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), but practical LLM pruning faces a trade-off between accuracy and deployability. Unstructured pruning often preserves quality better, while 2:4 structured pruning provides real memory and inference benefits on modern GPUs but incurs larger accuracy degradation. Recent sparse-plus-low-rank methods help recover this degradation, yet they still face two practical limitations: suboptimal accuracy–cost trade-offs during pruning and overlooked runtime overhead from the low-rank branch. To address these limitations, we propose Residue Pruning, a practical sparse-plus-low-rank framework for deployable 2:4 LLM pruning. At the algorithm level, Residue Pruning introduces an efficient alternating prune-fit recipe with low-rank-first initialization, activation-aware 2:4 pruning, and weight-space low-rank refitting. At the system level, we further design a three-level execution pipeline with 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 reduces 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. Among different sparsity formats, *semi-structured* N:M pruning, which keeps N nonzero weights within every group of M weights, offers a regular sparsity pattern that is easier to accelerate than unstructured sparsity (Fang et al., 2024; Zhang and Papyan, 2025; Sun et al., 2024). In particular, *2:4 semi-structured sparsity* has become one of the most practically important settings due to its direct support from modern GPU software and hardware stacks.

Despite its deployment advantages, N:M pruning often incurs substantially larger accuracy degradation than unstructured pruning at the same sparsity level, due to its strict structural constraints. More importantly, we observe that this quality gap becomes even more pronounced for newer and smaller LLMs, which are especially important for resource-constrained deployment. For example, 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×, while the gap further enlarges by 7.36× when moving from Qwen3-8B (Yang et al., 2025) to Qwen3-1.7B. These observations motivate the following practical question: *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; Shen et al., 2025) provide a promising 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 post-training 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 sparse-plus-low-rank recovery. However, these methods still face two important practical challenges. **First, at the algorithm level, the trade-off between accuracy and pruning cost**

remains suboptimal. TLR follows a direct fixed-mask recovery paradigm, which is efficient but provides limited accuracy improvement under strict 2:4 sparsity. OATS achieves a stronger recovery by incorporating activation-derived Hessian signals into weight reconstruction, but its low-rank update requires repeated truncated-SVD refits over large residual matrices, making the pruning process computationally expensive at model scale. Moreover, with limited calibration data relative to the model's original training data, activation-guided refitting can suffer from overfitting, leading to suboptimal overall performance. **Second, at the system level, existing sparse-plus-low-rank methods largely overlook the deployment overhead introduced by the low-rank branch**: although the low-rank branch adds only a small number of parameters, its runtime overhead can be far from negligible in practice and significantly weaken the efficiency gains of the sparse module.

To address these limitations, we propose **Residue Pruning**, a practical sparse-plus-low-rank framework tailored for 2:4 structured pruning. Building upon the sparse-plus-low-rank decomposition paradigm, in which each dense weight matrix is represented as a 2:4 sparse matrix with a compact low-rank residual, our goal is to improve the practical trade-off between accuracy and pruning cost, while further enhancing deployment efficiency. **At the algorithm level**, following an alternating optimization scheme for updating the sparse and low-rank branches, we develop an effective and efficient optimization recipe for this decomposition under the strict 2:4 constraint. 1) Instead of starting from a fixed 2:4 sparse branch via pruning and applying low-rank recovery afterward, we initialize the low-rank branch from the dense weights before 2:4 pruning, which better preserves the dominant global information of the original matrix. 2) We use calibration activations to guide 2:4 pruning for generating the sparse branch, allowing the sparse branch to preserve more important weights for output reconstruction. 3) After each 2:4 pruning step, we refit the remaining residual with an efficient rank-r low-rank approximation in weight space, avoiding overfitting and substantially reducing the optimization cost of repeated truncated SVD. **At the system level**, 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 demonstrate the effectiveness and practicality of Residue Pruning. Compared with OATS, Residue Pruning reduces pruning-stage optimization time by 24.0× on Llama2-7B at rank $r = 32$. It also improves model quality, achieving up to 2.13% higher average zero-shot CMQA accuracy and up to 9.3% lower WikiText-2 perplexity, while its optimized execution improves A100 prefill throughput by up to 19.0% over naive *sparse-plus-low-rank* deployment with negligible memory overhead.

