Residue Pruning: Closing the Gap Between 2:4 and Unstructured Sparse LLMs with Low-Rank Decomposition

Anonymous ACL submission

Abstract

Pruning is a fundamental technique for compressing large language models (LLMs) by removing redundant parameters while preserving task performance. However, existing pruning methods face a core dilemma between accuracy and deployability: unstructured pruning usually preserves model quality better, but is difficult to accelerate efficiently on modern GPUs, whereas 2:4 structured pruning is hardware-friendly and yields real memory and inference benefits, at the cost of substantially larger accuracy degradation. We further find that this quality gap becomes even more severe on newer and smaller LLMs, which are especially important for practical deployment. To address this challenge, we propose Residue Pruning, a sparse-plus-low-rank framework tailored for 2:4 structured pruning, where each dense weight matrix is decomposed into a 2:4 sparse component and a low-rank residue. At the algorithm level, we formulate compression as a layer-wise output reconstruction problem and develop an alternating prune-fit optimization that jointly updates the sparse and low-rank components. At the system level, we design a three-level optimization pipeline, including shared-input projection packing, epilogue fusion, and branch overlap, to reduce the inference overhead of the low-rank branch. Experiments across multiple LLM families and scales show that Residue Pruning consistently improves accuracy and perplexity over existing sparse-plus-low-rank 2:4 pruning methods, while achieving stronger inference efficiency than naive sparse-plus-low-rank deployment.

1 Introduction

Pruning (Ma et al., 2023; Frantar and Alistarh, 2023; Sun et al., 2024; Kim et al., 2024; An et al., 2023) is a fundamental technique for compressing neural networks by removing redundant parameters while preserving task performance. Existing pruning methods mainly fall into two categories: *unstructured pruning*, which removes individual weights in an element-wise manner, and *structured pruning*, which removes weights according to predefined hardware-friendly patterns or groups, such as channels, attention heads, or layers. Among different structured sparsity formats, *semi-structured* $N : M$ pruning has attracted particular attention because it offers a favorable trade-off between compression flexibility and hardware efficiency: within every group of M weights, only N are retained as nonzeros. In particular, *2:4 semi-structured sparsity*, which keeps two nonzero weights in every group of four, has become one of the most practically important settings due to its direct support from modern GPU software and hardware stacks.

Despite this progress, LLM pruning still faces a fundamental dilemma between *accuracy* and *deployability*. On the one hand, 2:4 structured pruning benefits from its regular sparsity pattern and corresponding hardware support, enabling actual reductions in memory usage and inference latency(Fang et al., 2024; Zhang and Papyan, 2025; Sun et al., 2024). On the other hand, unstructured pruning is much harder to accelerate in practice, often requiring extremely high sparsity together with specialized accelerator or kernel designs before parameter sparsity translates into real system-level gains (Wang et al., 2021; Wang, 2020). However, from the model-quality perspective, the situation is reversed: unstructured pruning usually achieves substantially better accuracy than 2:4 pruning at the same sparsity level because it imposes far fewer constraints on which weights can be preserved. More importantly, we further find that this quality gap is not only large, but becomes even more pronounced for newer and smaller LLMs, which are particularly important for practical deployment. When moving from Llama2-7B (Touvron et al., 2023) to Llama3.1-8B (Grattafiori et al., 2024), the perplexity gap between 2:4 and unstructured pruning increases by 2.64×. When moving from

Qwen3-8B (Yang et al., 2025) to Qwen3-1.7B, the gap further enlarges by 7.36×. This trend is particularly concerning because smaller models are often the primary candidates for deployment on resource-limited devices. These observations raise the central question of this work: *how can we retain the deployment benefits of 2:4 structured pruning while reducing its growing accuracy gap with unstructured pruning for LLMs?*

Recent *sparse-plus-low-rank* methods (Li et al., 2023; Zhang and Papyan, 2025; Saha et al., 2024) provide a natural direction for addressing this problem by decomposing a dense weight matrix into the combination of a sparse branch and a low-rank branch. Their main idea is intuitive: the sparse branch preserves the hardware-friendly main structure that delivers memory and throughput benefits, while the low-rank branch compensates for pruning-induced errors. Representative methods along this line include TLR (Shen et al., 2025), which refines a pruned weight matrix under a fixed sparse mask with an additional low-rank correction, and OATS (Zhang and Papyan, 2025), which further leverages outlier-aware guidance to improve low-rank recovery. However, these methods still suffer from two important limitations. First, they mainly formulate recovery in the *weight space* by reconstructing pruned weights, whereas pruning quality is ultimately determined by the functional distortion induced at the layer output. Second, they largely overlook the deployment challenge introduced by the low-rank branch: although it adds only a small number of parameters, its runtime overhead can be far from negligible in practice and may significantly weaken the efficiency gains of the sparse module.

To address these limitations, we propose **Residue Pruning**, a sparse-plus-low-rank framework tailored for 2:4 structured pruning. Specifically, for each dense weight matrix, we construct a compressed approximation by decomposing it into a 2:4 sparse component and a low-rank residue. **At the algorithm level**, unlike prior methods that mainly perform recovery in the weight space, our method formulates compression as a *layer-wise output reconstruction* problem, where pruning errors are measured by their effect on layer behavior under calibration activations. This formulation is better aligned with the functional objective of pruning, since the impact of compression is ultimately reflected in output distortion rather than weight mismatch alone. Based on this formulation, we develop an *alternating prune-fit optimization* that jointly updates the sparse and low-rank components. We first initialize the decomposition by extracting a low-rank branch from the dense weight matrix, so that the low-rank component captures the dominant global structure before structured pruning is applied. We then alternate between performing 2:4 pruning on the current residual target and refitting the low-rank residue to the remaining unrecovered error, enabling the sparse and low-rank components to play complementary roles rather than treating the residue as a purely post-hoc correction. **At the system level**, to mitigate this issue, we design a three-level system optimization pipeline, including *operator-level packing of shared-input projections*, *epilogue fusion of packing-induced overheads*, and *branch-level overlap across sparse and low-rank paths*, which together reduce the inference overhead of the low-rank branch and make the hybrid sparse-plus-low-rank design practically deployable. Overall, experiments across multiple LLM families and scales show that Residue Pruning consistently delivers better accuracy and perplexity than existing *sparse-plus-low-rank* pruning methods, while achieving stronger inference efficiency than naive *sparse-plus-low-rank* deployment.

