LAYER-WISE SENSITIVITY-AWARE SPARSITY ALLOCATION FOR EFFICIENT LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Model (LLM) inference presents substantial computational challenges when executed on commodity hardware, thereby necessitating the development of efficient acceleration techniques. While existing approaches predominantly focus on uniform compression strategies, they neglect the heterogeneous sensitivity patterns exhibited across different transformer layers. In this paper, we introduce the Adaptive Sparsity Allocation Framework (ASAF), which integrates rotation-based low-bit quantization with layer-wise adaptive sparsity allocation. The framework comprises two sequential phases with a dynamic programming strategy. In Phase 1, coarse-grained optimization determines the optimal number of layer groups and narrows sparsity rate search intervals. In Phase 2, fine-grained optimization determines precise consecutive layer allocation and exact sparsity rates within each group. The joint optimization of layer grouping decisions and sparsity rate assignments creates a combinatorial explosion in the solution space, rendering brute-force approaches computationally prohibitive; we employ a dynamic programming strategy that decomposes the exponential search space into manageable subproblems across both phases, achieving practical computational efficiency while guaranteeing optimality within the discretized two-phase search space. Experiments on the Llama-2 family, with additional memory and zero-shot results on Llama-3, reveal that our proposed framework sustains average benchmark accuracy degradation within 1%, concurrently achieving up to 3.63× prefill acceleration over an FP16 baseline and 12.63% memory reduction over the quantized baseline, on a single NVIDIA RTX 3090 GPU.

1 INTRODUCTION

LLMs underpin contemporary search, dialogue, and recommendation engines, driving a wide spectrum of user-facing AI services (Park et al., 2025). However, the computational demands of LLM inference impose substantial constraints on memory throughput and processing capacity of standard hardware accelerators (Frantar & Alistarh, 2023). A single inference pass alone can exhaust available GPU bandwidth, making each subsequent token generation a performance-critical bottleneck (Alizadeh et al., 2024). The sequential token-by-token generation inherent to autoregressive LLMs, where each prediction relies on the preceding context, establishes dependencies that inhibit parallel processing and amplify computational challenges (Korthikanti et al., 2023).

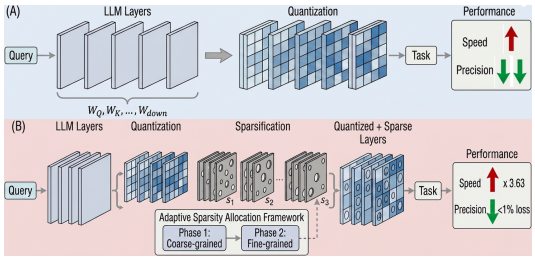


Figure 1: **Top:** quantizing every layer raises speed but lowers precision. **Bottom:** ASAF allocates sparsity to layer groups of the quantized model: 3.63× single-block prefill over FP16 (QuaRot: 3.36×), within 1% of QuaRot's accuracy.

Recent research on efficient LLM inference has converged on two main directions: *i)* This direction focuses on low-bit quantization, which compresses activations and weights through orthogonal rotations and uniform low-bit encodings while preserving model accuracy (Dettmers et al., 2022; Ashkboos et al., 2024c; Xiao et al., 2023). These quantization approaches typically achieve substantial memory reduction and computational speedup, but often struggle with activation outliers that can significantly degrade model quality (Wei et al., 2022); *ii)* network sparsification through weight

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

elimination, removing parameters to decrease computational workload (Han et al., 2015; Mishra et al., 2021). Although sparsification methods can attain high compression ratios with negligible performance degradation, sparsity is most often applied at a single rate across all network layers, and the schemes that do vary it per layer derive the rates from heuristic proxies on full-precision models rather than under an explicit accuracy-degradation budget, leaving per-layer sensitivity only partially exploited (Yin et al., 2024; Li et al., 2024). Nevertheless, these two approaches are fundamentally orthogonal, and current deployments usually have to choose one paradigm at the expense of the other (Sun et al., 2024b), leaving the combined gains untapped.

In this paper, we propose ASAF that unifies rotation-based low-bit quantization and adaptive sparsity allocation through a two-phase optimization framework, as shown in Figure 1. Phase 1 performs coarse-grained optimization to determine optimal layer group numbers and sparsity intervals, while Phase 2 conducts fine-grained optimization for precise layer allocation and exact sparsity rates. The joint optimization creates combinatorial explosion in the solution space, which we address through dynamic programming that efficiently decomposes the exponential search space across both phases while guaranteeing optimality within the discretized two-phase search space. We build our approach on the QuaRot low-bit kernels and realize the allocated sparsity by reducing the GEMM dimensions of each layer group, so that the baseline and ASAF share an identical kernel stack (NVIDIA, 2023; Dao et al., 2022). ASAF addresses key limitations of existing approaches: rotation-based quantization mitigates activation outliers that plague low-bit methods, while adaptive layer-wise sparsity allocation ensures optimal compression capacity utilization unlike uniform pruning approaches. The main contributions are as follows:

- We identify an under-explored gap: sparsity is allocated either uniformly or by heuristic proxies, and never under an explicit accuracy-degradation budget that accounts for the heterogeneous sensitivity of transformer layers.
- We propose a framework that formulates layer-wise sparsity allocation as a constrained optimization problem with consecutive layer grouping constraints. We develop a two-phase approach that employs dynamic programming across both phases to efficiently decompose the exponential search space into manageable subproblems, achieving efficient optimization while guaranteeing optimality within the discretized two-phase search space.
- Our framework achieves up to 3.63× prefill speed-up and a 12.63% memory reduction on Llama-2-70B, with less than 1% average degradation on standard language-understanding benchmarks, advancing the frontier of adaptive LLM compression on commodity GPUs.

2 MOTIVATION

In our explorations, we conduct a series of attempts to investigate the fusion of both techniques. Figure 2**(Left)** demonstrates the inference acceleration achieved on the Llama-2-7B model in a language generation task, after applying an aggressive 20% sparsity rate to various layers and layer combinations (weight matrices at 8-bit precision). We observe a significant improvement in model inference speed. Figure 2**(Right)** shows the model's performance across different quantization and sparsification configurations. As the bit-width of quantization decreases and the sparsity ratio increases, the perplexity rises substantially, indicating a clear trade-off between efficiency and performance. This observation gives rise to two key challenges:

Precision-Compression Integration. The introduction of quantization artifacts fundamentally reshapes weight significance patterns, rendering traditional sparsification strategies ineffective since they are originally developed for full-precision models (Dettmers et al., 2024). Simultaneously, sparsification operations modify activation characteristics, which subsequently disrupts optimal quantization parameter selection (Guo et al., 2024; Yao et al., 2022). The two mechanisms must therefore be optimized jointly (Guo et al., 2025), the premise of our formulation.

Solution Space Explosion. The intersection of quantization and sparsification transforms the combinatorial optimization problem into a multidimensional challenge involving layer grouping decisions, sparsity rate assignments, and quantization interactions. This exponential growth in combinatorial possibilities renders brute-force search strategies computationally prohibitive (Choi & Cho, 2025; Dettmers et al., 2022; Frantar & Alistarh, 2023). Contemporary approaches suffer from evaluation overhead, demanding multiple fine-tuning cycles for performance assessment of each candidate configuration (Liu et al., 2025a; Ma et al., 2023; Mozafarinia et al., 2024). The critical need

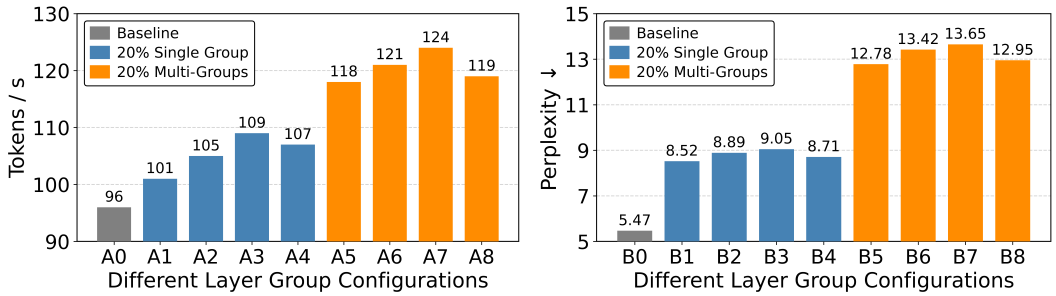


Figure 2: Llama-2-7B with 20% sparsity applied to different layer groups. **Left:** decode-stage throughput. **Right:** perplexity. More details are provided in Appendix A.2.

emerges for an efficient algorithm capable of identifying near-optimal layer grouping and sparsity allocation strategies without retraining each candidate configuration (Zhang et al., 2022).

3 Preliminaries and Problem Formulation

Preliminaries. We build on rotation-based INT4 quantization (Ashkboos et al., 2024c) and classical dynamic programming (Bellman, 1957); self-contained background on both, including the full QuaRot pipeline (rotation, RMSNorm fusion, Hadamard transform) and the DP optimal-substructure and overlapping-subproblems properties used directly by our framework, is provided in Appendices A.4 and A.5. ASAF keeps the quantizer fixed and allocates sparsity on the 4-bit model.

Problem Formulation. In order to solve the challenges mentioned in Section 2, we formulate them as a constrained optimization problem. The primary objective is to find the optimal layer grouping and sparsity allocation strategy that minimizes total computational FLOPs while ensuring accuracy degradation remains within acceptable bounds. The mathematical formulation is defined as follows:

$$\{G^*, \{\mathcal{L}_i^*\}, \{s_i^*\}\} = \arg \min_{G, \{\mathcal{L}_i\}, \{s_i\}} \sum_{i=1}^{G} \sum_{l \in \mathcal{L}_i} \phi_l \times (1 - s_i), \tag{1}$$

where G denotes the number of layer groups, $\mathcal{L}_i$ represents the set of consecutive layers in group i, s_i is the sparsity rate applied to group i, and ϕ_l represents the original computational cost (in FLOPs) of layer l. This optimization problem is subject to the following constraints:

$$\begin{aligned} &\text{(i) accuracy: } \textstyle\sum_{i=1}^{G} \xi(\mathcal{L}_i, s_i) \le \delta_{\max}, \\ &\text{(ii) sparsity: } \alpha \le s_i \le \beta, \ \forall i, \\ &\text{(iii) completeness: } \textstyle\bigcup_{i=1}^{G} \mathcal{L}_i = \{1, \ldots, p\}, \ \mathcal{L}_i \cap \mathcal{L}_j = \emptyset, \\ &\text{(iv) continuity: } \mathcal{L}_i = \{l_{\text{start}}^{(i)}, \ldots, l_{\text{end}}^{(i)}\} \end{aligned} \tag{2}$$

where $\xi(\mathcal{L}_i, s_i)$ is the sensitivity of layer group $\mathcal{L}_i$ to sparsity rate s_i, $\delta_{\max}$ represents the maximum allowable accuracy degradation threshold, α and β define the lower and upper bounds of the feasible sparsity range, $l_{\text{start}}^{(i)}$ and $l_{\text{end}}^{(i)}$ denote the starting and ending layer indices of group i, p denotes the total number of layers in the model, and the continuity constraint ensures consecutive sequences for efficient hardware batch processing, improved memory locality of adjacent layers, and significantly reduced combinatorial search space (Mishra et al., 2021). The sensitivity ξ is measured directly, as the perplexity increase of the sparsified model over the quantized baseline on a fixed 640-sample subset of WikiText-2, rather than through a proxy score (Appendix A.10).

4 Solution

We propose ASAF, which decomposes Eq. 1 into two sequential phases: a *coarse-grained* phase that fixes the optimal group count G^* and narrows each group's sparsity interval via layer sensitivity, followed by a *fine-grained* phase that pins down the exact consecutive-layer allocation $\{\mathcal{L}_i^*\}$ and sparsity rates $\{s_i^*\}$ within those intervals. Both phases are solved by dynamic programming: candidate groupings share overlapping substructures, so memoized tabulation reuses partial FLOP–accuracy costs and returns an optimal solution within the discretized two-phase search space under the original objective and degradation budget.

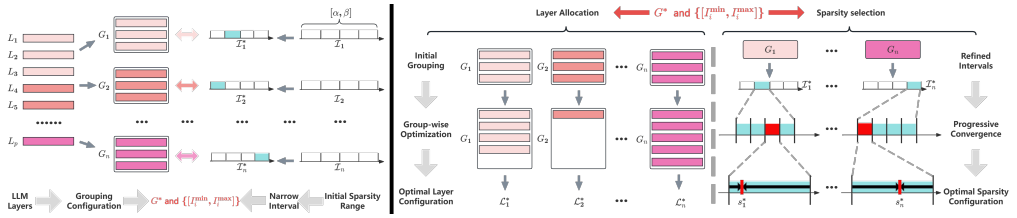


Figure 3: Overview of ASAF's two-phase optimization. **Left:** coarse-grained. **Right:** fine-grained.

Algorithm 1 Coarse-grained Optimization

Require: Layer count p, sparsity range $[\alpha, \beta]$, accuracy degradation threshold $\delta_{\max}$
Ensure: Optimal group number G^* and intervals $\{[I_i^{\min}, I_i^{\max}]\}$

Initialize $\mathrm{DP}_{\text{coarse}}[0..p][0..\delta_{\max}/\Delta] \leftarrow \infty$, set boundary condition: $\mathrm{DP}_{\text{coarse}}[0][0] \leftarrow 0$
Generate interval candidates $\mathcal{I} = \{I : I \subseteq [\alpha, \beta]\}$
Main DP Loop:
for $g = 1$ **to** p **do**
  for $b = 0$ **to** $\delta_{\max}/\Delta$ **do**
    for each interval $I \in \mathcal{I}$ **do**
      cost $\leftarrow$ OptimalCost(I) using tabulation H
      accuracy_cost $\leftarrow \Xi_{\min}(I)/\Delta$ using tabulation Ξ
      if $b -$ accuracy_cost ≥ 0 **then**
        total_cost $\leftarrow$ cost $+ \mathrm{DP}_{\text{coarse}}[g-1][b - \text{accuracy_cost}]$
        $\mathrm{DP}_{\text{coarse}}[g][b] \leftarrow \min(\mathrm{DP}_{\text{coarse}}[g][b], \text{total_cost})$
      end if
end for (all nested loops)
Solution Extraction:
$G^* \leftarrow \arg\min_g \{\min_b \mathrm{DP}_{\text{coarse}}[g][b]\}$
$\{I_i^{\min}, I_i^{\max}\} \leftarrow$ BacktrackOptimalIntervals$(\mathrm{DP}_{\text{coarse}}, G^*)$
return G^*, $\{[I_i^{\min}, I_i^{\max}]\}_{i=1}^{G^*}$

Coarse-grained Optimization. This phase jointly determines the optimal number of groups G^* and refined sparsity intervals $\{[I_i^{\min}, I_i^{\max}]\}_{i=1}^{G^*}$ by evaluating all feasible group configurations and narrowing the search space $[\alpha, \beta]$ to focused sub-ranges for each group, as shown in Figure 3 (**Left**). Formally, this phase solves:

$$\{G^*, \{I_i^*\}_{i=1}^{G^*}\} = \arg \min_{G, \{I_i\}_{i=1}^{G}} \left\{ \sum_{i=1}^{G} \min_{\mathcal{L}_i, s_i \in I_i} \sum_{l \in \mathcal{L}_i} \phi_l \times (1 - s_i) \right\}, \tag{3}$$

where I_i denotes the sparsity interval for group i; the remaining symbols are as in Eq. 1, and accuracy constraint (i) of Eq. 2 still applies. Sparsity rates are discretized as $S_{\text{discrete}} = \{\alpha + k\Delta \leq \beta : k \in \mathbb{N}\}$ with resolution Δ, so the candidate intervals I range over the finite set of sub-ranges with endpoints on that grid rather than over all subsets of $[\alpha, \beta]$.