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 et al., 2025), 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 under the guidance of task-wise loss with the sparse and low-rank components jointly optimized to preserve task performance. In the post-training recovery area, 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. While these methods demonstrate the promise of sparse-plus-low-rank recovery, the practical trade-off among pruning quality, pruning cost, and deployment efficiency remains insufficiently explored. Our method addresses this gap by jointly considering pruning quality, optimization cost, and deployment efficiency, coupling efficient weight-space low-rank residue refitting with 2:4 sparse structure selection while explicitly reducing the runtime overhead of the low-rank branch.

Table 1: Throughput–perplexity–memory trade-off of current unstructured pruning methods (marked as unstr.) and structured pruning methods (marked as str.) on Llama2-7B using an A100 GPU.

Method	Prefill (token/s) ↑	PPL ↓	Mem (GB) ↓
Dense	8572.4	**12.1907**	12.86
Wanda 2:4 (str.)	**9262.4**	28.2094	**7.58**
Wanda 50% (unstr.)	8497.3	16.5170	12.86

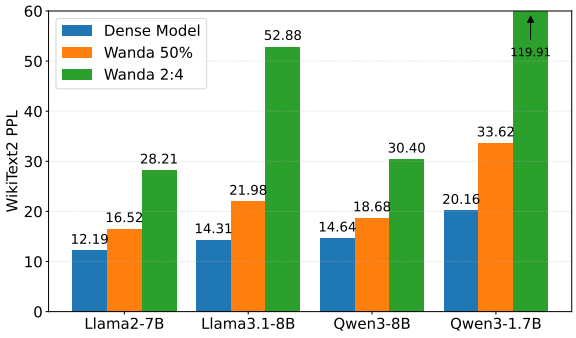

Figure 1: Wikitext-2 perplexity (lower the better) on dense, unstructured, and 2:4 structured pruned models across model families and scales.

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 pruning 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 using a single NVIDIA A100 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**

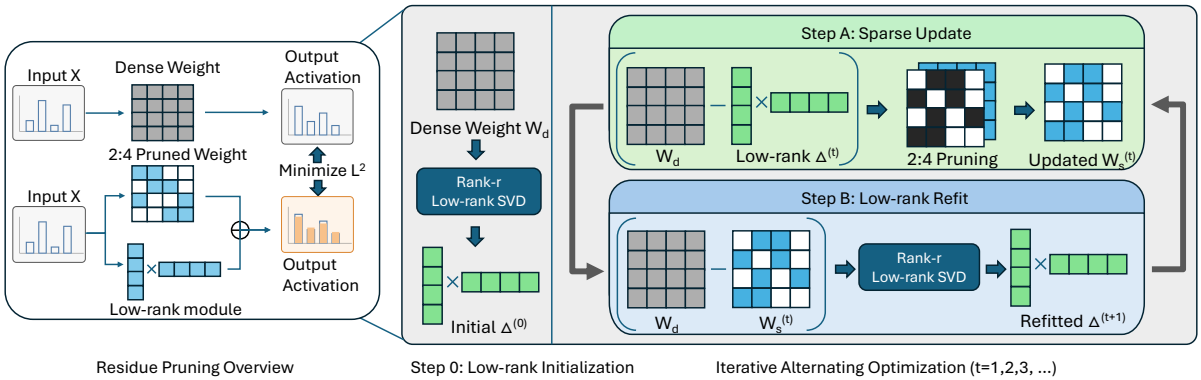

Figure 2: Overview of Residue Pruning and alternating prune-fit optimization.

more recent LLMs than for earlier ones. For example, when moving from Llama2-7B to Llama3.1-8B, the PPL gap between structured and unstructured pruning increases from 11.69 to 30.90, corresponding to a 2.64× enlargement. **2) The gap becomes more pronounced for smaller LLMs.** For instance, when moving from Qwen3-8B to Qwen3-1.7B, the gap further increases from 11.72 to 86.29, corresponding to a 7.36× 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 8% and reduces memory cost by about 41% compared to the 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).