2 Related Works

Pruning for LLMs. Pruning aims to remove less important parameters while preserving performance, reducing model size and inference cost. For large language models (LLMs), *post-training pruning* has become widely adopted, due to the prohibitively expensive cost of retraining. In general, existing methods estimate parameter importance from a small calibration set and then perform either one-shot (An et al., 2023; Frantar and Alistarh, 2023; Sun et al., 2024; Ma et al., 2023) or iterative pruning (Men et al., 2024; Wang et al., 2025; Sreenivas et al., 2024). Representative one-shot methods include SparseGPT (Frantar and Alistarh, 2023), Wanda (Sun et al., 2024), FLAP (An et al., 2023), and LLM-Pruner (Ma et al., 2023). SparseGPT formulates pruning as sparse regression with approximate Hessian inversion. Wanda simplifies SparseGPT's importance to weight-activation products, achieving similar accuracy with higher pruning efficiency. FLAP uses a fluctuation-based metric and bias compensation to recover performance. LLM-Pruner groups dependent parameters

and leverages gradients for importance estimation.

According to the sparsity granularity, existing LLM pruning methods can be broadly categorized as *unstructured pruning* and *structured pruning*. *Unstructured pruning* removes individual weights without imposing any specific sparsity pattern. This offers greater flexibility and often leads to higher accuracy at the same sparsity level compared to structured pruning. However, its irregular sparsity pattern rarely translates into practical speedup or memory savings on CPUs/GPUs unless sparsity is extremely high (e.g., above 95%) (Wang et al., 2021; Wang, 2020).

In contrast, structured pruning imposes deployment-friendly sparsity patterns at different levels. At a coarse level, channel-wise pruning removes attention heads or MLP channels (Ma et al., 2023; Wang, 2020), while layer-wise pruning discards less important transformer layers (Men et al., 2024; Sreenivas et al., 2024). At a finer level, *semi-structured N:M pruning* enforces a regular N:M sparsity pattern in which, for every group of M contiguous weights, only N is retained as nonzero. Among these structured sparsity patterns, N:M pruning has attracted particular attention because its regular pattern is better supported by modern GPU hardware. In particular, 2:4 sparsity has become a widely adopted configuration in practice, as it offers practical memory savings and inference speedup while typically preserving model accuracy better than more coarse-grained alternatives such as channel-wise or layer-wise pruning at the same sparsity level.

Low-rank decomposition and recovery. Low-rank decomposition is another widely used model compression technique that approximates a weight matrix as a product of two smaller matrices. In the context of LLMs, low-rank structure has been explored both as a standalone compression strategy (Saha et al., 2024) and, more relevant to our work, as a lightweight performance recovery mechanism to compensate for the compression error of pruned models (Li et al., 2023; Shen et al., 2025; Zhang and Papyan, 2025). In particular, LoSparse (Li et al., 2023) approximates each weight matrix as the sum of a structured sparse matrix and a low-rank matrix, and performs iterative prune-then-fine-tune procedures with the sparse and low-rank components jointly optimized to preserve task performance. TLR (Shen et al., 2025) refines a pruned weight matrix under a fixed sparse mask while learning an additional low-rank correction term to recover information lost during pruning. Its low-rank component is updated iteratively according to a rank schedule whose effective rank gradually decreases, resulting in a sparse matrix together with a compact low-rank residual. OATS (Zhang and Papyan, 2025) also decomposes model weights into sparse and low-rank components, and further leverages Hessian-diagonal information to better preserve outlier-sensitive directions without retraining. Existing methods mainly perform low-rank recovery in the weight space by reconstructing pruned weights. In contrast, our method performs optimization in the layer-wise output space, which more directly captures the functional impact of pruning errors on layer behavior and downstream model performance.

Table 1: Throughput–perplexity–memory trade-off of current unstructured (marked as unstr.) and structured pruning methods (marked as str.).

Method	Prefill (tok/s) ↑	PPL ↓	Mem (GB) ↓
Dense	5882.5	**12.1907**	12.86
Wanda 2:4 (str.)	**6891.7**	28.2094	**7.65**
Wanda 50% (unstr.)	5879.3	16.5170	12.87

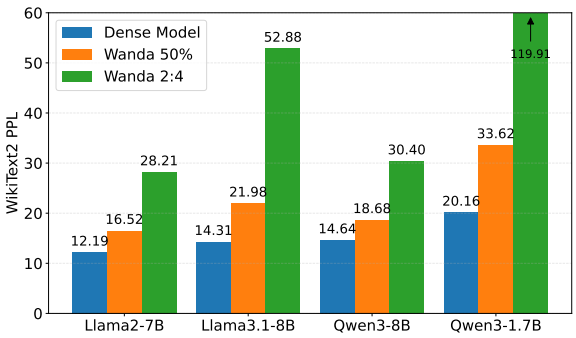

Figure 1: Wikitext 2 perplexity (lower the better) on dense, unstructured, and 2:4 structured pruned models across model family and scales.

3 Method

3.1 Motivation: Accuracy and Efficiency Trade-off in Unstructured *vs* 2:4 Pruning

To better understand the trade-off between model accuracy and deployment efficiency in unstructured pruning and semi-structured pruning, we conduct a set of preliminary studies from both the model and system perspectives. Specifically, we compare unstructured and 2:4 structured in terms of WikiText-2 perplexity by using the representative method Wanda (Sun et al., 2024), and measure prefill throughput and memory usage on Llama2-7B