We define the dynamic programming state $\mathrm{DP}_{\text{coarse}}[g][b]$ as the minimum total FLOP cost when using exactly g groups with discretized accuracy degradation budget b:

$$\mathrm{DP}_{\text{coarse}}[g][b] = \min_{\{I_i\}_{i=1}^{g}} \left\{ \sum_{i=1}^{g} \min_{\mathcal{L}_i, s_i \in I_i} \sum_{l \in \mathcal{L}_i} \phi_l \times (1 - s_i) \right\}, \tag{4}$$

where the state represents the optimal cost achievable with g groups under accuracy degradation budget $b \times \Delta$ (with Δ being the discretization step), and the minimization considers all valid interval partitions and corresponding layer-sparsity assignments.

For each state, we consider all ways to add one more group by selecting an interval subset:

$$\mathrm{DP}_{\text{coarse}}[g][b] = \min_{I \subseteq [\alpha, \beta]} \left\{ \text{OptimalCost}(I) + \mathrm{DP}_{\text{coarse}}[g-1][b - \xi_{\text{cost}}(I)/\Delta] \right\}, \tag{5}$$

where OptimalCost(I) and $\xi_{\text{cost}}(I)$ are the minimum FLOP cost and the accuracy degradation attainable within interval I.

Algorithm 2 Fine-grained Optimization

Require: G^* groups, intervals $\{[I_i^{\min}, I_i^{\max}]\}$, tabulation tables H, Ξ
Ensure: Optimal allocation $\{\mathcal{L}_i^*\}$ and sparsity rates $\{s_i^*\}$
Initialize $\text{DP}_{\text{fine}}[1..p+1][0..G^*][0..\delta_{\max}/\Delta] \leftarrow \infty$, $\text{choice}[1..p+1][0..G^*][0..\delta_{\max}/\Delta] \leftarrow \emptyset$
Set boundary condition: $\text{DP}_{\text{fine}}[p+1][0][b] \leftarrow 0$ for all $b \geq 0$
Backward DP Construction:
for $i = p$ **down to** 1 **do**
  for $g = 1$ **to** $\min(G^*, p - i + 1)$ **do**
    for $b = 0$ **to** $\delta_{\max}/\Delta$ **do**
      for $j = i$ **to** $p - g + 1$, $s \in \text{Discretize}([I_{G^*-g+1}^{\min}, I_{G^*-g+1}^{\max}])$ **do**
      group_cost, accuracy_cost $\leftarrow H[i][j-i+1][s], \Xi[i][j-i+1][s]/\Delta$
      if $b -$ accuracy_cost ≥ 0 **then**
        total_cost $\leftarrow$ group_cost $+ \text{DP}_{\text{fine}}[j+1][g-1][b - \text{accuracy_cost}]$
        if total_cost $< \text{DP}_{\text{fine}}[i][g][b]$ **then**
          $\text{DP}_{\text{fine}}[i][g][b] \leftarrow$ total_cost, $\text{choice}[i][g][b] \leftarrow (j, s)$
      end if (nested conditions)
end for (all nested loops)
Solution Reconstruction:
$\{\mathcal{L}_i^*, s_i^*\} \leftarrow \text{BacktrackSolution}(\text{choice}, 1, G^*, \delta_{\max}/\Delta)$
return $\{\mathcal{L}_i^*\}_{i=1}^{G^*}, \{s_i^*\}_{i=1}^{G^*}$

Algorithm 1 solves this recurrence by sweeping the group count $G \in [1, p]$ via tabulated DP, contracting the original search space $[\alpha, \beta]$ for the fine-grained phase.

Rationale for Layer Grouping. Consecutive grouping in ASAF reflects a Transformer regularity: adjacent blocks share hidden dimension, attention/MLP structure and residual pathway, yielding a smoothly varying depth-wise sparsity sensitivity rather than abrupt per-layer jumps (Men et al., 2025; Ma et al., 2023; Lu et al., 2024). ASAF leverages this smoothness to shrink the search space while the fine-grained stage retains intra-group flexibility. More details are provided in Appendix A.3.

Fine-grained Optimization. This phase receives the optimal group number G^* and refined intervals $\{[I_i^{\min}, I_i^{\max}]\}_{i=1}^{G^*}$ from the coarse-grained phase, then jointly determines the exact consecutive layer allocation $\{\mathcal{L}_i^*\}_{i=1}^{G^*}$ and precise sparsity rates $\{s_i^*\}_{i=1}^{G^*}$, as shown in Figure 3 (**Right**).

This phase therefore solves:

$$\{\{\mathcal{L}_i^*\}_{i=1}^{G^*}, \{s_i^*\}_{i=1}^{G^*}\} = \arg \min_{\{\mathcal{L}_i\}_{i=1}^{G^*}, \{s_i\}_{i=1}^{G^*}} \left\{ \sum_{i=1}^{G^*} \sum_{l \in \mathcal{L}_i} \phi_l \times (1 - s_i) \right\}, \tag{6}$$

subject to constraints (i)–(iv) of Eq. 2, with the sparsity bound tightened to $s_i \in [I_i^{\min}, I_i^{\max}]$.

We define the dynamic programming state $\text{DP}_{\text{fine}}[i][g][b]$ as the minimum FLOP cost for partitioning layers $[i..p]$ into g consecutive groups under accuracy budget b:

$$\text{DP}_{\text{fine}}[i][g][b] = \min_{\{\mathcal{L}_k\}_{k=1}^{g}, \{s_k\}_{k=1}^{g}} \left\{ \sum_{k=1}^{g} \sum_{l \in \mathcal{L}_k} \phi_l \times (1 - s_k) \right\}, \tag{7}$$

The minimization is subject to coverage $\bigcup_{k=1}^{g} \mathcal{L}_k = \{i, \ldots, p\}$ with each $\mathcal{L}_k$ consecutive, sparsity bounds $s_k \in [I_k^{\min}, I_k^{\max}]$, and budget $\sum_{k=1}^{g} \xi(\mathcal{L}_k, s_k) \leq b$.

For each state, we jointly enumerate all possible first-group formations and sparsity assignments within the corresponding refined interval. The state transition equation is:

$$\text{DP}_{\text{fine}}[i][g][b] = \min_{\substack{j \in [i, p-g+1] \\ s \in [I_g^{\min}, I_g^{\max}]}} \left\{ \text{GroupCost}(i, j, s) + \text{RemainingCost}(j+1, g-1, b') \right\}, \tag{8}$$

where the total cost consists of two components. (1) The current group cost $\text{GroupCost}(i, j, s) = H[i][j-i+1][s]$ represents the FLOP cost for assigning layers i to j with sparsity rate s, retrieved

Table 1: WikiText-2 perplexity for Llama-2 at 2048 tokens under 4-bit quantization with adaptive sparsity allocation. SmoothQuant and OmniQuant numbers are from (Shao et al., 2024); 128G denotes group-wise quantization with group size 128. More details are provided in Appendix A.12.

Method	Weight Quant.	#Outlier Features	Llama-2 7B	Llama-2 13B	Llama-2 70B
Baseline	-	-	5.47	4.88	3.32
SmoothQuant	RTN	0	83.12	35.88	-
OmniQuant	RTN	0	14.26	12.30	-
QUIK-4B	GPTQ	256	8.87	7.78	6.91
QuaRot	GPTQ	0	**6.10**	**5.40**	**3.79**
ASAF (Ours)	GPTQ	0	6.14	5.44	3.82
Atom-128G	GPTQ-128G	128	6.03	**5.26**	-
QuaRot-128G	GPTQ-128G	0	**5.93**	**5.26**	**3.61**
ASAF-128G (Ours)	GPTQ-128G	0	5.98	5.30	3.64

Table 2: Zero-shot accuracy on PIQA (PQ), WinoGrande (WG), HellaSwag (HS), Arc-Easy (A-e), Arc-Challenge (A-c) and LAMBADA (LA); higher is better. Average degradation stays below 1% versus QuaRot (see Appendix A.12).

Method	PQ	WG	HS	A-e	A-c	LA	Avg.
Llama-2-7B							
FP16	79.11	69.06	75.99	74.58	46.25	73.90	69.82
QuaRot	76.77	63.77	72.16	69.87	40.87	70.39	65.64
ASAF (Ours)	76.00	63.23	71.47	69.17	40.52	69.69	65.01
Llama-2-13B							
FP16	80.47	72.22	79.39	77.48	49.23	76.75	72.59
QuaRot	78.89	70.24	76.37	72.98	46.59	73.67	69.79
ASAF (Ours)	78.26	69.68	75.76	72.25	46.19	72.97	69.19
Llama-2-70B							
FP16	82.70	77.98	83.84	80.98	57.34	79.58	77.07
QuaRot	82.43	76.24	81.82	80.43	56.23	78.73	75.98
ASAF (Ours)	81.77	75.48	81.12	79.71	55.81	77.98	75.31

from pre-computed tabulation. (2) The remaining subproblem cost $\text{RemainingCost}(j{+}1, g{-}1, b') = \text{DP}_{\text{fine}}[j+1][g-1][b']$ represents the optimal cost for the remaining layers $[j+1, p]$ using $g-1$ groups with updated budget $b' = b - \lceil \Xi[i][j-i+1][s]/\Delta \rceil$, where $\Xi[i][j-i+1][s]$ gives the accuracy degradation cost from tabulation.

Algorithm 2 solves this recurrence by backward DP from the final layer toward the first, with each layer-boundary decision strictly honoring the refined sparsity intervals from the coarse phase, completing the two-phase ASAF solution.

Tabulation. ASAF precomputes two tables, H and Ξ, storing FLOP cost and accuracy degradation values for every consecutive layer interval at each discretized sparsity rate. These tables reduce each DP transition to an $O(1)$ lookup, making the entire two-phase search tractable. Full definitions, construction algorithm, complexity analysis, and storage analysis are provided in Appendix A.10.

5 Experiment

5.1 Implementation Details

Software and Hardware Setup. Our ASAF framework is built upon QuaRot (Ashkboos et al., 2024c) using PyTorch (Paszke et al., 2019) with CUDA-12.1 (NVIDIA, 2023), evaluated on Llama-2 and Llama-3 family models (Touvron et al., 2023). The dynamic programming tabulation and layer-wise accuracy degradation measurement are implemented in PyTorch, with all evaluations on NVIDIA RTX 3090 GPUs to facilitate reproducibility across academic research labs.

Framework Hyperparameters. We explore sparsity allocations where $\alpha = 1\%$ and $\beta = 15\%$ define the feasible sparsity range across all evaluated models. The maximum allowable accuracy degradation is set to $\delta_{\max} = 1\%$ with tabulation resolution $\Delta = 0.5\%$, chosen to balance the DP table size against allocation granularity, and together they yield 29 discrete sparsity levels. The accuracy budget binds first. More details are provided in Appendix A.17.

Experimental Configuration. We employ 4-bit GPTQ quantization (Frantar et al., 2022) for weights with group size 128, symmetric per-token quantization for activations (Xiao et al., 2023), and asymmetric quantization for KV caches (Dettmers et al., 2022). All configurations utilize clipping ratios of 0.9 and 0.95 and maintain numerical stability through FP32 accumulation during tabulation construction, matching the W4A4KV4 protocol of recent quantization baselines. More details are provided in Appendix A.11.

Precomputation Overhead. ASAF's preprocessing builds the tables H, Ξ and runs a DP search, requiring no additional training. The tabulation runs on a single consumer-grade 24GB RTX 3090 GPU, and this one-time, model-level cost amortizes across every subsequent deployment (full complexity analysis is provided in Appendix A.18).

Table 3: End-to-end throughput and latency on RTX 3090. The second column gives prompt length, generation length and batch size; higher tokens/s (↑) and lower latency (↓) are better. More details are provided in Appendix A.22.

Model	Prompt / Gen. / Batch	Method	Tokens/s (↑)	Latency (ms) (↓)
Llama-2-7B	512 / 128 / 1	QuaRot	909	711
		ASAF	920	703
	2048 / 512 / 1	QuaRot	703	3657
		ASAF	720	3570
Llama-2-13B	2048 / 512 / 8	QuaRot	2611	7877
		ASAF	2687	7654
	4096 / 512 / 8	QuaRot	2129	17554
		ASAF	2203	16961

Table 4: GPU memory (MB) of QuaRot and ASAF. More details are provided in Appendix A.12.

Model	Method	Sequence Length 512	2048	4096
Llama-2-7B	QuaRot	1,019	3,255	6,518
	ASAF	948	3,013	6,009
	Comp. Ratio	6.97%	7.43%	7.81%
Llama-2-70B	QuaRot	6,521	20,536	41,079
	ASAF	5,653	17,943	35,833
	Comp. Ratio	13.31%	12.63%	12.77%
Llama-3-8B	QuaRot	1,226	3,722	7,456
	ASAF	1,127	3,432	6,869
	Comp. Ratio	8.08%	7.79%	7.87%
Llama-3-70B	QuaRot	6,811	21,295	42,586
	ASAF	5,855	18,136	36,241
	Comp. Ratio	14.04%	14.83%	14.90%

5.2 Accuracy Analysis

Language Generation Tasks. We evaluate our ASAF framework on the WikiText-2 language-generation benchmark. Table 1 reports the perplexity after quantizing Llama-2 weights to 4 bits with GPTQ and applying our adaptive sparsity allocation across layer groups. Our framework demonstrates competitive performance compared to recent advanced quantization methods, achieving perplexity degradation less than 1% compared to QuaRot while providing additional computational benefits through optimized sparsity allocation. The layer-group-based pruning approach requires no additional outlier storage and no asymmetric weight quantization. The same holds under group-size-128 quantization.

Zero-Shot Tasks. We assess ASAF across six established zero-shot benchmarks: PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021), HellaSwag (Zellers et al., 2019), LAMBADA (Radford et al., 2019), and ARC-Easy and ARC-Challenge (Clark et al., 2018). Experiments utilize the LM Evaluation Harness (Gao et al., 2021; 2024) with default configurations. Table 2 shows that ASAF maintains strong performance across all Llama-2 model sizes, with average degradation consistently below 1% compared to QuaRot.

5.3 Performance Analysis

Prefill Stage Performance Increases. Figure 4 reports prefill-stage acceleration at batch sizes 1, 4, 16 and 32 with 2048-token sequences on a single transformer block of Llama-2 models. Our adaptive sparsity allocation consistently outperforms the QuaRot baseline across all configurations. The performance gains become more pronounced with larger batch sizes, as the computational workload increasingly overshadows memory bandwidth limitations. For the largest 70B model, our method reaches a peak acceleration of 3.63× over the FP16 baseline, against 3.36× for QuaRot under the same measurement, an 8% gain. The results reveal a trend where both increasing model complexity and batch size magnify the effectiveness of our strategy, demonstrating the scalable nature of the ASAF framework's adaptive sparsity allocation mechanism.

End-to-end Latency and Throughput. We measure full-sequence inference performance including both the prompt processing and autoregressive decoding phases to complement the prefill-focused results. We evaluate Llama-2-7B and Llama-2-13B models on an RTX 3090 GPU under several realistic configurations combining prompt lengths of 512, 2048, and 4096 tokens, generation lengths of 128 and 512 tokens, and batch sizes of 1 and 8. Table 3 reports the end-to-end tokens-per-second and wall-clock latency for the QuaRot baseline and our ASAF-optimized models. Across all tested settings, ASAF improves end-to-end throughput while respecting the same $\leq$1% accuracy budget used throughout the paper, delivering gains of 1.2% and 2.4% on Llama-2-7B and 2.9% and 3.5% on Llama-2-13B. The gains grow with prompt length, where prefill dominates the runtime; the allocation leaves the decoding phase untouched.