3.2 Residue Pruning: Formulation and Optimization

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 $\hat{W}$ of W_d using a hardware-friendly sparse module and a compact low-rank component:

$$\hat{W} = W_s + \Delta = W_s + BA, \tag{1}$$

where $W_s \in \mathbb{R}^{m\times n}$ satisfies the 2:4 sparsity constraint, and $\Delta = BA$ is a rank-constrained residue with $B \in \mathbb{R}^{m\times r}$ and $A \in \mathbb{R}^{r\times n}$.

Following the standard layer-wise reconstruction perspective in pruning (Dong et al., 2017; Frantar and Alistarh, 2023; Sun et al., 2024), the optimization objective of sparse-plus-low-rank can be formulated as:

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

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

Due to the coupled 2:4 sparsity and low-rank constraints, directly solving this objective is challenging. Following prior sparse-plus-low-rank methods (Zhang and Papyan, 2025; Shen et al., 2025), iterative alternating optimization provides a practical strategy for updating the sparse and low-rank branches. At a high level, the optimization alternates between two steps: updating the 2:4 sparse branch while fixing the low-rank branch, and refitting the low-rank branch based on the remaining residual after sparse recovery. This alternating process allows the two branches to progressively complement each other during recovery.

Residue Pruning. To improve the practical trade-off between recovery accuracy and pruning cost, we propose Residue Pruning, a practical optimization recipe for iterative alternating optimization. As shown in Figure 2, 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 a rank budget. The detailed algorithm is shown 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 ma-

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.

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

Compute $g_j \leftarrow \|X_{:,j}\|_2^2/N$ using calibration set D
$\Delta^{(0)} \leftarrow \mathrm{SVD}_{refit}(W_d)$ ▷ Init Δ
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
  $\Delta^{(t+1)} \leftarrow \mathrm{SVD}_{refit}(U^{(t)})$ ▷ Rank-r SVD
end for
return $W_s^{(T-1)}, \Delta^{(T)}$

trix 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. Empirically, our initialization achieves 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 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

$$M_{ij}^{(t)} = \left|R_{ij}^{(t)}\right| \sqrt{g_j}, \qquad g_j = \frac{1}{N} \|X_{:,j}\|_2^2 . \tag{5}$$

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. After fixing the sparse component $W_s^{(t)}$, Eq. (2) becomes an activation-weighted low-rank reconstruction:

$$\min_{\mathrm{rank}(\Delta) \le r} \frac{1}{2} \left\| X\left(\Delta - U^{(t)}\right) \right\|_F^2, \quad U^{(t)} = W_d - W_s^{(t)}. \tag{6}$$

Here, $U^{(t)}$ is the residual target left by the current 2:4 sparse component. Equivalently, by considering the output residual $X\Delta$, the activation-space refit seeks a rank-r approximation to $XU^{(t)}$. The existing activation-aware sparse-plus-low-rank method OATS (Zhang and Papyan, 2025) performs low-rank refitting in activation space using calibration activations. However, such a method heavily depends on limited calibration data, which may not fully represent the original training or deployment distribution. Recent studies further show that calibration choice can substantially affect pruned LLM quality (Ji et al., 2025; Bandari et al., 2024). To examine this, our empirical analysis in Table 2 shows that the activation-space refit achieves a lower calibration reconstruction loss, both on a representative layer and on a whole-model global reconstruction

Table 2: Comparison of different refit strategies. Loss is defined as the normalized magnitude of Eq. (6). Layer loss is measured on a representative layer, namely the Q projection of decoder layer 15 in Llama2-7B. Global loss is computed by averaging the layer-wise losses over the entire model.

Strategy	PPL ↓	Layer Loss ↓	Global Loss ↓
Weight-space	**21.87**	9.195×10^{-3}	1.286×10^{-2}
Activation-space	26.93	$\mathbf{8.242\times10^{-3}}$	$\mathbf{1.045\times10^{-2}}$

metric, but yields a higher WikiText-2 PPL. This suggests that directly fitting the low-rank residue to limited calibration activations can over-specialize to the calibration objective.