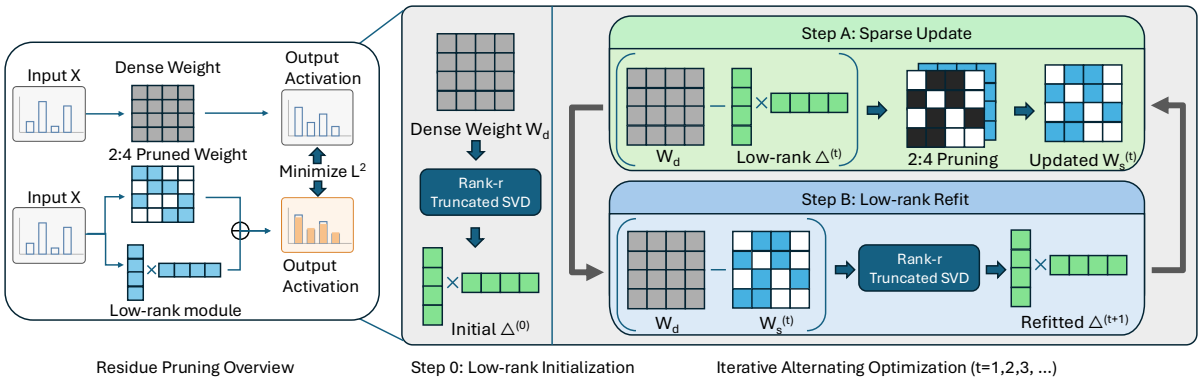

Figure 2: Overview of Residue Pruning and alternating prune-fit optimization.

using a single NVIDIA A6000 GPU. The results are summarized in Table 1 and Fig. 1.

From the **model-quality perspective**, 2:4 structured pruning causes substantially larger degradation than unstructured pruning at the same sparsity level. We make two further observations from Fig. 1: **1) the quality gap becomes larger for more recent LLMs than for earlier ones.** For example, when moving from Llama2-7B to Llama3.1-8B, the gap between structured and unstructured pruning increases by 24.65, corresponding to an approximately 87% increase. **2) The gap becomes more pronounced for smaller LLMs.** For instance, when moving from Qwen3-8B to Qwen3-1.7B, the gap further increases by 88.6, corresponding to a 2.91× enlargement.

From the **deployment perspective**, however, **2:4 structured pruning offers substantial system-level benefits.** As shown in Table 1, 2:4 sparsity improves prefill throughput by around 18% and reduces memory cost by about 41% comapred to dense model. In contrast, 50% unstructured pruning delivers almost no measurable end-to-end gain in our setting. This mismatch is consistent with current GPU software and hardware support: NVIDIA sparse tensor cores and the cuSPARSELt stack directly accelerate 2:4 semi-structured GEMM, whereas conventional unstructured SpMM kernels often fail to outperform dense GEMM until sparsity becomes extremely high (e.g., above 95%) (Wang et al., 2021; Wang, 2020), well beyond the moderate sparsity regimes typically preferred for preserving model quality (sparsity <=70%) (Yin et al., 2025; Sun et al., 2024).

Taken together, these observations reveal the central deployment gap of pure pruning: unstructured pruning is more accuracy-friendly but offers little practical speedup at moderate sparsity, whereas 2:4 structured pruning is more hardware-efficient but incurs substantially larger quality loss. This motivates a more practical direction: preserving the deployment-friendly 2:4 sparse module while introducing lightweight compensation to recover the lost accuracy.

3.2 Residue Pruning: Formulation and Optimization

To close the accuracy gap between 2:4 structured pruning and unstructured pruning while preserving the deployment benefits of 2:4 sparsity, we propose Residue Pruning, which decomposes each dense weight matrix into the combination of a 2:4 sparse matrix and a low-rank matrix. Unlike prior *sparse-plus-low-rank* methods, such as TLR (Shen et al., 2025) and OATS (Zhang and Papyan, 2025), which mainly optimize weight-space reconstruction loss to recover the pruned model, our method formulates compression in the layer-wise output space, where pruning errors are characterized by their effect on layer behavior rather than by weight reconstruction alone. Since layer outputs more directly reflect the functional importance of pruning errors to model performance, this formulation enables more effective recovery under the 2:4 sparsity constraint. Visualized as Figure 2, we present the optimization formulation and the corresponding procedure as follows.

Problem formulation. Consider a linear layer with calibration inputs $X \in \mathbb{R}^{N\times m}$ and dense weight matrix $W_d \in \mathbb{R}^{m\times n}$. Our goal is to construct a deployable approximation of $\hat{W}$ using a hardware-friendly sparse module as a compact low-rank component:

$$\hat{W} = W_s + \Delta = W_s + BA, \qquad (1)$$

where $W_s \in \mathbb{R}^{m\times n}$ satisfies the 2:4 sparsity con-

straint, and $\Delta = BA$ is a rank-constrained residue with $B \in \mathbb{R}^{m\times r}$ and $A \in \mathbb{R}^{r\times n}$.

Instead of reconstructing the removed weights directly in weight space, we optimize W_d to preserve the layer outputs on calibration inputs. Specifically, we define the layer-wise output reconstruction error as

$$\begin{aligned} \min_{W_s,\Delta} \quad & \frac{1}{2}\left\|X(W_s+\Delta-W_d)\right\|_F^2 \\ \text{s.t.} \quad & W_s \in C_{2:4}, \\ & \operatorname{rank}(\Delta) \le r, \end{aligned} \tag{2}$$

where $C_{2:4}$ denotes the 2:4 structured sparsity constraint, and r is the rank budget of the low-rank component.

Alternating prune-fit optimization. Directly optimizing Eq. (2) is challenging due to the discrete 2:4 mask selection and the non-convex rank constraint. To address this, we adopt an alternating optimization strategy that updates the sparse and low-rank components in turn for each LLM decoding layer. Starting from an initial low-rank approximation, each iteration consists of two steps: a **sparse update**, which prunes the current target under the 2:4 constraint, and a **low-rank update**, which refits the remaining error under the same rank budget. This alternating design allows the two components to better complement each other, with the sparse branch preserving the deployment-friendly main structure and the low-rank branch compensating for the most critical pruning-induced errors. The detailed algorithm is illustrated in Alg. 1.

Step 0: *Low-rank-first* initialization. We first initialize the low-rank component $\Delta^{(0)}$ by applying rank-r truncated SVD to the dense weight matrix W_d. This step captures the dominant low-rank structure of the dense matrix under the given rank budget. In the subsequent optimization, the low-rank component preserves this global structure, while the 2:4 sparse component focuses on compensating for the remaining reconstruction error.

This differs from prior *sparse-first* methods (Shen et al., 2025; Zhang and Papyan, 2025), which first derive a sparse matrix from the dense weight and then use a low-rank term for post-hoc compensation, and empirically achieve better performance. This is motivated by the fact that truncated SVD yields the optimal rank-r approximation of the dense matrix, thereby preserving its dominant global structure under the given rank budget. If low-rank fitting is applied only after 2:4 pruning, part of this structure may already be distorted by the

Algorithm 1 Alternating prune-fit procedure.