Memory and Computational Efficiency. The algorithmic efficiency of ASAF directly translates into substantial memory savings across both model families and sequence lengths. Table 4 expands our evaluation beyond the Llama-2 series to include the more recent Llama-3 models and reports

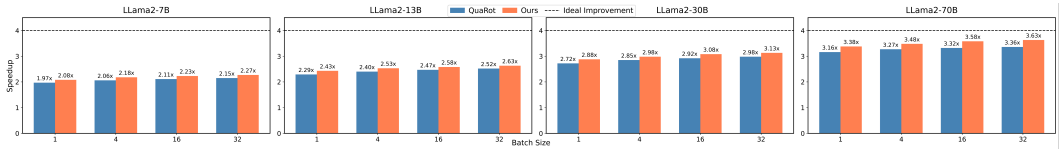


Figure 4: Prefill-stage acceleration of ASAF vs. QuaRot on a single transformer block of Llama-2 models (RTX 3090, 2048-token sequences, batch sizes 1, 4, 16, and 32). Both methods are measured against the FP16 baseline, for which the dashed line marks the 4× ideal of 4-bit execution. ASAF reaches 3.63× on Llama-2-70B, 8% above QuaRot.

Table 5: ASAF vs. recent compression baselines on LLaMA-7B: WikiText-2 perplexity and accuracy on five of the Table 2 benchmarks, BoolQ (Clark et al., 2019) and MMLU (Hendrycks et al., 2021). More details are provided in Appendix A.21.

Method	WikiText-2 PPL ↓	PQ ↑	WG ↑	HS ↑	A-e ↑	A-c ↑	BoolQ ↑	MMLU ↑	Average ↑
SpinQuant (Liu et al., 2025b)	5.94	76.14	69.58	72.16	69.30	40.37	63.77	30.00	60.19
OWL (Yin et al., 2024)	24.55	62.48	58.72	44.79	45.03	26.19	58.48	26.84	46.08
DSA (Li et al., 2024)	12.50	68.13	65.97	61.84	65.26	39.42	66.84	31.05	56.93
SLIM (Mozaffari et al., 2025)	9.80	75.04	67.83	67.35	67.14	41.38	70.23	29.41	59.77
ASAF (Ours)	5.98	76.00	63.23	71.47	69.17	40.52	74.23	31.87	60.93

results under three context lengths: 512, 2048, and 4096 tokens. This broader evaluation shows that ASAF maintains stable compression ratios across architectures and sequence lengths, achieving 7.0 to 14.9% memory reduction depending on model scale, and confirming that its sensitivity-aware sparsity allocation generalizes beyond a single model generation and one architecture family. Additional computational-efficiency results are provided in Appendix A.13.

Comparison with Recent Compression Methods. Table 5 compares ASAF against recent compression approaches on LLaMA-7B. ASAF achieves a competitive 4-bit WikiText-2 perplexity (5.98 vs. SpinQuant's 5.94), with negligible degradation of 0.04 points. More importantly, ASAF attains the highest average accuracy (60.93%) across seven zero-shot benchmarks, outperforming SpinQuant (60.19%), SLIM (59.77%), and every other baseline. In contrast, pure sparsification methods suffer substantial quality degradation, with OWL and DSA exhibiting perplexities of 24.55 and 12.50 respectively, alongside lower task accuracies (46.08% and 56.93%). These results demonstrate that ASAF's layer-wise sensitivity-aware sparsity allocation, when integrated with rotation-based quantization, outperforms both uniform quantization and standalone sparsification.

Layer Sensitivity and Sparsity Pattern Analysis. ASAF discovers a U-shaped depth-wise sensitivity profile on Llama-2-7B. Early and final blocks receive lower sparsity, while mid-depth MLPs tolerate higher sparsity. This pattern aligns with established Transformer redundancy structure and emerges from the search itself, without hand-crafted heuristics. More details are provided in Appendix A.19.

Prefill Stage Performance on NVIDIA 4090 GPU. Appendix A.14 reports prefill acceleration experiments across different models and methods. ASAF achieves up to 3.89× acceleration on Llama-2-70B over the FP16 baseline, further validating hardware scalability across GPU generations.

Zero-Shot Tasks on Llama-3 Family. Appendix A.15 reports zero-shot evaluation across different models and methods. On both Llama-3-8B and Llama-3-70B, ASAF further demonstrates strong generalizability and consistently maintains $<1\%$ average performance degradation versus QuaRot.

5.4 Ablation Study and Mechanism Analysis

Uniform versus Adaptive Allocation. To test the premise of the paper directly, we compare ASAF against a heuristic that assigns a single sparsity level uniformly across all layer groups, without sensitivity information and under the same search budget and evaluation protocol. For every coarse partition the uniform heuristic exceeds the $\leq 1\%$ accuracy budget, whereas ASAF stays within it, losing between 0.58 and 0.74%. More details are provided in Appendix A.16.

Where ASAF Fits. ASAF adds sparsity on top of low-bit execution: it takes a rotation-quantized model as given and decides how much sparsity each group of consecutive layers can absorb under an explicit accuracy-degradation budget. Each candidate group is scored by its perplexity increase on the quantized model, measured once and tabulated, so the search needs no retraining, and a dynamic program over consecutive groups returns the minimum-FLOP allocation on the dis-

cretized grid within the coarse intervals. Sparsity only shrinks GEMM dimensions, so ASAF shares QuaRot's kernels and its gains come from the allocation: single-block prefill speedup over FP16 on Llama-2-70B rises from 3.36× to 3.63× and end-to-end throughput by up to 3.5%, within 1% of QuaRot's accuracy.

RTN Quantization Strategy. To evaluate the robustness of our adaptive sparsity allocation approach, we compare ASAF's performance under RTN quantization, where GPTQ is the default weight quantization strategy. Table 6 demonstrates that at 8-bit precision, RTN maintains accuracy nearly identical to full precision for both approaches. At 4-bit quantization, while both methods experience quality degradation, our ASAF framework maintains average performance within 1% of QuaRot across all model sizes (Appendix A.12). Across INT4 and INT8 configurations, these findings confirm that layer-group-based adaptive sparsity allocation can be combined with RTN quantization without exceeding the accuracy budget, validating the generalizability of our optimization framework under both quantization schemes.

Table 6: WikiText-2 perplexity and average zero-shot accuracy under RTN quantization. More details are provided in Appendix A.12.

Model	Method	Prec.	PPL ↓	Average over Benchmarks ↑
7B	Baseline	FP16	5.47	69.82
	QuaRot-RTN	INT4	8.37	58.26
		INT8	5.50	69.65
	ASAF-RTN (Ours)	INT4	8.44	57.77
		INT8	5.54	69.06
70B	Baseline	FP16	3.32	77.07
	QuaRot-RTN	INT4	4.14	73.63
		INT8	3.33	77.17
	ASAF-RTN (Ours)	INT4	4.18	72.92
		INT8	3.36	76.44

6 Related Work

Compressing Large Language Models. LLM compression runs along four lines. Structured pruning removes whole attention heads or layers, giving predictable savings but little allocation freedom (Dutta et al., 2024; Muralidharan et al., 2024); weight quantization lowers numerical precision (Lin et al., 2024; Liu et al., 2025b; Shao et al., 2024); and distillation transfers capability into smaller students (Gu et al., 2023). Unstructured pruning keeps finer control over the accuracy-efficiency trade-off: SparseGPT (Frantar & Alistarh, 2023) reconstructs layer-wise, Wanda (Sun et al., 2024b) scores by activation-aware magnitude, and OWL (Yin et al., 2024), ALPS (Meng et al., 2024), BESA (Xu et al., 2024) and DSnoT (Zhang et al., 2024) refine that criterion; sparsity has been paired with FP4 activations (Meng et al., 2026) and 1.58-bit weights (Zhang et al., 2026).

Low-Bit Quantization and Joint Compression. Quantization at four bits is limited chiefly by activation outliers that dominate the dynamic range. Rotation-based methods redistribute outlier energy through orthogonal transforms (Ashkboos et al., 2024c; Chee et al., 2023) while SmoothQuant (Xiao et al., 2023) migrates the difficulty into the weights, and serving systems attack the same cost from the runtime side (Dao et al., 2022; Kwon et al., 2023; Yuan et al., 2025). Joint sparse-quantized formats (Dettmers et al., 2024; Guo et al., 2024) and non-uniform layer-wise sparsity (Yin et al., 2024; Li et al., 2024) have each been explored separately.

Inference Systems. Serving systems speed up a fixed model: SpeCache prefetches KV entries (Jie et al., 2025), FlashDecoding++ uses an asynchronous softmax (Hong et al., 2024), cascade speculative drafting drafts with smaller models (Chen et al., 2024), and Sarathi-Serve and Llumnix schedule requests (Agrawal et al., 2024; Sun et al., 2024a). ASAF instead reduces model computation.

7 Conclusion

We present ASAF, which systematically allocates sparsity across layer groups to minimize computational FLOPs while maintaining model accuracy. Our two-phase strategy employs dynamic programming with tabulation to achieve efficient optimization with manageable computational complexity, making large-scale optimization tractable for practical deployment. Experimental results demonstrate that ASAF maintains average accuracy degradation within 1% while achieving up to 3.63× prefill acceleration over an FP16 baseline and 12.63% memory reduction over the quantized baseline on Llama-2 models at a 2048-token context.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The Twelfth International Conference on Learning Representations*, 2024.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, 2024.

Keivan Alizadeh, Seyed Iman Mirzadeh, Dmitry Belenko, S Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. Llm in a flash: Efficient large language model inference with limited memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12562–12584, 2024.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE, 2022.

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. SliceGPT: Compress large language models by deleting rows and columns. In *International Conference on Learning Representations*, 2024a.

Saleh Ashkboos, Ilia Markov, Elias Frantar, Tingxuan Zhong, Xincheng Wang, Jie Ren, Torsten Hoefler, and Dan Alistarh. QUIK: Towards end-to-end 4-bit inference on generative large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024b. doi: 10.18653/v1/2024.emnlp-main.197.

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

Richard Ernest Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Ruisi Cai, Yeonju Ro, Geon-Woo Kim, Peihao Wang, Babak Ehteshami Bejnordi, Aditya Akella, Zhangyang Wang, et al. Read-me: Refactorizing llms as router-decoupled mixture of experts with system co-design. *Advances in Neural Information Processing Systems*, 37:116126–116148, 2024.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36:4396–4429, 2023.

Ciprian Chelba, Mia Chen, Ankur Bapna, and Noam Shazeer. Faster transformer decoding: N-gram masked self-attention. *arXiv preprint arXiv:2001.04589*, 2020.

Mengzhao Chen, Yi Liu, Jiahao Wang, Yi Bin, Wenqi Shao, and Ping Luo. PrefixQuant: Eliminating outliers by prefixed tokens for large language models quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026. doi: 10.1109/tpami.2026.3711802.

Ziyi Chen, Xiaocong Yang, Jiacheng Lin, Chenkai Sun, Kevin Chang, and Jie Huang. Cascade speculative drafting for even faster llm inference. *Advances in Neural Information Processing Systems*, 37:86226–86242, 2024.

Seonggyu Choi and Hyungmin Cho. Sparq: An accelerator architecture for large language models with joint sparsity and quantization techniques. In *Proceedings of the 26th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, pp. 3–15, 2025.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019. doi: 10.18653/v1/n19-1300.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018. URL https://api.semanticscholar.org/CorpusID:3922816.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 3rd edition, 2009. ISBN 9780262033848.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Rocktim Jyoti Das, Mingjie Sun, Liqun Ma, and Zhiqiang Shen. Beyond size: How gradients shape pruning decisions in large language models. arXiv preprint arXiv:2311.04902, 2023.

Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill Higher Education, Boston, MA, 2006.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30318–30332. Curran Associates, Inc., 2022. doi: 10.52202/068431-2198.

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

Oshin Dutta, Ritvik Gupta, and Sumeet Agarwal. Efficient llm pruning with global token-dependency awareness and hardware-adapted inference. In *Workshop on Efficient Systems for Foundation Models II@ ICML2024*, 2024.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Zichuan Fu, Wentao Song, Yejing Wang, Xian Wu, Yefeng Zheng, Yingying Zhang, Derong Xu, Xuetao Wei, Tong Xu, and Xiangyu Zhao. Sliding window attention training for efficient large language models. *arXiv preprint arXiv:2502.18845*, 2025.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. A framework for few-shot language model evaluation. *Zenodo*, 2021.

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

Ruihao Gong, Yifu Ding, Zining Wang, Chengtao Lv, Xingyu Zheng, Jinyang Du, Haotong Qin, Jinyang Guo, Michele Magno, and Xianglong Liu. A survey of low-bit large language models: Basics, systems, and algorithms. *arXiv preprint arXiv:2409.16694*, 2024.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543*, 2023.

Hang Guo, Yawei Li, and Luca Benini. Optimal brain restoration for joint quantization and sparsification of llms. *arXiv preprint arXiv:2509.11177*, 2025.

Jinyang Guo, Jianyu Wu, Zining Wang, Jiaheng Liu, Ge Yang, Yifu Ding, Ruihao Gong, Haotong Qin, and Xianglong Liu. Compressing large language models by joint sparsification and quantization. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 16945–16957. PMLR, 2024.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

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

Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Yuhan Dong, Yu Wang, et al. Flashdecoding++: Faster large language model inference with asynchronization, flat gemm optimization, and heuristics. *Proceedings of Machine Learning and Systems*, 6:148–161, 2024.

Wenxuan Huang, Zijie Zhai, Yunhang Shen, Shaosheng Cao, Fei Zhao, Xiangfeng Xu, Zheyu Ye, and Shaohui Lin. Dynamic-LLaVA: Efficient multimodal large language models via dynamic vision-language context sparsification. In *International Conference on Learning Representations*, 2025.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

Shibo Jie, Yehui Tang, Kai Han, Zhi-Hong Deng, and Jing Han. SpeCache: Speculative key-value caching for efficient generation of LLMs. In *International Conference on Machine Learning*, 2025.

Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5:111–126, 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Jaeseong Lee, Seung-won Hwang, Aurick Qiao, Daniel F. Campos, Zhewei Yao, and Yuxiong He. STUN: Structured-then-unstructured pruning for scalable MoE pruning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025. doi: 10.18653/v1/2025.acl-long.671.

Guanchen Li, Yixing Xu, Zeping Li, Ji Liu, Xuanwu Yin, Dong Li, and Emad Barsoum. Týr-the-pruner: Structural pruning LLMs via global sparsity distribution optimization. In *Advances in Neural Information Processing Systems 38*, 2025. doi: 10.52202/085713-4989.

Lujun Li, Peijie Dong, Zhenheng Tang, Xiang Liu, Qiang Wang, Wenhan Luo, Wei Xue, Qifeng Liu, Xiaowen Chu, and Yike Guo. Discovering sparsity allocation for layer-wise pruning of large language models. *Advances in Neural Information Processing Systems*, 37:141292–141317, 2024.

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

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. QServe: W4A8KV4 quantization and system co-design for efficient LLM serving. In *Conference on Machine Learning and Systems*, 2025.

Defu Liu, Yixiao Zhu, Zhe Liu, Yi Liu, Changlin Han, Jinkai Tian, Ruihao Li, and Wei Yi. A survey of model compression techniques: past, present, and future. *Frontiers in Robotics and AI*, 12: 1518965, 2025a.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: LLM quantization with learned rotations. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=ogO6DGE6FZ.

Yao Lu, Hao Cheng, Yujie Fang, Zeyu Wang, Jiaheng Wei, Dongwei Xu, Qi Xuan, Xiaoniu Yang, and Zhaowei Zhu. Reassessing layer pruning in llms: New insights and methods. *arXiv preprint arXiv:2411.15558*, 2024.

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

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20192–20204, 2025.

Haoqian Meng, Yilun Luo, Yafei Zhao, Wenyuan Liu, Huaqing Zheng, Xindian Ma, and Peng Zhang. SharQ: Bridging activation sparsity and FP4 quantization for LLM inference. *arXiv preprint arXiv:2606.26587*, 2026.