Therefore, we use activation information in Step A for activation-aware 2:4 pruning, while performing the low-rank refit in weight space during Step B. Specifically, after the activation-aware sparse update produces $W_s^{(t)}$, we approximate the residual target $U^{(t)}$ under the rank budget:

$$B^{(t+1)}, A^{(t+1)} = \textsc{LowRankRefit}_r\left(U^{(t)}\right), \quad \Delta^{(t+1)} = B^{(t+1)}A^{(t+1)} \approx U^{(t)}. \tag{7}$$

Although the low-rank refit is performed in weight space, it remains coupled with the activation-aware pruning process since the residual $U^{(t)}$ is recomputed after each 2:4 pruning iteration.

To further reduce optimization cost, we implement LowRankRefit as a rank-budgeted factor estimation procedure rather than an accurate truncated SVD. Instead of computing the full singular spectrum before truncation, the refit directly estimates rank-r factors under the target budget. This design is substantially more efficient for iterative LLM pruning, where Step B is repeatedly applied to large weight matrices. Empirically, replacing our refit with accurate truncated SVD only improves WikiText-2 PPL from 21.87 to 21.72, while making single-matrix runtime more than 100× slower.

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 6828.8 token/s, which is even 20.3% lower than the dense model on a modern GPU.

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

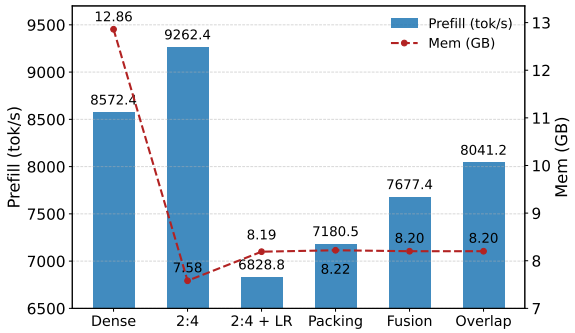

Figure 3: Prefill throughput and memory footprint analysis on Llama2-7B using an A100 GPU. The rank r is set to 128.

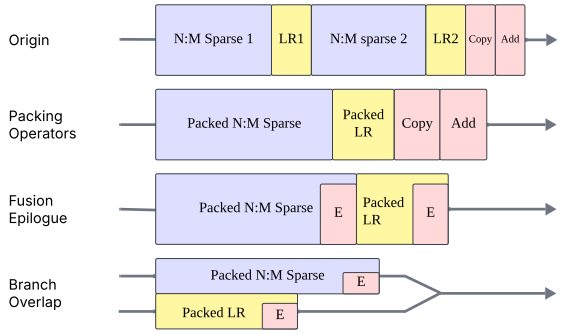

Figure 4: Visualization of the proposed three-level execution scheme on an example two-layer model. "LR" refers to low-rank modules. "E" marks the customized epilogue operation.

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 increases 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 primarily evaluate the pruned models on two categories of tasks: text generation perplexity on WikiText-2 (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.8. Additionally, we provide MMLU 5-shot evaluation results in Sec.A.7. We also provide scalability results on a larger model, Llama2-13B in Sec. A.9.

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) sparse-plus-low-rank methods: **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, and the maximum number of alternating prune-fit iterations to $T = 200$. For Residue Pruning, the number of power iterations for low-rank SVD approximation is set to 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 CMQA accuracy. Table 3 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 pruning-induced degradation with few additional low-rank parameters. Moreover, our method remains competitive with its unstructured variant: at

Table 3: Comparison of different pruning methods on text generation perplexity and reasoning accuracy.

Method	Sparsity	Perplexity on Wikitext-2 ↓				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 4: Comparison of different pruning methods on prefill throughput and memory usage on A100. Throughput and memory are tested on Llama2-7B.