Require: Dense weight W_d; calibration data D with N samples; 2:4 constraint $C_{2:4}$; rank r; max iters T; tolerance ϵ.

Ensure: Sparse weight W_s and low-rank residue Δ.

Compute $g_j \leftarrow \|X_{:,j}\|_2^2/N$ using calibration set D
$\Delta^{(0)} \leftarrow \mathrm{SVD}_r(W_d)$ ▷ Init Δ
$\mathrm{err}^{(0)} \leftarrow +\infty$ ▷ Init err
for $t \leftarrow 0$ to $T-1$ **do**
  $R^{(t)} \leftarrow W_d - \Delta^{(t)}$ ▷ Residue
  $M_{ij}^{(t)} \leftarrow \left|R_{ij}^{(t)}\right| \sqrt{g_j}$ ▷ Weight Saliency
  $W_s^{(t)} \leftarrow \textsc{Prune2:4}(R^{(t)}, M^{(t)}, C_{2:4})$
  $U^{(t)} \leftarrow W_d - W_s^{(t)}$ ▷ Refit target
  $\Delta^{(t+1)} \leftarrow \mathrm{SVD}_r(U^{(t)})$ ▷ Rank-r SVD
  $\mathrm{err}^{(t+1)} \leftarrow \frac{\|W_d-(W_s^{(t)}+\Delta^{(t+1)})\|_F}{\|W_d\|_F}$
  if $\left|\mathrm{err}^{(t)} - \mathrm{err}^{(t+1)}\right| < \epsilon$ **then**
    break ▷ Early stop
  end if
end for
return $W_s^{(t)}, \Delta^{(t+1)}$

structured sparsity constraint, making subsequent recovery less effective. Empirically, this low-rank-first initialization leads to better final performance than sparse-first alternatives. More details are provided in Sec.A.1.

Step A: 2:4 pruning. Fixing the current low-rank component $\Delta^{(t)}$, where t is the update step, the optimization in Eq. (2) reduces to

$$\min_{W_s\in C_{2:4}} \frac{1}{2}\left\|X\left(W_s+\Delta^{(t)}-W_d\right)\right\|_F^2 \tag{3}$$

Equivalently, it seeks a 2:4 sparse approximation of the residual target

$$R^{(t)} = W_d - \Delta^{(t)} \tag{4}$$

This yields a standard structured pruning formulation, except that the pruning target is no longer the original dense matrix W_d, but the residual target after removing the contribution of the low-rank component. To perform pruning efficiently, we adapt the activation-aware criterion of Wanda. Let $R^{(t)} \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$ denote the residual target at iteration t, and let $X \in \mathbb{R}^{N\times d_{\text{in}}}$ be the matrix of input activations collected from the calibration set. Here, $R_{ij}^{(t)}$ denotes the weight connecting the j-th input feature to the i-th output feature, and $X_{:,j}$ denotes the activations of the j-th input feature across the N calibration samples. We define the saliency

score of weight $R_{ij}^{(t)}$ as

$$M_{ij}^{(t)} = \left|R_{ij}^{(t)}\right| \sqrt{g_j}, \qquad g_j = \frac{1}{N} \|X_{:,j}\|_2^2 .$$

Note that g_j depends only on the input index j, reflecting the average activation energy of the j-th input feature. We then apply row-wise 2:4 grouping to prune the two smallest entries in each group.

Step B: budgeted low-rank refit. Fixing $W_s^{(t)}$ reduces Eq. (2) to

$$\min_{\mathrm{rank}(\Delta)\le r} \frac{1}{2} \left\| X\left(\Delta - U^{(t)}\right) \right\|_F^2, \qquad U^{(t)} = W_d - W_s^{(t)}. \tag{5}$$

This subproblem updates the low-rank component under the rank budget while keeping the sparse component fixed. Equivalently, it fits a rank-r approximation to the remaining target $U^{(t)}$, corresponding to the portion of the dense weight matrix not captured by the current 2:4 sparse component. In practice, however, solving Eq. (5) exactly in the layer-output space is expensive. We therefore approximate this step using the rank-r truncated SVD of $U^{(t)}$, and store the resulting low-rank component in factorized form $\Delta^t = B^t A^t$. Furthermore, we use a weight-space proxy for early stopping

$$\mathrm{err}^{(t)} = \frac{\|W_d - (W_s^{(t)} + \Delta^{(t)})\|_F}{\|W_d\|_F},$$

and stop when its improvement falls below a tolerance, i.e.,

$$|\mathrm{err}^{(t+1)} - \mathrm{err}^{(t)}| < \epsilon,$$

or when the maximum iteration budget is reached.

3.3 Efficient Execution of Residue Pruned LLMs

Deployment challenge of *sparse-plus-low-rank* decomposition. Although incorporating a low-rank component improves the quality of 2:4 structured pruning, the resulting decomposition of $Y = XW_s + XBA$ introduces extra inference cost compared with using a 2:4 sparse branch alone. In particular, beyond the main 2:4 sparse branch, inference must additionally execute the low-rank branch and merge the two outputs. While the low-rank branch adds only 4.94% of parameters over the dense model, we observe that its runtime overhead is far from negligible in practice. As shown in Figure 3, with rank r = 128, a direct PyTorch implementation reduces prefill throughput to 5517.7 token/s, which is even 6% lower than the dense model on a modern GPU.

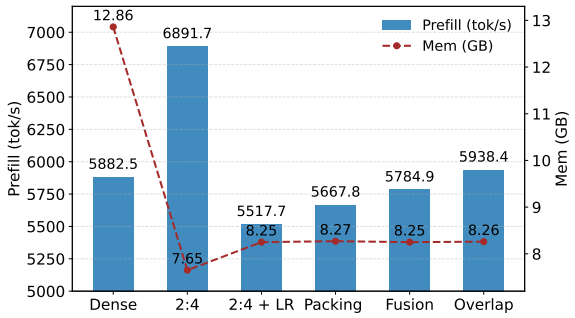

Figure 3: Prefill throughput and memory footprint analysis on Llama2-7B. The rank r is set to 128.