Xiang Meng, Kayhan Behdin, Haoyue Wang, and Rahul Mazumder. ALPS: Improved optimization for highly sparse one-shot pruning for large language models. In *Advances in Neural Information Processing Systems 37*, 2024. doi: 10.52202/079017-1187.

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

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

Mahsa Mozafarinia et al. Towards explaining deep neural network compression through a probabilistic latent space. *arXiv preprint arXiv:2403.00155*, 2024.

Mohammad Mozaffari, Amir Yazdanbakhsh, and Maryam Mehri Dehnavi. SLim: One-shot quantization and sparsity with low-rank approximation for LLM weight compression. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=4UfRP8MopP.

Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Compact language models via pruning and knowledge distillation. *Advances in Neural Information Processing Systems*, 37:41076–41102, 2024.

Zhenyu Ning, Jieru Zhao, Qihao Jin, Wenchao Ding, and Minyi Guo. Inf-mllm: Efficient streaming inference of multimodal large language models on a single gpu. *arXiv preprint arXiv:2409.09086*, 2024.

NVIDIA. *NVIDIA CUDA Toolkit 12.1*. NVIDIA Corporation, Santa Clara, CA, 2023. URL https://developer.nvidia.com/cuda-toolkit.

NVIDIA. Nvidia cutlass library, 2023. URL https://github.com/NVIDIA/cutlass/.

Sihyeong Park, Sungryeol Jeon, Chaelyn Lee, Seokhun Jeon, Byung-Soo Kim, and Jemin Lee. A survey on inference engines for large language models: Perspectives on optimization and efficiency. *arXiv preprint arXiv:2505.01658*, 2025.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021. doi: 10.1145/3474381.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. OmniQuant: Omnidirectionally calibrated quantization for large language models. In *International Conference on Learning Representations*, 2024.

Sharath Turuvekere Sreenivas, Saurav Muralidharan, Raviraj Joshi, Marcin Chochowski, Ameya Sunil Mahabaleshwarkar, Gerald Shen, Jiaqi Zeng, Zijia Chen, Yoshi Suhara, Shizhe Diao, et al. Llm pruning and distillation in practice: The minitron approach. *arXiv preprint arXiv:2408.11796*, 2024.

Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 173–191, 2024a.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Jinguang Wang, Jingyu Wang, Haifeng Sun, Tingting Yang, Zirui Zhuang, Wanyi Ning, Yuexi Yin, Qi Qi, and Jianxin Liao. Mergequant: Accurate 4-bit static quantization of large language models by channel-wise calibration. *arXiv preprint arXiv:2503.07654*, 2025.

Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. Outlier suppression: Pushing the limit of low-bit transformer language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 17402–17414. Curran Associates, Inc., 2022. doi: 10.52202/068431-1265.

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

Peng Xu, Wenqi Shao, Mengzhao Chen, Shitao Tang, Kaipeng Zhang, Peng Gao, Fengwei An, Yu Qiao, and Ping Luo. BESA: Pruning large language models with blockwise parameter-efficient sparsity allocation. In *International Conference on Learning Representations (ICLR)*, 2024.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183, 2022.

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, Ajay Kumar Jaiswal, Mykola Pechenizkiy, Yi Liang, Michael Bendersky, Zhangyang Wang, and Shiwei Liu. Outlier weighed layerwise sparsity (OWL): A missing secret sauce for pruning LLMs to high sparsity. In *International Conference on Machine Learning*, 2024.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025. doi: 10.18653/v1/2025.acl-long.1126.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. doi: 10.18653/v1/p19-1472.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, pp. 12360–12371, 2019.

Di Zhang, Xun Wu, Shaohan Huang, Yudong Wang, Hanyong Shao, Yingbo Hao, Zewen Chi, Li Dong, Ting Song, Yan Xia, Zhifang Sui, and Furu Wei. Sparse-BitNet: 1.58-bit LLMs are naturally friendly to semi-structured sparsity. *arXiv preprint arXiv:2603.05168*, 2026.

Yuxin Zhang, Mingbao Lin, ZhiHang Lin, Yiting Luo, Ke Li, Fei Chao, Yongjian Wu, and Rongrong Ji. Learning best combination for efficient n:m sparsity. In *Advances in Neural Information Processing Systems*, volume 35, pp. 941–953. Curran Associates, Inc., 2022. doi: 10.52202/068431-0069.

Yuxin Zhang, Lirui Zhao, Mingbao Lin, Yunyun Sun, Yiwu Yao, Xingjia Han, Jared Tanner, Shiwei Liu, and Rongrong Ji. Dynamic sparse no training: Training-free fine-tuning for sparse LLMs. In *International Conference on Learning Representations*, 2024.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate LLM serving. In *Conference on Machine Learning and Systems*, 2024.

AI USE STATEMENT

We used generative AI tools to translate the text (Claude Opus 5.0), to check grammar and spelling and polish the wording (GPT-5.6), to guide the drawing of the figures (Gemini), and to check the content (Grok). We did not use generative AI tools to modify code. We have reviewed all AI-assisted work: we checked every reference for hallucinated citations. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The details needed to reproduce the results of this paper, including the search settings, the search procedure, the evaluation metrics, and the software setup, are given in this appendix.

APPENDIX

Table A1 lists the contents of each part of the appendix.

APPENDIX A.0: LIMITATIONS

Prefill acceleration across sequence lengths. Prompt lengths in deployment range from a few hundred to several thousand tokens, and the share of runtime spent in prefill, which is where a smaller GEMM pays off, changes with that length. The single-block prefill acceleration in this paper is measured at 2048 tokens, so how the gain of ASAF over the quantized baseline evolves at shorter and longer contexts lies outside this study.

Quantization group size. Group-wise weight quantization trades scale storage against accuracy, and practitioners choose the group size to fit their memory and kernel constraints. The allocation in this paper is evaluated with per-channel weight scales and with 128-element groups, so its behaviour under other group sizes lies outside this study.

Table A1: Contents of each part of the appendix.

Section	Contents
Appendix A.0	Limitations
Appendix A.1	Extended Related Work
Appendix A.2	Motivation
Appendix A.3	Architectural Justification for Consecutive Layer Grouping
Appendix A.4	Quantization Framework
Appendix A.5	Dynamic Programming Background
Appendix A.6	Dynamic Programming Structure of the ASAF Objective
Appendix A.7	ASAF Framework Visual Illustration
Appendix A.8	Coarse-Grained Optimization
Appendix A.9	Fine-Grained Optimization
Appendix A.10	Tabulation
Appendix A.11	Complete Hyperparameter Specifications
Appendix A.12	Accuracy Results
Appendix A.13	Computational Efficiency Analysis
Appendix A.14	NVIDIA 4090 GPU Performance Analysis
Appendix A.15	Llama-3 Experimental Results
Appendix A.16	Comparison with Naive Heuristic Exploration
Appendix A.17	Ablation Study for Accuracy Budget and Sparsity Range
Appendix A.18	Precomputation Overhead Analysis
Appendix A.19	Sparsity Allocation and Sensitivity Analysis
Appendix A.20	Models, Software, and Benchmarks Used
Appendix A.21	Per-Benchmark Results for the Compression Baselines
Appendix A.22	End-to-End Throughput and Latency by Configuration

Stability of the allocation across models. A practitioner who reuses ASAF on a new model needs to know whether the selected layer grouping keeps its shape across model scales, model families and context lengths. This paper reports the accuracy and efficiency of each model separately and shows the selected allocation for Llama-2-7B, so a side-by-side comparison of the allocations selected for different models lies outside this study.

APPENDIX A.1: EXTENDED RELATED WORK

Recent advances in LLM compression have significantly improved inference efficiency while preserving model capabilities. Structured pruning techniques eliminate entire components like attention heads or layers (Dutta et al., 2024; Muralidharan et al., 2024), with approaches like TVAPrune leveraging variational information bottleneck principles to compress model representations (Dutta et al., 2024). Weight quantization methods such as GPTQ (Frantar et al., 2022), AWQ (Lin et al., 2024), and OmniQuant (Shao et al., 2024) enable 4-bit weight representation with minimal performance degradation. Knowledge distillation approaches like MiniLLM (Gu et al., 2023) and GKD (Agarwal et al., 2024) transfer knowledge from larger teacher models to smaller students, with the latter introducing on-policy distillation to address train-inference distribution mismatch. Combined approaches leveraging pruning with distillation have yielded results comparable to the strongest baselines (Sreenivas et al., 2024; Muralidharan et al., 2024).

Building upon these structural approaches, unstructured pruning enhances LLM compression by flexibly removing weights, often achieving higher accuracy at high sparsity levels compared to structured methods (Lee et al., 2025). Recent advancements in this field include post-training one-shot pruning techniques such as SparseGPT (Frantar & Alistarh, 2023), which leverages second-order approximations to prune over 50% of weights in a 175B model while maintaining performance with minimal loss, and Wanda (Sun et al., 2024b), which simplifies the pruning process through activation-aware magnitude pruning for efficient sparsity. Other notable contributions in this category, such as OWL (Yin et al., 2024), Tyr-the-Pruner (Li et al., 2025), and GBLM-Pruner (Das et al., 2023), further refine sparsity allocation and incorporate gradient correction to enhance pruning outcomes. Additionally, optimization-based methods like ALPS (Meng et al., 2024) approach pruning

as a constrained optimization problem, while DSnoT (Zhang et al., 2024) iteratively refines weights to optimize results over multiple steps.

Beyond pruning techniques, model quantization is essential for deploying LLMs efficiently, reducing memory and computation by lowering the precision of weights and activations (Gong et al., 2024). However, the 'outlier' problem, where a few large values dominate the quantization range, poses a significant challenge (Ashkboos et al., 2024c). Early solutions like LLM.int8() (Dettmers et al., 2022) addressed this by retaining some activations in higher precision. Recent advancements include GPTQ (Frantar et al., 2022), which uses second-order information for accurate 4-bit weight quantization, and AWQ (Lin et al., 2024), which protects key weights based on activation statistics. For activation quantization, SmoothQuant (Xiao et al., 2023) redistributes outliers to enable 8-bit quantization, while QuaRot (Ashkboos et al., 2024c) and QuIP (Chee et al., 2023) employ rotation transformations for 4-bit and 2-bit quantization, respectively. Additionally, QUIK (Ashkboos et al., 2024b) and QServe (Lin et al., 2025) combine quantization with system optimizations for practical deployment. Other notable methods include PrefixQuant (Chen et al., 2026) and MergeQuant (Wang et al., 2025) for efficient static quantization.

With models compressed through the aforementioned techniques, large language model inference faces significant computational challenges that researchers address through various optimization approaches. Recent work has focused on KV cache management techniques such as PagedAttention (Kwon et al., 2023), which treats cache as virtual memory to reduce fragmentation, SpeCache (Jie et al., 2025), which intelligently prefetches needed keys, and QuaRot (Ashkboos et al., 2024c) for outlier-free 4-bit inference. Algorithmic optimizations like Cascade Speculative Drafting (Chen et al., 2024) leverage a tiered approach where smaller models draft for larger ones, while N-gram masked self-attention (Chelba et al., 2020) truncates attention windows. Memory efficiency improvements include SWAT (Fu et al., 2025) for sliding window attention and Inf-MLLM (Ning et al., 2024) for streaming inference via attention saddle patterns. Architecture innovations explore MoE approaches like Read-ME (Cai et al., 2024) which refactors dense models into router-decoupled experts, and Dynamic-LLaVA (Huang et al., 2025) which sparsifies vision-language contexts. System-level optimizations include operator fusion in FasterTransformer (Aminabadi et al., 2022), FlashAttention's IO-aware computation (Dao et al., 2022), FlashDecoding++ (Hong et al., 2024) with asynchronous softmax, and specialized schedulers like Sarathi-Serve (Agrawal et al., 2024), vLLM (Kwon et al., 2023), and Llumnix (Sun et al., 2024a) that balance throughput and latency through innovative resource management strategies.

APPENDIX A.2: MOTIVATION

In the deployment of LLMs, quantization and pruning have evolved as independent acceleration techniques with minimal integration. In our explorations, we conduct a series of attempts to investigate the combination of both techniques. Figure A1 demonstrates the inference acceleration achieved on the Llama-2-7B model in a language generation task, after applying 20% sparsity rate to various layers and layer combinations. We observe a significant improvement in model inference speed. Figure A2 shows the model's performance across different quantization and sparsification configurations. As the bit-width of quantization decreases and the sparsity ratio increases, the perplexity rises substantially, indicating a clear trade-off between efficiency and performance.

A.2.1 LAYER-GROUP SENSITIVITY ANALYSIS

The experimental results reveal distinct sensitivity patterns across different layer groups in the Llama-2-7B architecture under 20% sparsity allocation. Early layers (1-8) demonstrate moderate resilience to sparsification, with configuration A1 achieving 5.2% throughput improvement while B1 exhibits 55.8% perplexity degradation.

Middle layers show varying sensitivity patterns: layers 9-16 (A2/B2) achieve 9.4% throughput gains with 62.5% quality degradation, while layers 17-24 (A3/B3) demonstrate the highest single-group acceleration of 13.5% with 65.4% perplexity increase. These middle-late layers exhibit the most severe single-group degradation, suggesting their critical role in semantic processing.

Late layers (25-32) show a balanced pattern with A4 achieving 11.5% throughput improvement while B4 incurs 59.2% quality loss. This pattern suggests that while late layers contain exploitable

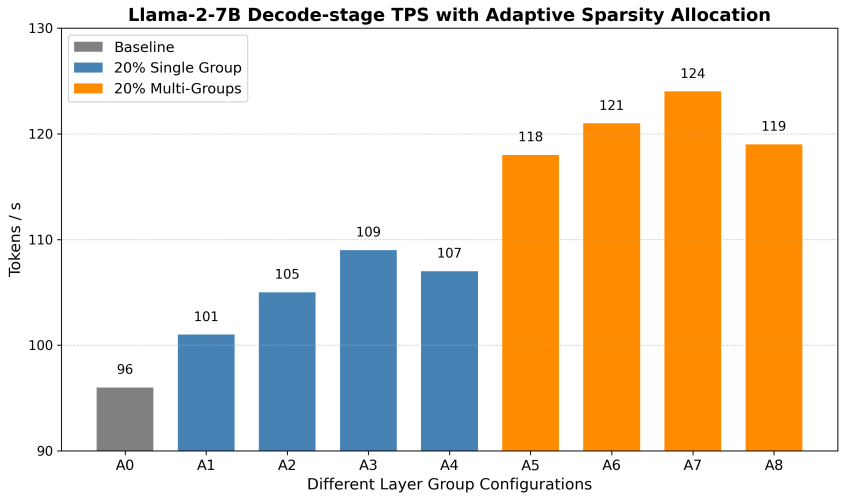


Figure A1: Decode-stage TPS with sparsity allocation across layer groups, based on 8-bit QuaRot quantization. Task and hardware: left-to-right language-model generation, each run takes a 2048-token prompt from the WikiText-103 validation set, allowing Llama-2-7B to autoregressively generate the next 256 tokens, and runs on a single NVIDIA RTX 3090 GPU. Variants on the X-axis denote different layer group sparsity configurations: A0: baseline, no sparsity applied (96 tokens/s); A1: 20% sparsity applied to layers 1-8 (early attention layers, 101 tokens/s); A2: 20% sparsity applied to layers 9-16 (middle attention layers, 105 tokens/s); A3: 20% sparsity applied to layers 17-24 (middle-late layers, 109 tokens/s); A4: 20% sparsity applied to layers 25-32 (late transformer layers, 107 tokens/s); A5: 20% sparsity applied to layers 1-8 and 17-24 simultaneously (118 tokens/s); A6: 20% sparsity applied to layers 9-16 and 25-32 simultaneously (121 tokens/s); A7: 20% sparsity applied to layers 1-8, 9-16, and 17-24 (124 tokens/s); A8: 20% sparsity applied to all layer groups 1-32 (119 tokens/s). Y-axis: measured tokens-per-second during decode phase. Higher is better.

computational redundancy, their sparsification significantly impacts final output quality, though less severely than middle-late layers.