Method	Sparsity	Throughput (token/s) ↑	Memory (GB) ↓
OATS	2:4 + r32	6979.6	7.74
TLR	2:4 + r32	6979.6	7.74
Ours	2:4 + r32	8309.1	7.76
OATS	2:4 + r128	6828.8	8.19
TLR	2:4 + r128	6828.8	8.19
Ours	2:4 + r128	8041.2	8.20

Table 5: Comparison of different pruning methods on total time cost and memory requirement on the Llama2-7B model, measured on A6000. The rank r is set to 32.

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

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. As shown in Table 4, at a batch size of 1 and a sequence length of 512, across both rank budgets, it delivers about 17.8% and 19.0% higher throughput, with minimal increases in memory usage. This shows that our method improves runtime efficiency without sacrificing the memory advantages of the sparse recovery setting. More efficiency results and analysis, including results under different batch sizes and sequence lengths, as well as results on other models and GPUs, can be found in Sec.A.3, Sec.A.4, and Sec.A.5.

Optimization Cost. Beyond delivering the best recovery performance, our method is also highly efficient. As shown in Table 5, our method incurs substantially lower optimization cost than the strongest competing baseline. In particular, compared with OATS, on an A6000 GPU, 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 cost.

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. At the algorithm level, Residue Pruning introduces an efficient alternating prune-fit recipe with low-rank-first initialization, activation-aware 2:4 pruning, and weight-space low-rank refitting. 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.

Abhinav Bandari, Lu Yin, Cheng-Yu Hsieh, Ajay Kumar Jaiswal, Tianlong Chen, Li Shen, Ranjay Krishna, and Shiwei Liu. 2024. Is c4 dataset optimal for pruning? an investigation of calibration data for llm pruning. *Preprint*, arXiv:2410.07461.

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.

Xin Dong, Shangyu Chen, and Sinno Jialin Pan. 2017. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Preprint*, arXiv:1705.07565.

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Yixin Ji, Yang Xiang, Juntao Li, Qingrong Xia, Ping Li, Xinyu Duan, Zhefeng Wang, and Min Zhang. 2025. Beware of calibration data for pruning large language models. *Preprint*, arXiv:2410.17711.

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, Raviraj Joshi, Marcin Chochowski, Ameya Sunil Mahabaleshwarkar, Gerald Shen, Jiaqi Zeng, Zijia Chen, Yoshi Suhara, Shizhe Diao, Chenhan Yu, Wei-Chun Chen, Hayley Ross, Oluwatobi Olabiyi, Ashwath Aithal, Oleksii Kuchaiev, Daniel Korzekwa, Pavlo Molchanov, Mostofa Patwary, and 3 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, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 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 6: 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 6, across all rank budgets, initializing the low-rank module from the dense weights consistently yields better performance. These results further support our design choice.

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 7 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.

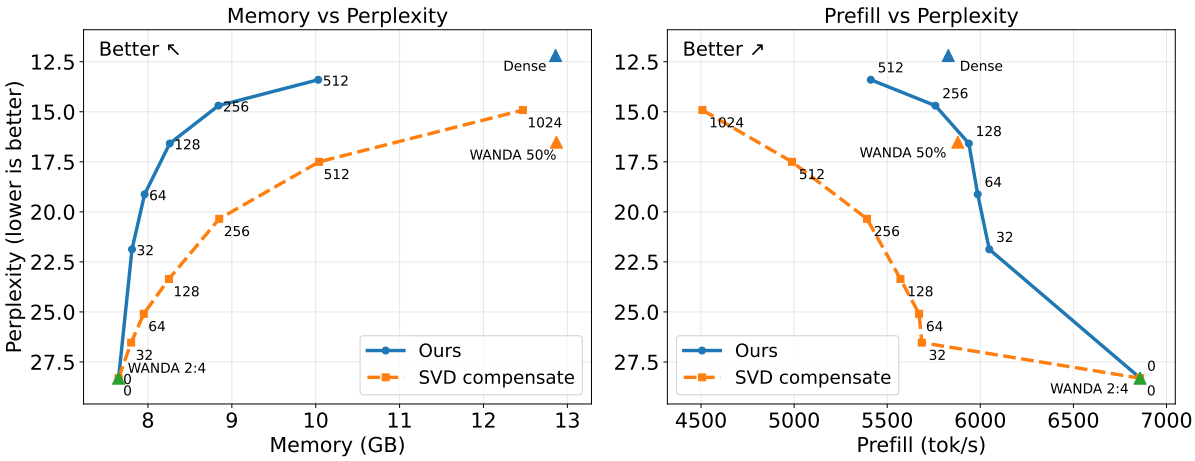

Figure 5: Memory-perplexity-throughput analysis of Residue Pruning under different rank budgets.