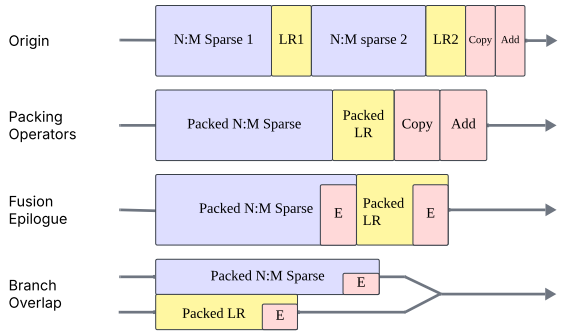

Figure 4: Visualization of the proposed three-level execution scheme on an example two-layer model. "LR" refers to low-rank modules. "E" marks the customized epilogue operation.

This exposes a key deployment challenge that is overlooked by prior *sparse-plus-low-rank* methods (Shen et al., 2025; Zhang and Papyan, 2025): a low-rank branch can recover quality, but if executed naively, it can erase the system advantage of structured sparsity. To mitigate this issue, we develop a three-level system optimization, shown in Figure 4, including projection packing, fused epilogues, and branch overlap, to reduce the inference cost of the hybrid sparse-plus-low-rank computation.

Operator-level packing of shared-input projections. A major source of inference overhead in the naive implementation comes from operator fragmentation. In decoding blocks, several projections share the same input activation, most notably the Q/K/V projections in self-attention and the gate/up projections in the MLP. Executing these projections independently leads to multiple small kernel launches and repeated reads of the same input activation, which lowers hardware utilization and in-

creases launch overhead. This issue is particularly pronounced for the low-rank residue branch: because each projection is further decomposed into factorized low-rank operations, the resulting kernels are smaller and therefore more sensitive to launch inefficiency. Detailed preliminary analysis can be found in Sec.A.2. To mitigate this problem, as illustrated by the Packing Operator in Figure 4, we pack projections with a shared input into grouped operators and execute them jointly within each branch. This design amortizes kernel launch overhead, improves data reuse on the shared input activation, and increases effective GPU utilization. The corresponding end-to-end effect is shown as "Packing" in Figure 3.

Epilogue fusion of packing-induced overheads. Although packing shared-input projections reduces operator fragmentation, it introduces post-processing overhead. In self-attention, packed outputs must still be split into Q, K, and V before attention; similarly, packed gate/up projections in the MLP must be rearranged before use. Performing these steps in Python on the CPU increases host involvement and slows execution. To avoid this, we fuse post-processing into the epilogue of the packed projection kernels. As illustrated by the Fusion Epilogue in Figure 4, we implement two specialized epilogues, one for QKV and the other for gate/up, so that output splitting and layout handling are completed in-kernel rather than by separate follow-up operations. This reduces CPU involvement and packing overhead, enabling clear end-to-end throughput gains, as shown by the "Fusion" result in Figure 3.

Branch-level overlap across sparse and low-rank branches. After operator packing and epilogue fusion, the sparse module and the low-rank residue branch are individually more efficient, but they still introduce extra latency if executed strictly sequentially. Importantly, both branches take the same input activation X and remain independent until their outputs are merged, which creates an opportunity for parallel execution. To exploit this property, we launch the sparse branch and the low-rank branch on separate CUDA streams and synchronize only at the final accumulation stage, as illustrated by the Branch Overlap design in Figure 4. In this way, part of the low-rank computation can be hidden behind the execution of the sparse module, reducing the effective critical-path overhead of the hybrid decomposition. The resulting end-to-end improvement is reflected by the "Overlap" result in Figure 3.

4 Experiments

Models and Evaluation. To demonstrate the effectiveness of the proposed method, we evaluate our method across multiple model families and various model scales, including Llama2-7B (Touvron et al., 2023), Llama3.1-8B (Grattafiori et al., 2024), Qwen3-8B (Yang et al., 2025), and Qwen3-1.7B (Yang et al., 2025). Following previous pruning works (Ma et al., 2023; An et al., 2023; Wang et al., 2025), we evaluate the pruned models on two categories of tasks: text generation perplexity on WikiText2 (Merity et al., 2016) and zero-shot accuracy on commonsense reasoning (CMQA) tasks, including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2019), ARC-Easy and ARC-Challenge (Clark et al., 2018), and OpenBookQA (Mihaylov et al., 2018) via lm-eval-harness benchmark tool (Gao et al., 2024). We report average accuracy in this section. Detailed per-task accuracy can be found in Sec.A.4.

Baselines. We compare our method against representative baselines across three categories: (1) unstructured pruning **Wanda** (Sun et al., 2024), (2) 2:4 structured pruning **Wanda 2:4**, (3) methods using low-rank recovery in weight space upon 2:4 structured pruning: **one-shot SVD**, **TLR** (Shen et al., 2025), and **OATS** (Zhang and Papyan, 2025).

Implementation Details. We set the random seed to 0, low-rank residue rank r to 32 and 128, the maximum number of alternating prune-fit iterations to $T = 200$, and the early stopping tolerance to $\epsilon = 10^{-8}$. Calibration sets are randomly collected from C4 (Raffel et al., 2023), consisting of 128 sampled sequences of length 2048 from the training split, following the standard post-training pruning setup (Sun et al., 2024; An et al., 2023; Shen et al., 2025). All methods share the same calibration set.

4.1 Experimental Results

Perplexity and zero-shot accuracy. Table 2 summarizes the overall performance of different recovery methods. Compared with Wanda 2:4, our method effectively restores performance degraded by structured pruning. Even with a limited low-rank budget, it consistently improves both perplexity and zero-shot CMQA accuracy across all evaluated models. This effect is especially evident in smaller models, suggesting that it recovers

Table 2: Comparison of different pruning methods on text generation perplexity and reasoning accuracy.