A.2.2 Multi-Group Sensitivity Analysis

The multi-group configurations (A5-A8, B5-B8) reveal the catastrophic nature of naive sparsity combination strategies. Configuration A5, combining layers 1-8 and 17-24 with 20% sparsity each, achieves 22.9% throughput improvement but incurs 133.6% perplexity degradation (B5).

The most aggressive multi-group configuration A7 (layers 1-8, 9-16, 17-24) demonstrates maximum throughput gains of 29.2% but suffers catastrophic quality collapse with 149.5% perplexity degradation (B7). Paradoxically, the all-groups configuration A8 shows slightly reduced acceleration (24.0%) and degradation (136.7%), suggesting complex non-linear interactions when the entire model undergoes simultaneous sparsification.

Appendix A.3: Architectural Justification for Consecutive Layer Grouping

The empirical pruning literature consistently reports a U-shaped fragility profile across Transformer depth: the first one or two layers and the final one or two layers carry disproportionate weight in downstream task quality, while mid-depth blocks behave as near-interchangeable units (Men et al., 2025; Ma et al., 2023; Lu et al., 2024). ASAF's choice of consecutive grouping is designed against this profile.

Why consecutive over arbitrary clustering. Consecutive groups respect the Transformer's residual stream, which incrementally refines representations along depth. Arbitrary (e.g., k-means on

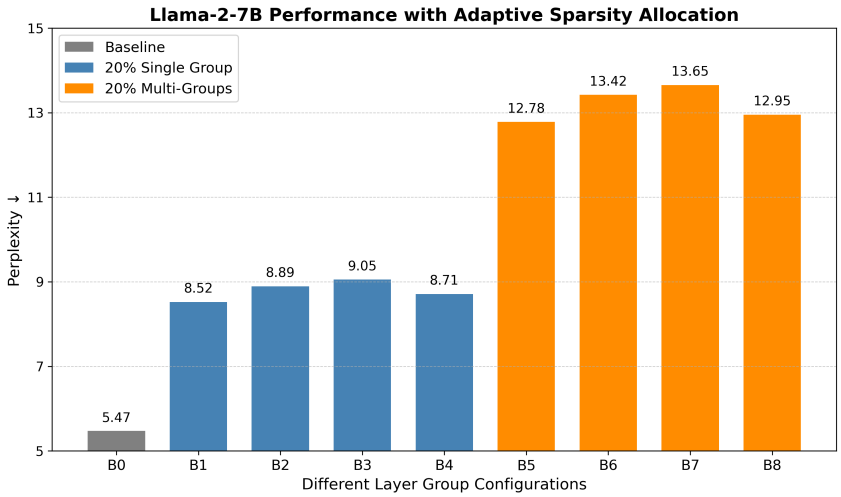


Figure A2: Performance of quantization and layer-group sparsity allocation on Llama-2-7B. Task and hardware: left-to-right language-model evaluation on the WikiText-2 validation subset. The evaluation uses approximately $1\,000$ passages, each with a sequence length of 2048, running on a single NVIDIA RTX 3090 GPU with 8-bit QuaRot quantization applied to all weights. Variants on the X-axis denote different layer group sparsity configurations: B0: 8-bit quantized baseline, no additional sparsity (5.47 perplexity); B1: 8-bit quantization + 20% sparsity on layers 1-8 (8.52 perplexity, 55.8% degradation); B2: 8-bit quantization + 20% sparsity on layers 9-16 (8.89 perplexity, 62.5% degradation); B3: 8-bit quantization + 20% sparsity on layers 17-24 (9.05 perplexity, 65.4% degradation); B4: 8-bit quantization + 20% sparsity on layers 25-32 (8.71 perplexity, 59.2% degradation); B5: 8-bit quantization + 20% sparsity on layers 1-8 and 17-24 (12.78 perplexity, 133.6% degradation); B6: 8-bit quantization + 20% sparsity on layers 9-16 and 25-32 (13.42 perplexity, 145.3% degradation); B7: 8-bit quantization + 20% sparsity on layers 1-8, 9-16, and 17-24 (13.65 perplexity, 149.5% degradation); B8: 8-bit quantization + 20% sparsity on all layer groups (12.95 perplexity, 136.7% degradation). Y-axis: measured perplexity on WikiText-2 (↓). Lower is better.

sensitivity) grouping can split tightly coupled refinement stages and introduces skip-pattern artifacts in the KV cache that complicate kernel-level fusion. Consecutive grouping additionally isolates the fragile end layers from the interchangeable interior, so the coarse-stage allocation never averages a fragile layer's sensitivity with an interchangeable one.

Search-space reduction. For an L-layer model with $|S|$ discretized sparsity rates, the brute-force search space is $|S|^L$. ASAF's two-stage decomposition reduces this to $|S|^G$ in the coarse stage (where $G \ll L$ is the number of groups) followed by $|S|^{L/G}$ within-group fine-grained search, an exponential reduction in the dominant term. The fine-grained stage acts as an escape hatch for cases where within-group sensitivity variance is non-trivial, recovering per-layer expressivity that pure block-wise allocation would lose. Empirical validation of the smoothness assumption is provided in Appendix A.19.

Appendix A.4: Quantization Framework

In the framework illustrated in Figure A3, the hidden state X is first multiplied by a random orthogonal matrix R stored in FP(m) precision. Because $R^\top R = I$, this transformation preserves Euclidean norms and inner products, i.e. $(RX)^\top(RY) = X^\top Y$ (Ashkboos et al., 2024c). Consequently, substituting $X \mapsto RX$ and $W \mapsto R^\top W$ leaves each linear layer's output unchanged:

$$\underbrace{XW}_{\text{original}} = \underbrace{(RX)\big(R^\top W\big)}_{\text{rotated}}. \tag{A1}$$

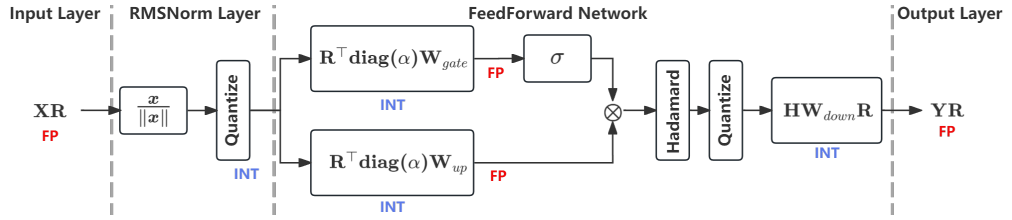


Figure A3: Low-bit quantization pipeline for a feed-forward network block with quantization and rotation.

This computational invariance (Ashkboos et al., 2024a) enables the framework to redistribute heavy-tailed activation energy across dimensions before quantization, thereby mitigating the "outlier" problem that hampers uniform low-bit encodings (Ashkboos et al., 2024c).

After rotation, the activations are quantized to INT(n). The feed-forward weights W_{gate} and W_{up} are premultiplied by $R^\top$ and rescaled by $\mathrm{diag}(\alpha)$ (RMSNorm) (Zhang & Sennrich, 2019). The non-linearity σ is applied, followed by an on-the-fly Hadamard transform H; this matrix is also orthogonal, so fusing H into the down-projection W_{down} preserves functional equivalence while further flattening variance. A second INT(n) quantization converts the output back to low precision before casting to FP(m).

Because every orthogonal transform is absorbed into adjacent linear layers during an offline preprocessing step, the run-time kernel sequence matches that of the baseline network while operating entirely on 4-bit integers. Combined with per-channel GPTQ calibration (Frantar et al., 2022), this rotation–quantize–fuse pipeline achieves end-to-end INT4 inference without mixed-precision fallbacks.

Appendix A.5: Dynamic Programming Background

A.5.1 Algorithmic Paradigm

Dynamic programming is both a mathematical optimization method and an algorithmic paradigm developed by Richard Bellman in the 1950s (Bellman, 1957). The approach simplifies complex problems by breaking them down into simpler subproblems in a recursive manner, then combining their solutions to solve the original problem (Cormen et al., 2009). Figure A4 illustrates this decomposition of a problem into subproblems.

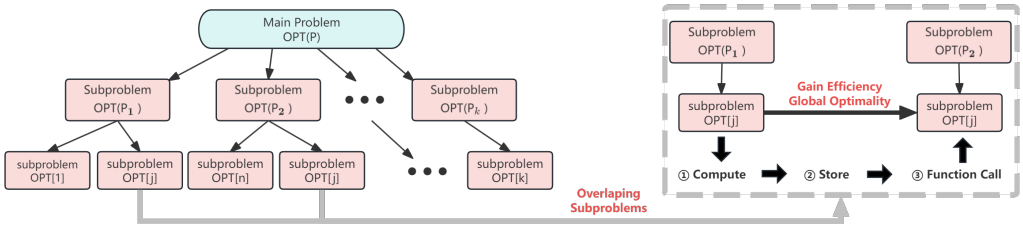


Figure A4: Illustration of dynamic programming approach showing problem decomposition into subproblems.

The paradigm applies when a problem exhibits two fundamental properties: optimal substructure and overlapping subproblems. Unlike divide-and-conquer algorithms, which solve entirely independent subproblems, dynamic programming exploits the fact that subproblems are not independent; the same subproblems arise repeatedly during the recursive solution process.

A.5.2 Optimal Substructure Property

A problem exhibits optimal substructure if an optimal solution can be constructed efficiently from optimal solutions of its subproblems (Cormen et al., 2009). This property is fundamental because it ensures that solving subproblems optimally contributes to the global optimum.

Mathematically, optimal substructure can be expressed as a functional relationship. For a problem P, if $\text{OPT}(P)$ represents the optimal solution, then:

$$\text{OPT}(P) = f(\text{OPT}(P_1), \text{OPT}(P_2), \ldots, \text{OPT}(P_k)), \tag{A2}$$

where f is a function that combines optimal solutions of subproblems $\{P_i\}$ to yield the optimal solution of the original problem.

A.5.3 Overlapping Subproblems Property

The overlapping subproblems property distinguishes dynamic programming from divide-and-conquer approaches. This property requires that the space of subproblems be small relative to the total number of recursive calls, meaning the same subproblems are solved repeatedly (Cormen et al., 2009).

Dynamic programming exploits this overlap through memoization, storing computed solutions in a table for subsequent lookup. This technique reduces the time complexity from exponential to polynomial by ensuring each distinct subproblem is solved exactly once (Dasgupta et al., 2006).

A.5.4 Mathematical Formulation

Dynamic programming algorithms follow a general mathematical structure based on the principle of optimality. The fundamental recurrence relation takes the form:

$$\text{OPT}[i] = \min_{j \in \mathcal{J}(i)} \{\text{OPT}[j] + \text{Cost}(j, i)\}, \tag{A3}$$

where $\text{OPT}[i]$ represents the optimal solution value for subproblem i, $\mathcal{J}(i)$ denotes the set of feasible predecessor states, and $\text{Cost}(j, i)$ represents the immediate cost of transitioning from state j to state i.

The state space design constitutes a critical component of dynamic programming formulations. States must encapsulate sufficient information to make optimal decisions while maintaining computational tractability. The dimensionality of the state space directly affects both algorithm correctness and efficiency.

Boundary conditions provide the foundation for recursive computations. These base cases correspond to trivial subproblems that can be solved directly without further decomposition:

$$\text{OPT}[0] = \text{base_value}. \tag{A4}$$

The optimization objective varies depending on the problem context. Minimization problems seek $\min_j$, maximization problems use $\max_j$, and counting problems sum over all valid transitions. Each formulation requires careful consideration of how subproblem solutions combine.

Appendix A.6: Dynamic Programming Structure of the ASAF Objective

We show that the two properties of Appendix A.5, optimal substructure and overlapping subproblems, hold for the layer-grouping and sparsity-allocation formulation of Eq. 1 of the main paper, and how they dictate the algorithmic structure adopted in Appendices A.8–A.10.

A.6.1 Optimal Substructure of the ASAF Objective

Let $\text{OPT}(\mathcal{L}, \delta)$ denote the minimum total computational FLOPs achievable on a contiguous layer subsequence $\mathcal{L} \subseteq \{1, \ldots, p\}$ subject to an accuracy-degradation budget $\delta \leq \delta_{\max}$. If the optimal

solution partitions $\mathcal{L}$ into groups $\{\mathcal{L}_i^*\}_{i=1}^{G^*}$ with sparsity rates $\{s_i^*\}$, then for any prefix $\bigcup_{i\le k}\mathcal{L}_i^*$ the corresponding sub-allocation must itself be optimal under the residual budget $\delta - \sum_{i\le k}\bar{\xi}(\mathcal{L}_i^*, s_i^*)$. The argument is by exchange: were the prefix sub-allocation strictly suboptimal, replacing it with the actual optimum would preserve feasibility (residual budget unchanged) while strictly reducing the FLOP cost, contradicting optimality on the discretized search space. This optimality-of-prefixes property allows us to recursively define the optimum for $\mathcal{L}$ in terms of optima on shorter prefixes plus the cost of one additional group, which is exactly the recurrence solved by our DP.

A.6.2 Overlapping Subproblems and the Need for Tabulation

A naive enumeration over $(G, \{\mathcal{L}_i\}, \{s_i\})$ traverses an exponentially large schedule space; however, the same per-block quantities recur across an exponentially large family of grouping schemes. Concretely, for every contiguous block $[i, j] \subseteq \{1, \dots, p\}$ and every admissible sparsity rate $s \in \mathcal{S}$, both the FLOP cost

$$H[i][\,j-i+1\,][s] \;=\; \sum_{l=i}^{j} \phi_l\,(1-s) \tag{A5}$$

and the calibration-set sensitivity

$$\Xi[i][\,j-i+1\,][s] \;=\; \xi\big([i,j], s\big) \tag{A6}$$

appear identically in every grouping that contains $[i, j]$ as one of its groups. Without sharing, each occurrence triggers a recomputation; with sharing, all occurrences look up the same precomputed entry. The number of distinct $(i, j-i+1, s)$ triples is $O(p^2 \cdot |\mathcal{S}|)$, which is the size of the tabulation tables H and Ξ described in Appendix A.10. Once H and Ξ are populated, every cost or sensitivity query inside the DP is an $O(1)$ table lookup, reducing an exponential search to polynomial time.

A.6.3 General Recurrence and ASAF's Two-Phase Instantiation

Our DP follows the canonical recurrence

$$\mathrm{OPT}[i] \;=\; \min_{j\in\mathcal{J}(i)} \Big\{\, \mathrm{OPT}[j] \;+\; \mathrm{Cost}(j,i) \,\Big\}, \tag{A7}$$

where $\mathcal{J}(i)$ is the set of feasible predecessor states and $\mathrm{Cost}(j, i)$ is the cost of extending an allocation from a prefix of length j to a prefix of length i. In ASAF, $\mathrm{Cost}(j, i)$ is realized as $H[j+1][i-j][s]$ (with sparsity rate s optimized within the active interval), and the budget tracking is folded into the state through an additional axis indexing accumulated sensitivity in discretized units. Because solving Eq. 1 of the main paper directly couples three discrete decisions—group count G, group memberships $\{\mathcal{L}_i\}$, and sparsity rates $\{s_i\}$—we factor the optimization into two DP passes that share H and Ξ:

- The coarse-grained pass (Appendix A.8) selects G^* and per-group sparsity intervals $\{I_i^*\}$ via state $\mathrm{DP}_{\mathrm{coarse}}[g][b]$;
- The fine-grained pass (Appendix A.9) refines each group's exact layer boundaries and sparsity rate within I_i^*.

