model as closely as possible, it may also retain that model's biases, safety risks, and failure modes.

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: Our contributions are listed at the end of the introduction, and summarized in the abstract.

   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: We discuss the limitations of our work in the Appendix, focusing on the underlying assumptions on the calibration set and the hardware support for sparse computations.

   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: [No]

Justification: The primary contribution of this work does not lie in theoretical analysis.

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: We clearly explain our shared dictionary-based post-training compression method in the manuscript. We will release our model and code to foster the reproduction of results.

   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 will be made available on a dedicated GitHub repository to support reproducibility of our core experimental results using the public dataset. All the assets used are included in 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: Details on baselines, hyperparameters, and optimizer settings are presented in the main paper, with additional information provided in the Appendix.

   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: [No]

   Justification: We do not report error bars, consistent with prior post-training LLM compression work, and instead evaluate over large-scale language modeling and reasoning datasets as a more indicative measure of model performance. Repeated runs for variance estimation are impractical due to the high computational cost of these tasks.

   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: We include details of our computational platform–such as CPU, GPU, and RAM specifications.

   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: We have reviewed the NeurIPS Code of Ethics and ensured that our research complies with all stated guidelines.

   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: We discuss the broader impact of this work in the Appendix.

    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 release of our models and code does not pose any known risks.

    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: All datasets are properly cited, and their licenses are listed in the Appendix.

    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: New assets are well documented and released with supporting materials.

    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: This research does not involve the use of crowdsourcing or human subjects.

    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: This research does not require IRB approval as it does not involve human subjects.

    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: We used OpenAI's Codex as a writing and coding assistant during manuscript preparation and implementation inspection. In particular, Codex was used to help read and summarize parts of the codebase, check notation consistency, and draft or polish explanatory text. All technical ideas, experimental design choices, implementation decisions, numerical results, and scientific conclusions were produced, verified, and approved by the authors. No LLM-generated content was used without author review.

    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.