Method	Sparsity	Perplexity on Wikitext2 ↓				Zero-shot CMQA ACC (%) ↑			
		Llama2	Llama3.1	Qwen3		Llama2	Llama3.1	Qwen3	
		7B	8B	8B	1.7B	7B	8B	8B	1.7B
Dense	0	12.19	14.31	14.64	20.16	66.72	70.79	70.41	62.55
Wanda	0.5	16.52	21.98	18.68	33.62	63.47	64.45	67.33	54.52
Wanda 2:4	2:4	28.21	52.88	30.40	119.91	54.53	51.25	60.10	43.73
One-shot SVD	2:4 + r32	26.53	53.58	30.99	97.75	55.13	52.21	59.33	44.73
TLR	2:4 + r32	26.06	48.12	29.99	93.57	56.15	52.56	59.58	45.40
OATS	2:4 + r32	24.10	40.43	26.65	57.36	56.59	**56.18**	61.91	47.94
Ours	2:4 + r32	**21.87**	**37.31**	**25.86**	**55.32**	**58.72**	56.10	**62.03**	**48.57**
One-shot SVD	2:4 + r128	23.35	43.05	27.24	61.53	58.03	54.35	61.45	48.08
TLR	2:4 + r128	21.38	35.37	25.07	48.59	59.47	56.51	63.14	51.93
OATS	2:4 + r128	16.93	25.38	20.89	**29.82**	62.10	62.16	64.71	56.94
Ours	2:4 + r128	**16.58**	**24.27**	**20.44**	29.85	**62.54**	**63.15**	**65.39**	**58.94**

Table 3: Comparison of different pruning methods on prefill throughput and memory usage. Throughput and memory are tested on Llama2-7B

Method	Sparsity	Throughput (tok/s) ↑	Memory (GB) ↓
OATS	2:4 + r32	5685.9	7.80
TLR	2:4 + r32	5685.9	7.80
Ours	2:4 + r32	6048.2	7.81
OATS	2:4 + r128	5517.7	8.25
TLR	2:4 + r128	5517.7	8.25
Ours	2:4 + r128	5938.4	8.26

Table 4: Comparison of different pruning methods on total time cost and memory requirement on Llama2-7B model. The rank r is set to 32.

Method	Time cost	Memory
Oneshot SVD	3 min	18328MiB
TLR	3 min	20430 MiB
OATS	888.32 min	17700 MiB
Ours	37 min	18123 MiB

pruning-induced degradation with few additional low-rank parameters. Moreover, our method remains competitive with its unstructured variant: at rank $r = 128$, it achieves performance close to that of the unstructured variant.

Compared with other recovery baselines, our method consistently outperforms them under both small and large rank budgets, showing greater robustness across settings. This advantage is more pronounced in the low-budget regime, where recovery is most challenging. These results demonstrate that our recovery mechanism restores pruned model capability in a parameter-efficient manner.

System benefits in memory and throughput. Furthermore, from a systems perspective, our method consistently achieves higher prefill throughput than the competing recovery methods, while incurring almost no extra memory cost. Across both rank budgets, it delivers about 6.4%–7.6% higher throughput, with minimal memory usage increases. This shows that our method improves runtime efficiency without sacrificing the memory advantages of the sparse recovery setting.

Method Cost. Beyond delivering the best recovery performance, our method is also highly efficient. As shown in Table 4, our method incurs substantially lower optimization cost than the strongest competing baseline. In particular, compared with OATS, our method reduces the time cost by 851.32 minutes, corresponding to a 24.0× speedup. This result shows that our method not only achieves superior recovery quality but also offers a much more practical and efficient solution in terms of optimization overhead.

5 Conclusion

In this work, we propose *Residue Pruning*, a practical compression framework that combines a hardware-friendly 2:4 sparse module with a compact low-rank compensation module for LLM deployment. Residue Pruning formulates compression in the activation space and optimizes the two modules with an alternating prune-fit procedure to more effectively recover pruning-induced errors under a strict budget. We further introduce an efficient execution scheme with shared-input projection packing, fused epilogues, and branch overlap. Extensive experiments show that Residue Pruning consistently outperforms existing sparse-plus-low-rank baselines, while improving throughput and preserving the memory benefits of 2:4 sparsity.

Limitations

One limitation of our current system design is that, although our execution optimizations already reduce the overhead of low-rank compensation and make the hybrid sparse-plus-low-rank design practical, there is still room to further reduce its cost. For example, one could introduce additional sparsity into the low-rank branch to make it more lightweight, which we leave for future work. Additionally, while our method already reduces memory usage and improves throughput, we will try combining it with quantization to validate its effectiveness in a joint compression setting. Exploring this direction remains valuable for future exploration.

References

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. 2023. Fluctuation-based adaptive structured pruning for large language models. *Preprint*, arXiv:2312.11983.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. Piqa: Reasoning about physical commonsense in natural language. *Preprint*, arXiv:1911.11641.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *Preprint*, arXiv:1905.10044.

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

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. 2024. Maskllm: Learnable semi-structured sparsity for large language models. *Preprint*, arXiv:2409.17481.

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

Leo Gao, Jonathan Tow, Abbasi, and 1 others. 2024. The language model evaluation harness.

Aaron Grattafiori, Abhimanyu Dubey, and 1 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. 2024. Shortened llama: Depth pruning for large language models with comparison of retraining methods. *Preprint*, arXiv:2402.02834.

Yixiao Li, Yifan Yu, Qingru Zhang, Chen Liang, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2023. Losparse: Structured compression of large language models based on low-rank and sparse approximation. *Preprint*, arXiv:2306.11222.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Preprint*, arXiv:2305.11627.

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

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

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *Preprint*, arXiv:1809.02789.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer. *Preprint*, arXiv:1910.10683.

Rajarshi Saha, Naomi Sagan, Varun Srivastava, Andrea J. Goldsmith, and Mert Pilanci. 2024. Compressing large language models using low rank and low precision decomposition. *Preprint*, arXiv:2405.18886.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *Preprint*, arXiv:1907.10641.

Li Shen, Anke Tang, Yong Luo, Tao Sun, Han Hu, and Xiaochun Cao. 2025. Targeted low-rank refinement: Enhancing sparse language models with precision. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 54457–54475. PMLR.

Sharath Turuvekere Sreenivas, Saurav Muralidharan, and 1 others. 2024. Llm pruning and distillation in practice: The minitron approach. *Preprint*, arXiv:2408.11796.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. A simple and effective pruning approach for large language models. *Preprint*, arXiv:2306.11695.

Hugo Touvron and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Yang Wang, Chen Zhang, Zhiqiang Xie, Cong Guo, Yunxin Liu, and Jingwen Leng. 2021. Dual-side sparse tensor core. *Preprint*, arXiv:2105.09564.