The resulting aggregate complexity, stated in Appendix A.13, is polynomial in p and $|\mathcal{S}|$, replacing the exponential brute-force search with a tractable two-stage procedure on the same tabulation tables.

Appendix A.7: ASAF Framework Visual Illustration

A.7.1 Coarse-Grained Optimization Phase

Figure A5 illustrates the coarse-grained optimization phase of the ASAF framework, which addresses the fundamental challenge of determining optimal layer grouping strategies and sparsity interval refinement. The process begins with individual transformer layers $L_1, L_2, \dots, L_p$ from the large language model, where each layer is represented as an independent computational unit with specific FLOPs characteristics ϕ_l.

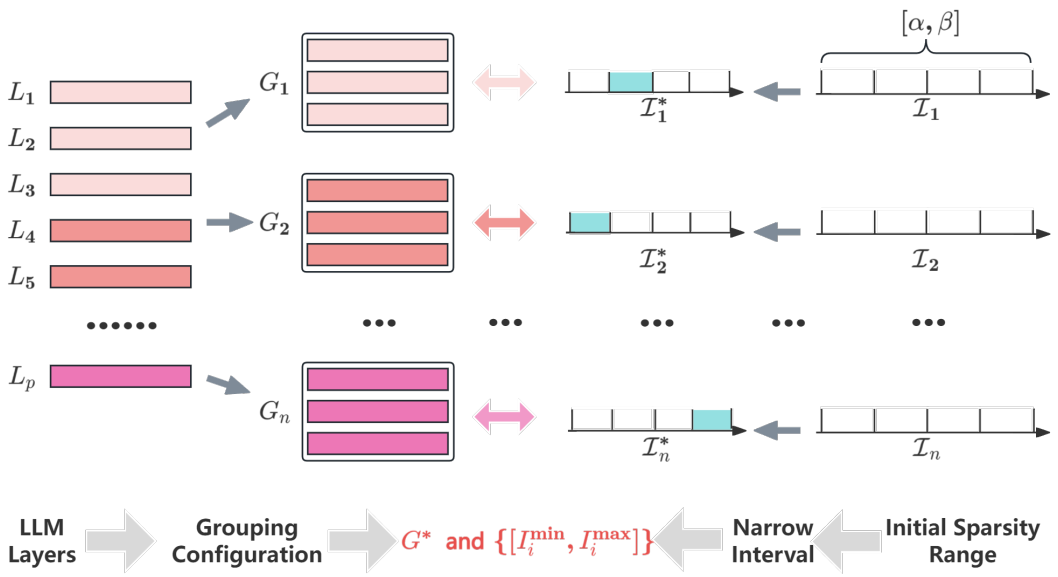


Figure A5: Coarse-grained optimization process in the ASAF framework. The phase transforms individual LLM layers $L_1, L_2, \ldots, L_p$ into optimal grouping configurations $G_1, G_2, \ldots, G_n$ while simultaneously refining sparsity search intervals from the initial range $[\alpha, \beta]$ to narrowed intervals $\{[I_i^{\min}, I_i^{\max}]\}$ for each group, where $G^* = n$ denotes the optimal number of groups. The optimization process employs iterative interval refinement strategies $\mathcal{I}_1^*, \mathcal{I}_2^*, \ldots, \mathcal{I}_n^*$ to achieve computational efficiency while maintaining sensitivity constraints.

The grouping transformation process aggregates these individual layers into cohesive groups $G_1, G_2, \ldots, G_n$, where each group G_i contains a set of consecutive layers $\mathcal{L}_i = \{l_{\text{start}}^{(i)}, l_{\text{start}}^{(i)} + 1, \ldots, l_{\text{end}}^{(i)}\}$. This grouping strategy ensures that layers within each group can be managed collectively while preserving the sequential structure of the transformer architecture.

The simultaneous sparsity interval refinement process transforms the initial broad search space $[\alpha, \beta]$ into narrowed, group-specific intervals $\{[I_i^{\min}, I_i^{\max}]\}$. The refinement process utilizes the RefineIntervals$(\cdot)$ and NarrowInterval$(\cdot)$ functions to achieve progressive convergence toward optimal sparsity allocations.

The optimization trajectory shown in the figure demonstrates how the algorithm iteratively narrows the search space through successive refinements $\mathcal{I}_1^*, \mathcal{I}_2^*, \ldots, \mathcal{I}_n^*$, where each iteration produces tighter bounds on feasible sparsity rates. This process continues until convergence criteria are satisfied, yielding the optimal number of groups G^* and their corresponding refined sparsity intervals.

A.7.2 Fine-Grained Optimization Phase

Figure A6 depicts the fine-grained optimization phase, which performs precise decision-making within the framework established by the coarse-grained phase. This phase receives as input the optimal number of groups G^* and the refined sparsity intervals $\{[I_i^{\min}, I_i^{\max}]\}$ determined in the previous phase.

The layer allocation component systematically determines the exact number of consecutive layers assigned to each group. This process involves solving the optimization problem $\{j^*, s^*\}_i = \arg\min_{j,s}\{T[i][j][s]\}$ for each group i, where $T[i][j][s]$ represents the total FLOPs cost for group i containing j consecutive layers with sparsity rate s. The allocation process ensures complete coverage of all transformer layers while respecting continuity constraints.

The sparsity selection component operates in parallel with layer allocation, determining optimal sparsity rates from the discretized candidate sets generated by the Discretize$(\cdot)$ function. For each

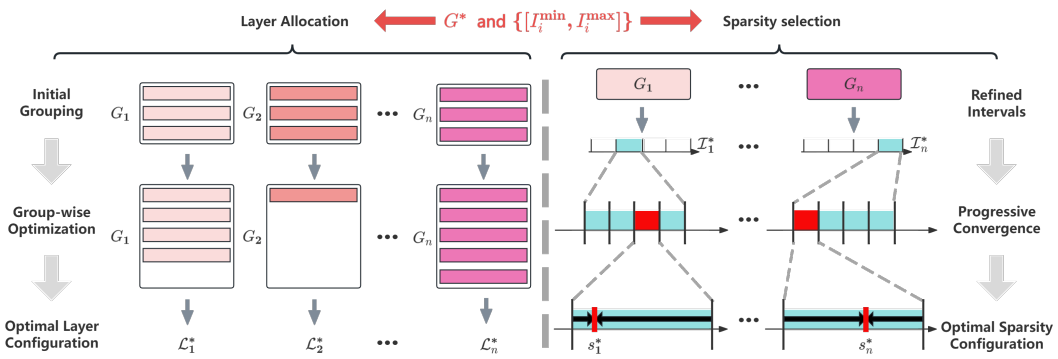


Figure A6: Fine-grained optimization process in the ASAF framework. Building upon the coarse-grained results G^* and refined intervals $\{[I_i^{\min}, I_i^{\max}]\}$, this phase performs precise layer allocation and sparsity selection through group-wise optimization. The process progressively converges from initial grouping configurations through iterative optimization to optimal layer configurations $\mathcal{L}_1^*, \mathcal{L}_2^*, \ldots, \mathcal{L}_n^*$ and optimal sparsity rates $s_1^*, s_2^*, \ldots, s_n^*$, where $G^* = n$ denotes the optimal number of groups.

group i, the algorithm evaluates sparsity candidates $S_{\text{cand}} = \{I_i^{\min} + k\Delta : k \in \mathbb{Z}, 0 \leq k \leq \lfloor (I_i^{\max} - I_i^{\min})/\Delta \rfloor\}$ with sampling resolution $\Delta = 0.5\%$.

The progressive convergence mechanism illustrated in the figure shows how the optimization iteratively refines both layer assignments and sparsity selections. The process employs tabulation table lookups $H[l][j][s]$ to efficiently evaluate different configurations without redundant computation. Each iteration improves upon previous solutions by leveraging the optimal substructure property inherent in the problem formulation.

The final optimization output produces the complete ASAF solution: optimal layer configurations $\mathcal{L}_1^*, \mathcal{L}_2^*, \ldots, \mathcal{L}_n^*$ specifying the exact layer ranges for each group, and optimal sparsity rates $s_1^*, s_2^*, \ldots, s_n^*$ that minimize total computational FLOPs while satisfying sensitivity constraints $\sum_{i=1}^{G^*} \xi(\mathcal{L}_i^*, s_i^*) \leq \delta_{\max}$.

Appendix A.8: Coarse-Grained Optimization

The coarse-grained optimization phase serves as the foundation of our ASAF framework by establishing the optimal group structure and narrowing the sparsity search space for subsequent fine-grained optimization. This phase addresses the fundamental challenge of determining how many layer groups should be formed and what sparsity intervals each group should operate within, effectively decomposing the exponential search space into manageable subproblems.

A.8.1 Mathematical Formulation

The coarse-grained optimization addresses the first subproblem of our original formulation in Eq. 3 of the main paper by minimizing the total computational FLOPs across all possible group configurations while satisfying accuracy degradation constraints:

$$\{G^*, \{I_i^*\}_{i=1}^{G^*}\} = \arg \min_{G, \{I_i\}_{i=1}^{G}} \left\{ \sum_{i=1}^{G} \min_{\mathcal{L}_i, s_i \in I_i} \sum_{l \in \mathcal{L}_i} \phi_l \times (1 - s_i) \right\}, \tag{A8}$$

where G represents the number of groups, I_i denotes the sparsity interval for group i, $\mathcal{L}_i$ represents consecutive layers in group i, s_i is the sparsity rate for group i, and ϕ_l is the original computational cost of layer l. The constraint ensures $\sum_{i=1}^{G} \xi(\mathcal{L}_i, s_i) \leq \delta_{\max}$, where $\xi(\mathcal{L}_i, s_i)$ quantifies the accuracy degradation caused by applying sparsity rate s_i to layer group $\mathcal{L}_i$.

The dynamic programming state formulation defines $\text{DP}_{\text{coarse}}[g][b]$ as the minimum total FLOP cost when using exactly g groups with discretized accuracy degradation budget b:

$$\text{DP}_{\text{coarse}}[g][b] = \min_{\{I_i\}_{i=1}^{g}} \left\{ \sum_{i=1}^{g} \min_{\mathcal{L}_i, s_i \in I_i} \sum_{l \in \mathcal{L}_i} \phi_l \times (1 - s_i) \right\}, \tag{A9}$$

where the state represents the optimal cost achievable with g groups under accuracy degradation budget $b \times \Delta$ (with Δ being the discretization step), and the minimization considers all valid interval partitions and corresponding layer-sparsity assignments.

The state transition equation considers all possible ways to add one more group by selecting an appropriate interval subset:

$$\text{DP}_{\text{coarse}}[g][b] = \min_{I \subseteq [\alpha, \beta]} \left\{ \text{OptimalCost}(I) + \text{DP}_{\text{coarse}}[g-1][b - \xi_{\text{cost}}(I)/\Delta] \right\}, \tag{A10}$$

where I represents the sparsity interval assigned to the g-th group, $\text{OptimalCost}(I)$ denotes the minimum FLOP cost achievable within interval I, and $\xi_{\text{cost}}(I)$ represents the accuracy degradation.

A.8.2 Algorithmic Analysis

The dynamic programming formulation operates on a two-dimensional state space where $\text{DP}_{\text{coarse}}[g][b]$ represents the minimum achievable computational cost when utilizing exactly g groups with discretized accuracy degradation budget b. The discretization parameter $\Delta = 0.5\%$ transforms the continuous constraint into a tractable discrete optimization problem while maintaining sufficient precision for practical deployment.

The initialization establishes boundary conditions essential for correctness. The infinite initialization for $\text{DP}_{\text{coarse}}[g][b]$ ensures only valid transitions result in finite costs, while $\text{DP}_{\text{coarse}}[0][0] = 0$ represents the base case with no groups and no degradation.

The interval generation constructs candidate sparsity intervals $\mathcal{I}$ that partition $[\alpha, \beta]$ into meaningful subranges. Each interval $I \in \mathcal{I}$ represents a potential sparsity operating range for a layer group, enabling exploration of different sparsity allocation granularities.

The core dynamic programming loop systematically explores group counts and accuracy budgets through three nested iterations. The outermost loop over g considers configurations from single groups to maximum granularity where $G_{\max} = p$. The middle loop over budget b discretizes the accuracy constraint, while the innermost loop over intervals I evaluates each potential sparsity assignment.

Within each iteration, the algorithm queries tabulation tables H and Ξ for precomputed costs and degradation values. The tabulation mechanism transforms expensive evaluations into constant-time operations, enabling scalability to large layer counts. The cost computation combines immediate interval contribution with optimal subproblem costs, maintaining optimal substructure.

The feasibility check $b - \text{accuracy_cost} \geq 0$ ensures configurations remain within the accuracy threshold $\delta_{\max}$. When feasible configurations are identified, the minimization operation updates the dynamic programming table only for improved solutions.

Algorithm 3 Coarse-grained Optimization

Require: Layer count p, sparsity range $[\alpha, \beta]$, accuracy degradation threshold $\delta_{\max}$
Ensure: Optimal group number G^* and intervals $\{[I_i^{\min}, I_i^{\max}]\}$

Initialize $\text{DP}_{\text{coarse}}[0..p][0..\delta_{\max}/\Delta] \leftarrow \infty$
Set boundary condition: $\text{DP}_{\text{coarse}}[0][0] \leftarrow 0$
Generate interval candidates $\mathcal{I} = \{I : I \subseteq [\alpha, \beta]\}$
Main DP Loop:
for $g = 1$ **to** p **do**
  for $b = 0$ **to** $\delta_{\max}/\Delta$ **do**
    for each interval $I \in \mathcal{I}$ **do**
      cost $\leftarrow$ OptimalCost(I) using tabulation H
      accuracy_cost $\leftarrow \xi_{\text{cost}}(I)/\Delta$ using tabulation Ξ
      if $b -$ accuracy_cost ≥ 0 **then**
        total_cost $\leftarrow$ cost $+ \text{DP}_{\text{coarse}}[g-1][b - \text{accuracy_cost}]$
        $\text{DP}_{\text{coarse}}[g][b] \leftarrow \min(\text{DP}_{\text{coarse}}[g][b], \text{total_cost})$
      end if
    end for
  end for
end for
Solution Extraction:
$G^* \leftarrow \arg\min_g \{\min_b \text{DP}_{\text{coarse}}[g][b]\}$
$\{I_i^{\min}, I_i^{\max}\} \leftarrow$ BacktrackOptimalIntervals($\text{DP}_{\text{coarse}}, G^*$)
return G^*, $\{[I_i^{\min}, I_i^{\max}]\}_{i=1}^{G^*}$

The solution extraction identifies the optimal configuration by examining all computed states: $G^* = \arg\min_g \{\min_b \text{DP}_{\text{coarse}}[g][b]\}$. The backtracking procedure reconstructs specific interval assignments by tracing decisions that led to the optimal solution, producing refined sparsity intervals that narrow the search space from $[\alpha, \beta]$.

State Transition Intuition. The dynamic programming transitions embody intuitive decision-making processes. In the coarse-grained phase, each state transition addresses the question: "Given that I have optimally allocated the first $g-1$ groups, what sparsity interval should I assign to the g-th group to minimize total cost while staying within the accuracy budget?" This decomposition enables systematic exploration of interval assignments without reconsidering previous group decisions. The state $\text{DP}_{\text{coarse}}[g][b]$ encapsulates all optimal ways to use exactly g groups with budget b, allowing efficient evaluation of adding one more group with a specific interval choice.

A.8.3 Output of the Coarse-Grained Pass

Upon completion, the coarse-grained optimization produces two critical outputs: the optimal number of groups G^* that minimizes computational overhead while satisfying accuracy constraints, and refined sparsity intervals $\{[I_i^{\min}, I_i^{\max}]\}_{i=1}^{G^*}$ that narrow the search space from the original range $[\alpha, \beta]$. These outputs provide essential guidance for the subsequent fine-grained optimization phase, significantly reducing the search space while preserving the optimum attainable on the discretized search space.