Table 7: Preliminary operator-level benchmarks of packing on two branches. Results are profiled from Llama2-7B model on RTX A6000 GPU.

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

A.3 Additional Scaling Efficiency Results

We further evaluate the prefill throughput under different batch sizes and sequence lengths on an NVIDIA A100 GPU. All measurements are conducted on Llama2-7B, and throughput is reported in tokens per second.

As shown in Table 8, our optimized implementation consistently improves prefill throughput over the baseline implementation across different ranks, batch sizes, and sequence lengths. The improvement is especially clear at smaller batch sizes, where reducing kernel launch overhead and overlapping the low-rank branch with the structured-sparse branch is more beneficial.

A.4 Additional Efficiency Results on RTX A6000

We also report the corresponding efficiency results on an NVIDIA A6000 GPU. All measurements are conducted on Llama2-7B, and throughput is reported in tokens per second.

Table 9 first summarizes the A6000 throughput–perplexity–memory trade-off among dense, 2:4 structured pruning, and 50% unstructured pruning. The results confirm our motivation that 2:4 pruning provides clear memory and throughput benefits but incurs a large degradation in perplexity, whereas unstructured pruning preserves better perplexity but offers little practical system benefit.

Furthermore, as shown in Table 10, in the sparse-plus-low-rank setting, our optimized implementation consistently improves prefill throughput over the baselines for both rank budgets, while keeping memory usage nearly unchanged.

Together, these results show that our method improves sparse-plus-low-rank execution efficiency on A6000 and further support the goal of recovering 2:4 pruning quality while preserving its system benefits.

A.5 Additional Efficiency Results on Qwen3-8B

We further report efficiency results on Qwen3-8B using an NVIDIA A100 GPU. As shown in Table 11, under the 2:4 sparsity with rank-32 setting, our optimized implementation improves prefill throughput from 5466.6 to 6323.3 token/s, achieving a 15.7% speedup over the baseline while keeping memory usage unchanged.

For Qwen3-8B, which uses GQA with more irregular attention shapes, enabling streams in the attention path can introduce extra latency due to smaller matrices and additional stream overhead. Therefore, in our final implementation, we only apply stream parallelism to the MLP part.

Together, these results show that our optimized implementation improves Qwen3-8B prefill efficiency on A100 while maintaining the same memory footprint.

Table 8: Prefill throughput comparison under different batch sizes and sequence lengths on Llama2-7B using an A100 GPU. All values are reported in token/s.

Batch	Method	Rank $r = 32$				Rank $r = 128$			
		length=256	**length=512**	**length=1024**	**length=2048**	**length=256**	**length=512**	**length=1024**	**length=2048**
$bs = 1$	Other	4117.3	6979.6	9816.7	10089.1	4183.2	6828.8	9588.8	10059.8
	Ours	**4472.6**	**8309.1**	**10575.4**	**10666.4**	**4386.2**	**8041.2**	**9758.7**	**10618.4**
$bs = 2$	Other	7076.9	9952.3	8729.3	9258.3	6874.3	7962.2	8943.1	9546.8
	Ours	**8227.7**	**10641.3**	**10590.9**	**10622.1**	**8015.6**	**10608.3**	**10792.9**	**10440.4**
$bs = 4$	Other	10032.4	9984.8	10040.6	10122.7	7317.0	9361.9	9511.9	9735.9
	Ours	**10721.8**	**10769.9**	**10775.9**	**10423.6**	**10683.9**	**10535.5**	**10593.0**	**10278.4**

Table 9: Throughput–perplexity–memory trade-off of current unstructured pruning methods (marked as unstr.) and structured pruning methods (marked as str.) on Llama2-7B using an A6000 GPU.