Ziheng Wang. 2020. Sparsert: Accelerating unstructured sparsity on gpus for deep learning inference. *Preprint*, arXiv:2008.11849.

Ziyan Wang, Enmao Diao, Qi Le, Pu Wang, Minwoo Lee, Shu ping Yeh, Evgeny Stupachenko, Hao Feng, and Li Yang. 2025. From local to global: Revisiting structured pruning paradigms for large language models. *Preprint*, arXiv:2510.18030.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, Ajay Jaiswal, Mykola Pechenizkiy, Yi Liang, Michael Bendersky, Zhangyang Wang, and Shiwei Liu. 2025. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity. *Preprint*, arXiv:2310.05175.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *Preprint*, arXiv:1905.07830.

Stephen Zhang and Vardan Papyan. 2025. Oats: Outlier-aware pruning through sparse and low rank decomposition. *Preprint*, arXiv:2409.13652.

A Appendix

A.1 Ablation of Initialization Strategy

Table 5: Initialization ablation on Llama2-7B. We compare sparse-first and SVD-first initialization under different low-rank budgets. Lower PPL is better.

Rank r	Sparse-first PPL	SVD-first PPL
0	28.31	28.31
32	22.41	**21.87**
64	19.33	**19.12**
128	17.00	**16.58**
256	15.10	**14.69**
512	13.72	**13.40**

Motivated by the fact that truncated SVD provides the optimal rank-r approximation of a dense matrix, thereby preserving its dominant global structure under a fixed rank budget, we choose to initialize the low-rank module before applying 2:4 pruning.

To verify this intuition, we conduct an ablation study of different initialization strategies on the Llama2-7B model. As shown in Table 5, across all rank budgets, initializing the low-rank module from the dense weights consistently yields better performance. These results further support our design choice.

Table 6: Preliminary operator-level benchmarks of packing on two branches. Results are profiled from Llama2-7B model.

Branch	Latency (ms)↓	Improvement
Structured-sparse	51.535 → 49.832	3.3%
Low-rank residue	8.284 → 7.059	14.8%

A.2 Operator-level analysis

To examine the effect of shared-input projection packing, we conduct operator-level microbenchmarks on the structured-sparse branch and the low-rank residue branch. Table 6 compares the latency before and after packing under the same input setting.

Packing brings clear latency reductions to both branches, improving the structured-sparse branch by 3.3% and the low-rank residue branch by 14.8%. This indicates that shared-input packing is effective for both execution paths, while providing an even larger benefit to the low-rank branch, whose smaller factorized operators are more sensitive to kernel launch overhead. These results justify our use of packing in Sec. 3.3.

A.3 Residue Pruning under Different Rank Budgets

We further analyze residue pruning under different rank budgets from both model quality and system efficiency perspectives. As shown in Fig. 5, in the memory-perplexity trade-off, our method consistently achieves a better Pareto frontier than the low-rank compensation baseline across all rank budgets. Specifically, for a similar memory footprint, our method attains lower perplexity, while for a comparable perplexity level, it requires less memory. This demonstrates that our residue pruning strategy utilizes the rank budget more effectively and delivers a superior accuracy–memory trade-off.

More importantly, the advantage of our method is also evident from the system perspective. In the prefill-perplexity trade-off, our method consistently outperforms other low-rank compensation baselines, yielding a more favorable Pareto frontier between throughput and model quality. We attribute this gain to the proposed three-level system optimization, which enables efficient execution

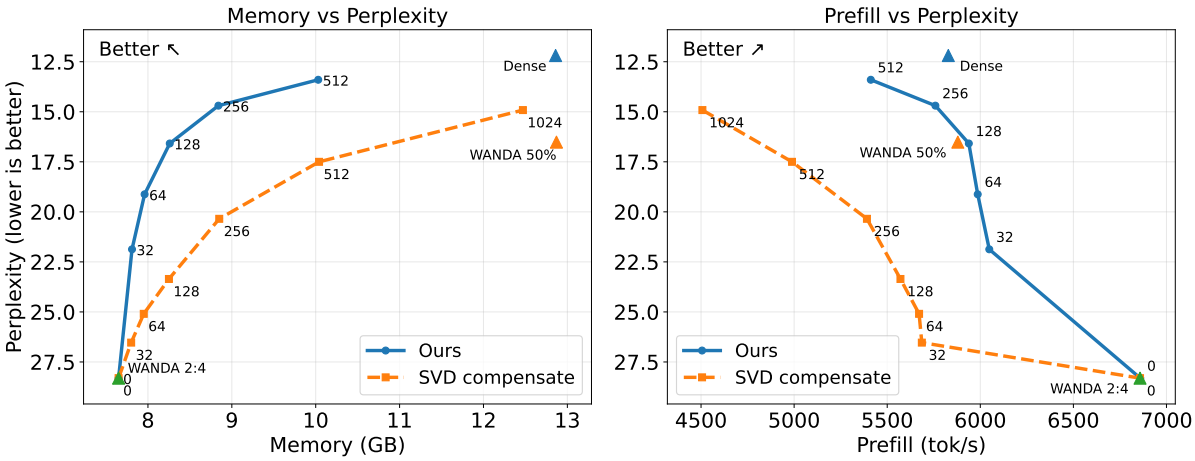

Figure 5: Memory-perplexity-throughput analysis of Residue Pruning under different rank budgets.

of the low-rank residual module and substantially reduces the runtime overhead typically introduced by low-rank compensation. As a result, our method not only improves the compression-quality trade-off, but also translates these gains into better end-to-end serving efficiency.

A.4 Detailed zero-shot CMQA accuracy

Task-wise zero-shot reasoning accuracy. To complement the averaged zero-shot reasoning results in Table 2, we further report task-wise zero-shot accuracy on the seven evaluation tasks in Table 7, 8, 9, 10. Overall, the task-level results are consistent with the observations in the main text. Compared with Wanda 2:4, our method improves performance on most tasks across different model sizes and rank budgets, confirming that the gain in averaged reasoning accuracy is broad-based rather than driven by only a few tasks. This trend is particularly clear for smaller models and in the low-rank regime, where recovery is more challenging, and our method still provides stable improvements.

In addition, compared with other recovery baselines, our method remains competitive on nearly all tasks and more consistently achieves stronger recovery under limited rank budgets. While the magnitude of improvement varies across tasks, the overall pattern matches the main experimental findings: our method restores pruning-induced degradation in a robust and parameter-efficient manner.