Appendix A.9: Fine-Grained Optimization

The fine-grained optimization phase receives the optimal group number G^* and refined sparsity intervals from the coarse-grained phase and determines the precise layer allocation and sparsity rate assignment that minimizes computational FLOPs while satisfying accuracy constraints. This phase operates on a significantly reduced search space compared to the original problem, enabling detailed optimization within the refined parameter ranges.

A.9.1 MATHEMATICAL FORMULATION

The fine-grained optimization addresses the second subproblem of our original formulation by minimizing the exact total computational FLOPs through optimal layer partitioning and sparsity assignment:

$$\{\{\mathcal{L}_i^*\}_{i=1}^{G^*}, \{s_i^*\}_{i=1}^{G^*}\} = \arg \min_{\{\mathcal{L}_i\}_{i=1}^{G^*}, \{s_i\}_{i=1}^{G^*}} \left\{ \sum_{i=1}^{G^*} \sum_{l \in \mathcal{L}_i} \phi_l \times (1 - s_i) \right\}, \tag{A11}$$

where $\{\mathcal{L}_i\}_{i=1}^{G^*}$ represents the layer allocation with each $\mathcal{L}_i$ containing consecutive layers, and $\{s_i\}_{i=1}^{G^*}$ denotes the sparsity rates. The constraints ensure $\sum_{i=1}^{G^*} \xi(\mathcal{L}_i, s_i) \leq \delta_{\max}$, $s_i \in [I_i^{\min}, I_i^{\max}]$, $\mathcal{L}_i$ are consecutive, $\bigcup_{i=1}^{G^*} \mathcal{L}_i = \{1, 2, \ldots, p\}$, and $\mathcal{L}_i \cap \mathcal{L}_j = \emptyset$ for $i \neq j$.

The dynamic programming state formulation defines $\text{DP}_{\text{fine}}[i][g][b]$ as the minimum total FLOP cost for optimally partitioning layers $[i..p]$ into exactly g consecutive groups with remaining accuracy degradation budget b:

$$\text{DP}_{\text{fine}}[i][g][b] = \min_{\{\mathcal{L}_k\}_{k=1}^{g}, \{s_k\}_{k=1}^{g}} \left\{ \sum_{k=1}^{g} \sum_{l \in \mathcal{L}_k} \phi_l \times (1 - s_k) \right\}, \tag{A12}$$

where the state covers layers from position i to p, requires exactly g groups, has remaining budget b, and ensures $\bigcup_{k=1}^{g} \mathcal{L}_k = \{i, i+1, \ldots, p\}$ with each $\mathcal{L}_k$ consecutive, $s_k \in [I_k^{\min}, I_k^{\max}]$, and $\sum_{k=1}^{g} \xi(\mathcal{L}_k, s_k) \leq b$.

Algorithm 4 Fine-grained Optimization

Require: G^* groups, intervals $\{[I_i^{\min}, I_i^{\max}]\}$, tabulation tables H, Ξ
Ensure: Optimal allocation $\{\mathcal{L}_i^*\}$ and sparsity rates $\{s_i^*\}$

Initialize $\text{DP}_{\text{fine}}[1..p+1][0..G^*][0..\delta_{\max}/\Delta] \leftarrow \infty$
Initialize $\text{choice}[1..p+1][0..G^*][0..\delta_{\max}/\Delta] \leftarrow \emptyset$
Set boundary condition: $\text{DP}_{\text{fine}}[p+1][0][b] \leftarrow 0$ for all $b \geq 0$
Backward DP Construction:
for $i = p$ **down to** 1 **do**
  for $g = 1$ **to** $\min(G^*, p - i + 1)$ **do**
    for $b = 0$ **to** $\delta_{\max}/\Delta$ **do**
      for $j = i$ **to** $p - g + 1$ **do**
        for $s \in \text{Discretize}([I_{G^*-g+1}^{\min}, I_{G^*-g+1}^{\max}])$ **do**
          $\text{group_cost}, \text{accuracy_cost} \leftarrow H[i][j-i+1][s], \Xi[i][j-i+1][s]/\Delta$
          if $b - \text{accuracy_cost} \geq 0$ **then**
            $\text{total_cost} \leftarrow \text{group_cost} + \text{DP}_{\text{fine}}[j+1][g-1][b - \text{accuracy_cost}]$
            if $\text{total_cost} < \text{DP}_{\text{fine}}[i][g][b]$ **then**
              $\text{DP}_{\text{fine}}[i][g][b] \leftarrow \text{total_cost}$
              $\text{choice}[i][g][b] \leftarrow (j, s)$
            end if
          end if
        end for
      end for
    end for
  end for
end for
Solution Reconstruction:
$\{\mathcal{L}_i^*, s_i^*\} \leftarrow \text{BacktrackSolution}(\text{choice}, 1, G^*, \delta_{\max}/\Delta)$
return $\{\mathcal{L}_i^*\}_{i=1}^{G^*}, \{s_i^*\}_{i=1}^{G^*}$

The state transition equation jointly enumerates all possible first-group formations and sparsity assignments within the corresponding refined interval. The transition first identifies the optimal group

end position j and sparsity rate s:

$$j^*, s^* = \arg \min_{\substack{j \in [i, p-g+1] \\ s \in [I^{\min}_{G^*-g+1}, I^{\max}_{G^*-g+1}]}} \left\{ H[i][j-i+1][s] + \Xi[i][j-i+1][s] \right\}. \tag{A13}$$

Then the state transition equation becomes:

$$\text{DP}_{\text{fine}}[i][g][b] = H[i][j^* - i + 1][s^*] + \text{DP}_{\text{fine}}[j^* + 1][g-1][b - \Xi[i][j^* - i + 1][s^*]], \tag{A14}$$

where j defines the end of the first group (layers i to j), s is the sparsity rate from the refined interval for this group position, $H[i][j-i+1][s]$ provides the exact FLOP cost from tabulation, and $\Xi[i][j-i+1][s]$ gives the exact accuracy degradation cost.

A.9.2 Algorithmic Analysis

The fine-grained optimization employs three-dimensional dynamic programming where $\text{DP}_{\text{fine}}[i][g][b]$ represents the minimum cost for partitioning layers $[i..p]$ into exactly g consecutive groups with remaining budget b. The backward construction enables locally optimal decisions while preserving optimality on the discretized search space through optimal substructure. The choice table records specific decisions for efficient solution reconstruction.

The boundary condition $\text{DP}_{\text{fine}}[p+1][0][b] = 0$ provides the foundation for backward construction. The backward process begins from the final layer and works toward the initial layer, systematically considering all possible first-group formations in each subproblem.

The nested loops systematically enumerate all feasible configurations. The constraint $g \leq \min(G^*, p - i + 1)$ prevents creating more groups than specified or available. The group end position $j \leq p - g + 1$ ensures sufficient layers remain for subsequent groups. The sparsity rate s is constrained to refined intervals $[I^{\min}_{G^*-g+1}, I^{\max}_{G^*-g+1}]$, significantly reducing search space compared to $[\alpha, \beta]$.

Tabulation lookups retrieve precomputed values: $H[i][j-i+1][s]$ for FLOP costs and $\Xi[i][j-i+1][s]/\Delta$ for accuracy degradation. The feasibility check $b - \text{accuracy_cost} \geq 0$ maintains constraint satisfaction. Cost updates occur only when improvements are found, maintaining optimality while avoiding redundant computations.

The solution reconstruction traverses the choice table from initial state $(1, G^*, \delta_{\max}/\Delta)$ and follows recorded decisions to construct the complete solution. Each choice (j, s) specifies group boundaries and sparsity rates, continuing until all groups are identified.

State Transition Intuition. The fine-grained optimization addresses the question: "To optimally partition layers $[i..p]$ into exactly g consecutive groups with remaining budget b, which layers should form the first group and what sparsity rate should be applied?" The backward construction ensures that when deciding the first group boundary j and sparsity s, all subsequent decisions from layer $j+1$ onward are already optimal. This decomposition transforms the complex joint optimization of layer allocation and sparsity assignment into a sequence of local decisions that remain optimal on the discretized search space.

A.9.3 Output of the Fine-Grained Pass

Upon completion, the fine-grained optimization generates the complete optimal solution: precise consecutive layer allocation $\{\mathcal{L}_i^*\}_{i=1}^{G^*}$ and optimal sparsity rates $\{s_i^*\}_{i=1}^{G^*}$ that minimize computational FLOPs. Each $\mathcal{L}_i^* = \{l_{\text{start}}^{(i)}, l_{\text{start}}^{(i)} + 1, \ldots, l_{\text{end}}^{(i)}\}$ defines consecutive layers satisfying continuity and completeness constraints, while $s_i^* \in [I_i^{\min}, I_i^{\max}] \subseteq [\alpha, \beta]$ ensures adherence to the refined sparsity intervals.

Appendix A.10: Tabulation

The tabulation mechanism forms the computational backbone of the ASAF framework by providing efficient access to FLOP costs and accuracy degradation metrics for arbitrary consecutive layer

sequences and sparsity rate combinations. This precomputation strategy transforms the dynamic programming algorithms from computationally prohibitive procedures into tractable optimization methods suitable for large-scale model deployment.

A.10.1 Mathematical Formulation

The tabulation mechanism constructs two three-dimensional tables that provide $O(1)$ lookup for any layer sequence and sparsity combination:

$$H[\text{start}][\text{length}][s] = \sum_{l=\text{start}}^{\text{start}+\text{length}-1} \phi_l \times (1 - s), \tag{A15}$$

The second table is indexed by the same triple and is filled during the same precomputation pass:

$$\Xi[\text{start}][\text{length}][s] = \xi(\{\text{start}, \ldots, \text{start} + \text{length} - 1\}, s), \tag{A16}$$

where H stores FLOP costs and Ξ stores accuracy degradation costs. Here, $\text{start} \in [1, p]$ denotes the starting layer index, $\text{length} \in [1, p - \text{start} + 1]$ represents the number of consecutive layers, $s \in S_{\text{discrete}}$ is the discretized sparsity rate with resolution $\Delta = 0.5\%$, ϕ_l is the original computational cost of layer l, and $\xi(\cdot)$ quantifies accuracy degradation.

Specifically, the accuracy degradation function $\xi(\mathcal{L}, s)$ is defined based on perplexity increase measured on a fixed 640-sample subset of the WikiText-2 validation set:

$$\xi(\mathcal{L}, s) = \text{PPL}(\mathcal{M}_{\mathcal{L},s}) - \text{PPL}(\mathcal{M}_{\text{baseline}}), \tag{A17}$$

where $\mathcal{M}_{\text{baseline}}$ denotes the quantized model without sparsification, and $\mathcal{M}_{\mathcal{L},s}$ denotes the model with sparsity rate s applied to layers in $\mathcal{L}$ while all other layers remain unchanged.

The discrete sparsity set is defined as:

$$S_{\text{discrete}} = \{\alpha + k\Delta : k \in \mathbb{N}, \alpha + k\Delta \leq \beta\}. \tag{A18}$$

This discretization partitions the continuous sparsity range $[\alpha, \beta]$ into uniformly spaced discrete points, enabling efficient tabulation while maintaining sufficient precision for optimization purposes.

Algorithm 5 Tabulation Construction

Require: Model layers $\{1, 2, \ldots, p\}$, sparsity discretization $\Delta = 0.5\%$
Ensure: Tabulation tables H, Ξ

```
Initialize H[1..p][1..p][|S_discrete|] ← 0
Initialize Ξ[1..p][1..p][|S_discrete|] ← 0
S_discrete ← {α + kΔ : k ∈ ℕ, α + kΔ ≤ β}
Systematic Pre-computation:
for start = 1 to p do
    for length = 1 to p − start + 1 do
        L_seq ← {start, start + 1, ..., start + length − 1}
        for each sparsity s ∈ S_discrete do
            H[start][length][s] ← Σ_{l∈L_seq} φ_l × (1 − s)
            Ξ[start][length][s] ← ξ(L_seq, s)
        end for
    end for
end for
return H, Ξ
```

A.10.2 Algorithmic Analysis

The tabulation construction systematically precomputes FLOP costs and accuracy degradation values for all consecutive layer sequences and discretized sparsity rates. The three-dimensional tables H and Ξ enable constant-time lookup during optimization phases, eliminating redundant calculations and concentrating computational burden in preprocessing.

The discrete sparsity set S_{discrete} balances computational efficiency with optimization precision. The systematic precomputation exhaustively evaluates all valid combinations of starting positions, sequence lengths, and sparsity rates. The constraint length $\leq p - \text{start} + 1$ maintains validity by preventing sequences extending beyond available layers.

The FLOP cost calculation $H[\text{start}][\text{length}][s] = \sum_{l \in \mathcal{L}_{\text{seq}}} \phi_l \times (1 - s)$ represents total computational cost when applying sparsity rate s. The factor $(1 - s)$ reflects computational reduction from sparsification, while summation over ϕ_l accounts for heterogeneous layer costs.

The accuracy degradation calculation $\Xi[\text{start}][\text{length}][s] = \xi(\mathcal{L}_{\text{seq}}, s)$ quantifies performance impact. The sensitivity function $\xi(\cdot)$ requires careful design balancing accuracy with computational tractability. Gradient-based methods provide theoretical foundations but require significant resources, while activation-based methods offer efficiency at potential accuracy cost.

The tabulation tables enable the transformation of dynamic programming state transitions from $O(\text{group_size})$ computations to $O(1)$ operations. Without tabulation, each state transition would require recomputation of FLOP costs and accuracy degradation values, leading to prohibitive computational complexity for large models.

A.10.3 Tabulation Storage Requirements

Storage Specifications. The tabulation tables H and Ξ require $\mathcal{O}(L^2 \times |S_{\text{discrete}}|)$ storage each. For Llama-2-7B with $L = 32$ layers and $|S_{\text{discrete}}| = 29$ sparsity levels, each table requires approximately 116 KB in single precision ($32 \times 32 \times 29$ = 29,696 entries $\times$ 4 bytes/entry $\approx$ 116 KB), with both tables totaling approximately 232 KB. The preprocessing computation is $\mathcal{O}(L^2 \times |S_{\text{discrete}}| \times C_\xi)$ where C_ξ represents the cost of evaluating the sensitivity function $\xi(\cdot)$.

Reusability. Tables can be computed once per model and reused across multiple optimization scenarios. The constant-time lookup capability enables efficient exploration of large solution spaces.

Appendix A.11: Complete Hyperparameter Specifications

A.11.1 ASAF Framework Configuration

Dynamic Programming Parameters. Our ASAF framework explores sparsity allocations within the constrained range where $\alpha = 1\%$ represents the minimum sparsity threshold and $\beta = 15\%$ defines the maximum sparsity level, consistent with the low-sparsity regime adopted in recent layer-wise LLM pruning work (Yin et al., 2024). The maximum allowable accuracy degradation is set to $\delta_{\max} = 1\%$ to ensure practical deployment viability. Tabulation sampling resolution is configured as $\Delta = 0.5\%$, providing sufficient granularity for optimization while maintaining computational tractability. The dynamic programming state space discretizes the accuracy degradation budget with resolution $\Delta = 0.5\%$, with maximum group exploration set to the total layer count to allow full granularity in layer allocation decisions.

Tabulation Construction. The sensitivity measurement employs 640 calibration samples to ensure statistical robustness during tabulation construction (Frantar & Alistarh, 2023). Tabulation cache allocation is adapted to the available GPU memory on NVIDIA RTX 3090 GPUs while providing sufficient storage for memoization. All tabulation computations utilize FP32 precision accumulation to maintain numerical stability throughout the dynamic programming process, preventing precision degradation that could affect optimization quality (Jacob et al., 2018). A single seed, fixed at 0, selects the calibration windows used both for the sensitivity tabulation and for the GPTQ calibration set, and the same seed enters the key under which a tabulation is cached, so a repeated run reuses the same windows and the same tabulated sensitivities.