Method	Prefill (token/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

Table 10: Comparison of different pruning methods on prefill throughput and memory usage on A6000. Throughput and memory are evaluated on Llama2-7B.

Method	Sparsity	Throughput (token/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 11: Comparison of the baseline and our optimized implementation on Qwen3-8B prefill throughput and memory usage on A100 under 2:4 sparsity with rank 32.

Method	Throughput (token/s) ↑	Memory (GB) ↓
Baseline	5466.6	10.00
Ours	6323.3	10.00

Table 12: Comparison of different pruning methods applied to Llama2-7B on MMLU evaluation.

Method	Sparsity	MMLU Acc. (%) ↑
Dense	0	45.84
Wanda	0.5	34.00
Wanda 2:4	2:4	27.49
One-shot SVD	2:4 + r128	29.63
TLR	2:4 + r128	32.10
OATS	2:4 + r128	34.76
Ours	2:4 + r128	**35.43**

A.6 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 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.7 MMLU results for Residue Pruning

In addition to text generation perplexity and zero-shot reasoning CMQA tasks, we add MMLU 5-shot accuracy as an extended evaluation benchmark. The additional experiment is conducted on Llama2-7B under the same settings as our main experiments via the lm-eval-harness evaluation pipeline.

These results further support the robustness of Residue Pruning. As shown in Table 12, under the same $2{:}4 + r128$ budget, Residue Pruning achieves the highest MMLU accuracy among all compressed models, including the strong OATS baseline.

Table 13: Comparison of different pruning methods applied to Llama2-13B on WikiText-2 perplexity and zero-shot CMQA accuracy.

Method	Sparsity	WikiText-2 PPL ↓	Zero-shot CMQA Acc. (%) ↑
Dense	0	10.9822	69.26
Wanda	0.5	14.2626	67.15
Wanda 2:4	2:4	21.7239	59.94
One-shot SVD	2:4 + r32	20.7275	60.43
TLR	2:4 + r32	20.1963	60.86
Ours	2:4 + r32	**17.8548**	**62.29**

Table 14: Comparison of different pruning methods on prefill throughput and memory usage on A6000. Throughput and memory are evaluated on Llama2-13B.

Method	Sparsity	Throughput (token/s) ↑	Memory (GB) ↓
Baseline	2:4 + r32	3222.7	14.71
Ours	2:4 + r32	**3472.8**	14.72

A.8 Detailed zero-shot CMQA accuracy

Task-wise zero-shot reasoning accuracy. To complement the averaged zero-shot reasoning results in Table 3, we further report task-wise zero-shot accuracy on the seven evaluation tasks in Table 15, 16, 17, 18. 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.9 Additional Results on Llama2-13B

To further examine the scalability of Residue Pruning, we conduct additional experiments on Llama2-13B under the same post-training pruning setting. We use a low-rank budget of $r = 32$, and evaluate both model quality and system efficiency. For model quality, we report WikiText-2 perplexity and zero-shot CMQA accuracy. For system efficiency, we report prefill throughput and memory usage measured on A6000.

The results show that the advantage of Residue Pruning persists at the 13B scale. As shown in Table 13, under the same $2{:}4 + r32$ budget, our method achieves the best model quality among the evaluated recovery baselines. Compared with TLR, Residue Pruning reduces WikiText-2 perplexity from 20.1963 to 17.8548 and improves zero-shot CMQA accuracy from 60.86% to 62.29%, demonstrating stronger quality recovery under a limited rank budget.

The system-efficiency results in Table 14 further show that our optimization remains effective at the 13B scale. Compared with the sparse-plus-low-rank baseline execution, our method improves prefill throughput from 3222.7 token/s to 3472.8 token/s, achieving a 7.8% speedup with nearly unchanged memory usage. This indicates that our system design improves serving efficiency while maintaining a similar memory footprint.

A.10 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 15: 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 16: 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 17: 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 18: 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