A.5 LLM Usage

In accordance with the ARR AI Writing/Coding Assistance Policy, we disclose that LLM-based tools (e.g., ChatGPT) were used solely to aid in polishing the writing and improving the clarity of exposition. They were not used for research ideation, experimental design, data analysis, or other substantive contributions. All scientific content, results, and conclusions are the responsibility of the authors.

Table 7: Detailed task-wise zero-shot accuracy of different pruning methods on Llama2-7B.

Method	Sparsity	BoolQ	PIQA	Hellaswag	Winogrande	ARC_e	ARC_c	OBQA	AVG
Dense	0	77.77	79.05	75.98	69.14	74.54	46.33	44.20	66.72
Wanda	0.5	75.29	76.77	70.67	67.01	69.23	42.75	42.60	63.47
Wanda 2:4	2:4	68.04	71.22	55.41	62.35	57.66	31.66	35.40	54.53
One-shot SVD	2:4 + r32	68.75	71.11	56.86	63.38	58.50	32.51	34.80	55.13
TLR	2:4 + r32	71.44	71.87	57.59	63.30	60.44	33.02	35.40	56.15
OATS	2:4 + r32	68.56	72.03	59.86	63.61	60.73	34.73	36.60	56.59
Ours	2:4 + r32	72.48	72.74	62.52	64.80	63.93	37.20	37.40	58.72
One-shot SVD	2:4 + r128	72.54	73.29	60.19	65.59	62.21	36.77	35.60	58.03
TLR	2:4 + r128	73.91	74.05	62.94	65.59	64.86	37.12	37.80	59.47
OATS	2:4 + r128	73.24	76.17	68.63	68.51	68.27	39.68	40.20	62.10
Ours	2:4 + r128	76.18	75.90	69.14	68.03	68.06	40.27	40.20	62.54

Table 8: Detailed task-wise zero-shot accuracy of different pruning methods on Llama3.1-8B.

Method	Sparsity	BoolQ	PIQA	Hellaswag	Winogrande	ARC_e	ARC_c	OBQA	AVG
Dense	0	82.08	81.23	78.88	73.72	81.02	53.58	45.00	70.79
Wanda	0.5	79.42	77.64	69.34	70.88	69.74	43.94	40.20	64.45
Wanda 2:4	2:4	66.94	69.15	49.45	59.43	51.85	29.69	32.20	51.25
One-shot SVD	2:4 + r32	67.95	69.48	51.14	60.93	52.15	31.23	32.60	52.21
TLR	2:4 + r32	67.00	69.37	52.96	61.25	53.45	31.31	32.60	52.56
OATS	2:4 + r32	70.95	70.84	56.93	63.54	60.19	35.41	35.40	56.18
Ours	2:4 + r32	68.78	72.20	59.39	65.35	58.12	34.64	34.20	56.10
One-shot SVD	2:4 + r128	69.02	70.51	55.24	64.96	54.25	33.28	33.20	54.35
TLR	2:4 + r128	68.84	73.07	59.31	66.61	58.08	34.47	35.20	56.51
OATS	2:4 + r128	74.68	76.39	67.72	68.03	68.43	41.47	38.40	62.16
Ours	2:4 + r128	79.08	74.97	68.68	69.69	69.91	42.92	36.80	63.15

Table 9: Detailed task-wise zero-shot accuracy of different pruning methods on Qwen3-8B.

Method	Sparsity	BoolQ	PIQA	Hellaswag	Winogrande	ARC_e	ARC_c	OBQA	AVG
Dense	0	83.06	79.38	78.65	72.77	80.01	56.83	42.20	70.41
Wanda	0.5	79.72	77.97	70.62	70.09	79.55	52.39	41.00	67.33
Wanda 2:4	2:4	76.48	73.18	59.21	64.33	70.83	40.87	35.80	60.10
One-shot SVD	2:4 + r32	73.36	72.42	58.52	64.48	69.49	40.61	36.40	59.33
TLR	2:4 + r32	73.39	72.42	59.23	65.98	69.53	40.53	36.00	59.58
OATS	2:4 + r32	77.06	74.32	61.39	66.38	72.56	43.86	37.80	61.91
Ours	2:4 + r32	70.28	74.10	62.38	67.01	76.14	45.90	38.40	62.03
One-shot SVD	2:4 + r128	75.81	73.34	60.79	66.06	73.57	43.94	36.60	61.45
TLR	2:4 + r128	78.47	75.14	63.10	68.19	74.75	44.71	37.60	63.14
OATS	2:4 + r128	69.48	76.71	67.62	68.43	79.08	50.26	41.40	64.71
Ours	2:4 + r128	75.20	76.12	69.36	68.82	77.86	49.15	41.20	65.39

Table 10: Detailed task-wise zero-shot accuracy of different pruning methods on Qwen3-1.7B.

Method	**Sparsity**	**BoolQ**	**PIQA**	**Hellaswag**	**Winogrande**	**ARC_e**	**ARC_c**	**OBQA**	**AVG**
Dense	0	79.05	75.84	66.47	64.01	68.43	45.05	39.00	62.55
Wanda	0.5	67.06	70.08	52.89	58.48	62.79	35.92	34.40	54.52
Wanda 2:4	2:4	62.51	60.99	35.66	51.62	45.71	23.21	26.40	43.73
One-shot SVD	2:4 + r32	62.51	62.51	37.31	53.28	46.55	23.98	27.00	44.73
TLR	2:4 + r32	62.42	63.38	37.90	53.67	48.32	24.91	27.20	45.40
OATS	2:4 + r32	65.57	65.72	42.39	55.25	50.25	27.82	28.60	47.94
Ours	2:4 + r32	63.24	66.43	43.96	54.54	52.86	28.58	30.40	48.57
One-shot SVD	2:4 + r128	64.31	65.89	42.93	55.09	52.48	27.47	28.40	48.08
TLR	2:4 + r128	69.24	67.57	46.48	57.30	60.06	30.89	32.00	51.93
OATS	2:4 + r128	72.05	71.76	55.01	61.72	65.95	36.86	35.20	56.94
Ours	2:4 + r128	73.27	70.73	56.22	62.75	69.57	40.44	39.60	58.94