A.11.2 Quantization Configuration

Weight Quantization Parameters. Weight quantization employs 4-bit precision for all linear layers, with per-channel symmetric quantization applied to preserve fine-grained statistics (Frantar

et al., 2022). The quantization scale s for each channel c is computed as:

$$s_c = \frac{\max(|W_c|) \cdot \text{clip_ratio}}{2^{\text{bits}-1} - 1}, \tag{A19}$$

where clip_ratio $= 0.9$ controls the quantization range to handle outliers. For weight quantization, we employ both round-to-nearest (RTN) method (Jacob et al., 2018) and GPTQ (Frantar et al., 2022) approaches. We use per-channel symmetric quantization (Jacob et al., 2018) with group size 128 to balance quantization quality and computational efficiency. During GPTQ quantization, we utilize 128 samples from the WikiText-2 dataset (Merity et al., 2017) with sequence length of 2048 as the calibration set, following established protocols for transformer quantization. The GPTQ solver applies a damping factor of 0.01 to the Hessian diagonal. Both calibration sets are drawn in the same way: the split is concatenated, tokenised once, and cut into non-overlapping 2048-token chunks, and the calibration set is a seeded permutation of those chunks truncated to the required sample count.

Activation Quantization Parameters. We apply per-token symmetric quantization to quantize the inputs, where each row of the activation matrix shares a common quantization scale (Xiao et al., 2023). The clipping ratio is fixed at 0.9 across all experiments to maintain consistent quantization behavior. This approach effectively handles the dynamic range of activations while preserving computational efficiency during sparse matrix operations.

KV Cache Quantization. The KV caches are quantized using asymmetric quantization (Dettmers et al., 2022), organized into groups of 128 elements to match the head dimension structure of transformer architectures. A constant clipping ratio of 0.95 is applied to accommodate the typically wider dynamic range of cached key-value pairs compared to standard activations (Ashkboos et al., 2024c).

A.11.3 Evaluation Configuration

Language Generation Tasks. We evaluate on WikiText-2 perplexity using 2048-token sequences with a sliding window stride of 512 (Merity et al., 2017). Throughput evaluation encompasses batch sizes of 1, 4, 16, and 32 to assess scalability across different deployment scenarios. All language generation tasks utilize greedy decoding with temperature set to 0.0, so the reported metrics are deterministic.

Zero-shot Classification. We assess our framework across six established benchmarks: PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021), HellaSwag (Zellers et al., 2019), LAMBADA (Radford et al., 2019), ARC-Easy and ARC-Challenge (Clark et al., 2018). All experiments utilize the LM Evaluation Harness (Gao et al., 2021; 2024) with default configurations. Maximum generation length is limited to 256 tokens with KV cache enabled for efficient inference during evaluation.

A.11.4 Target Matrix Configuration

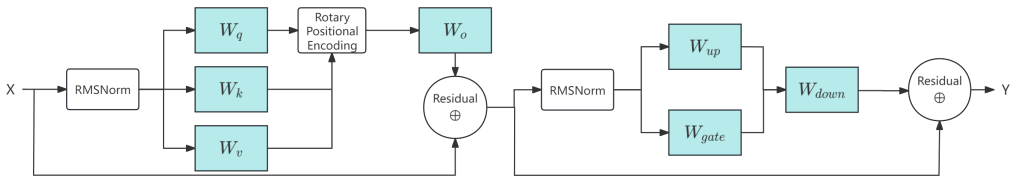


Figure A7: Target weight matrices in transformer layers for ASAF quantization and sparsification.

Figure A7 illustrates the complete transformer layer architecture highlighting the seven target weight matrices that undergo both 4-bit quantization and adaptive sparsity allocation in our ASAF framework. The green boxes represent the linear projection matrices: query (W_q), key (W_k), value (W_v), and output (W_o) projections in the multi-head attention mechanism, as well as up-projection (W_{up}),

gate projection (W_{gate}), and down-projection (W_{down}) matrices in the feed-forward network. These matrices constitute the primary computational bottlenecks in transformer inference and are the focus of our layer-group-based optimization strategy, while components like RMSNorm and positional encoding remain unmodified to preserve numerical stability.

Matrix-Level Configuration. Our ASAF framework applies unified quantization and adaptive sparsity allocation to all linear projection matrices within each transformer layer, as illustrated in Figure A7. The attention mechanism matrices include query projection $W_q \in \mathbb{R}^{d_{model} \times d_{head}}$, key projection $W_k \in \mathbb{R}^{d_{model} \times d_{head}}$, value projection $W_v \in \mathbb{R}^{d_{model} \times d_{head}}$, and output projection $W_o \in \mathbb{R}^{d_{head} \times d_{model}}$ (Touvron et al., 2023). The feed-forward network matrices consist of up-projection $W_{up} \in \mathbb{R}^{d_{model} \times d_{ff}}$, gate projection $W_{gate} \in \mathbb{R}^{d_{model} \times d_{ff}}$, and down-projection $W_{down} \in \mathbb{R}^{d_{ff} \times d_{model}}$ where $d_{ff} = 4 \times d_{model}$ following standard transformer scaling (Touvron et al., 2023).

Layer-Group Sparsity Application. Each target matrix within a layer group receives identical sparsity allocation determined by our dynamic programming optimization. For layer group $\mathcal{L}_i$ with optimized sparsity rate s_i^*, all seven projection matrices (W_q, W_k, W_v, W_o, W_{up}, W_{gate}, W_{down}) undergo magnitude-based pruning at rate s_i^* (Han et al., 2015). This uniform application ensures consistent computational benefits across all matrix operations within each layer while preserving the structural integrity of the transformer architecture. The RMSNorm parameters and positional encoding remain unmodified to maintain numerical stability during inference (Zhang & Sennrich, 2019).

Quantization Integration. Following the rotation-based quantization pipeline (Ashkboos et al., 2024c), each target matrix first undergoes 4-bit GPTQ quantization (Frantar et al., 2022) before sparsity application. The quantization process preserves the relative magnitude relationships critical for effective pruning, while the subsequent sparsification leverages the quantized weight distributions for optimal efficiency. This sequential approach ensures compatibility between the compression techniques while maximizing hardware acceleration potential on modern GPU architectures.

A.11.5 Software Environment

Runtime and Library Versions. Model weights are loaded in float16 onto a single device, and the interpreter floor is Python 3.9. The run environment uses CUDA 12.1 together with torch 2.11.0, transformers 5.14.1, tokenizers 0.23.1, accelerate 1.14.0, datasets 5.0.0, numpy 2.4.6, lm-eval 0.4.2, and fast-hadamard-transform 1.0.4.post1.

Appendix A.12: Accuracy Results

Table A2: WikiText-2 perplexity comparison for Llama-2 models (2048 sequence length) using 4-bit quantization. SmoothQuant and OmniQuant results are from (Shao et al., 2024), and 128G indicates group-wise quantization with 128 group size. Our ASAF framework applies layer-group-based pruning, with 4-bit precision across weights, activations, and KV caches. Lower perplexity indicates better performance.

Method	Weight Quantization	#Outlier Features	Llama-2 7B	13B	30B	70B
Baseline	-	-	5.47	4.88	4.09	3.32
SmoothQuant (Xiao et al., 2023)	RTN	0	83.12	35.88	-	-
OmniQuant (Shao et al., 2024)	RTN	0	14.26	12.30	-	-
QUIK-4B (Ashkboos et al., 2024b)	GPTQ	256	8.87	7.78	7.28	6.91
QuaRot	GPTQ	0	**6.10**	**5.40**	**4.41**	**3.79**
ASAF (Ours)	GPTQ	0	6.14	5.44	4.44	3.82
Atom-128G (Zhao et al., 2024)	GPTQ-128G	128	6.03	**5.26**	-	-
QuaRot-128G		0	**5.93**	**5.26**	**4.25**	**3.61**
ASAF-128G (Ours)		0	5.98	5.30	4.28	3.64

Language Generation Tasks. We evaluate our ASAF framework on the WikiText-2 language-generation benchmark. Table A2 reports the perplexity after quantizing Llama-2 weights to 4 bits with GPTQ and applying our adaptive sparsity allocation across layer groups. Our framework demonstrates competitive performance compared to state-of-the-art quantization methods, achieving perplexity degradation less than 1% compared to QuaRot while providing additional computational benefits through optimized sparsity allocation. The layer-group-based pruning approach requires no additional outlier storage and no asymmetric weight quantization. When using group-size-128 quantization, ASAF maintains comparable performance with perplexity increases within 1% of QuaRot-128G while enabling more efficient inference through adaptive sparsity patterns.

Table A3: Zero-shot accuracy of Llama models with our ASAF framework on PIQA (PQ), WinoGrande (WG), HellaSwag (HS), ARC-Easy (A-e), ARC-Challenge (A-c), and LAMBADA (LA). Our method applies adaptive sparsity allocation across layer groups.

Model	Method	PQ ↑	WG ↑	HS ↑	A-e ↑	A-c ↑	LA ↑	Average ↑
Llama-2-7B	FP16	79.11	69.06	75.99	74.58	46.25	73.90	69.82
	QuaRot	76.77	63.77	72.16	69.87	40.87	70.39	65.64
	ASAF (Ours)	76.00	63.23	71.47	69.17	40.52	69.69	65.01
Llama-2-13B	FP16	80.47	72.22	79.39	77.48	49.23	76.75	72.59
	QuaRot	78.89	70.24	76.37	72.98	46.59	73.67	69.79
	ASAF (Ours)	78.26	69.68	75.76	72.25	46.19	72.97	69.19
Llama2-30B	FP16	81.13	73.94	80.72	78.52	51.65	77.59	73.93
	QuaRot	79.94	72.03	77.99	75.20	49.46	75.18	71.63
	ASAF (Ours)	79.14	71.42	77.37	74.60	48.99	74.58	71.02
Llama-2-70B	FP16	82.70	77.98	83.84	80.98	57.34	79.58	77.07
	QuaRot	82.43	76.24	81.82	80.43	56.23	78.73	75.98
	ASAF (Ours)	81.77	75.48	81.12	79.71	55.81	77.98	75.31

Zero-Shot Tasks. We assess ASAF across six established zero-shot benchmarks: PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021), HellaSwag (Zellers et al., 2019), LAMBADA (Radford et al., 2019), and ARC-Easy and ARC-Challenge (Clark et al., 2018). Experiments utilize the LM Evaluation Harness (Gao et al., 2021; 2024) with default configurations. Table A3 shows that ASAF maintains strong performance across all Llama-2 model sizes, with average degradation consistently below 1% compared to QuaRot. ASAF preserves model capabilities while enabling computational efficiency gains through optimized layer-group pruning patterns.

Prefill Stage Performance Increases. Figure A8 demonstrates the acceleration performance of ASAF across various batch configurations (1, 4, 16, and 32) with 2048-token sequences on Llama-2 models. Our adaptive sparsity allocation approach consistently outperforms the QuaRot baseline implementation across all tested configurations. The performance gains become more pronounced with larger batch sizes, as the computational workload increasingly overshadows memory bandwidth limitations. For the largest 70B model, our method reaches peak acceleration of 3.63×.

Memory and Computational Efficiency. The algorithmic efficiency of ASAF directly translates into substantial memory savings across both model families and sequence lengths. Table A4 expands our evaluation beyond the Llama-2 series to include the more recent Llama-3 models and reports results under three context lengths: 512, 2048, and 4096 tokens. This broader evaluation shows that ASAF maintains stable compression ratios across architectures and sequence lengths, achieving 7 to 15% memory reduction depending on model scale, and confirming that its sensitivity-aware sparsity allocation generalizes beyond a single model generation. Furthermore, the multi-length analysis addresses prefill-related concerns: ASAF consistently reduces memory usage even at short (512) and long (4096) contexts, providing favorable conditions for high-throughput prefill optimization.

RTN Quantization Strategy. To evaluate the robustness of our adaptive sparsity allocation approach, we compare ASAF's performance under RTN quantization, where GPTQ serves as the default weight quantization strategy. Table A5 demonstrates that at 8-bit precision, RTN maintains accuracy nearly identical to full precision for both approaches. At 4-bit quantization, while both

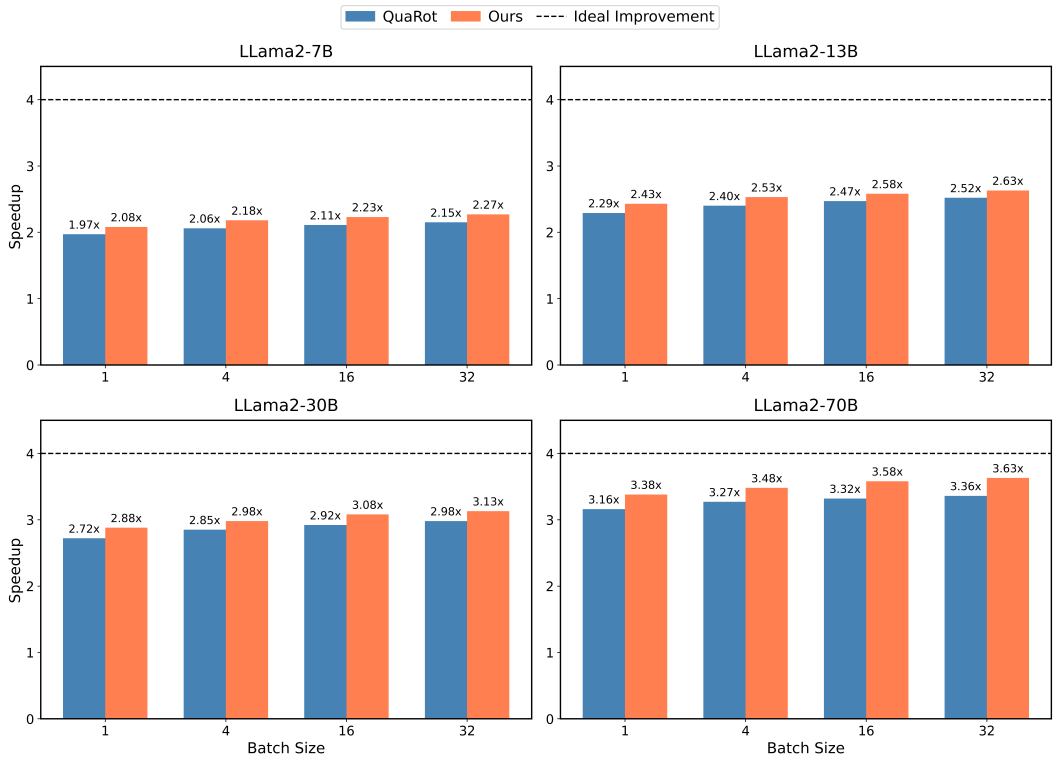


Figure A8: Performance comparison between our framework and QuaRot on Llama-2 models, evaluated on NVIDIA RTX 3090 GPUs with 2048-token sequences across various batch sizes.

Table A4: GPU memory consumption (MB) of QuaRot and ASAF across Llama-2 and Llama-3 models with varying sequence lengths. Compression Ratio is the memory reduction of ASAF relative to QuaRot, in percent.

Model	**Method**	**Sequence Length**		
		512	**2048**	**4096**
Llama-2-7B	QuaRot	1,019	3,255	6,518
	ASAF	948	3,013	6,009
	Compression Ratio	6.97%	7.43%	7.81%
Llama-2-13B	QuaRot	1,849	5,753	11,486
	ASAF	1,688	5,276	10,502
	Compression Ratio	8.71%	8.29%	8.57%
Llama-2-30B	QuaRot	3,573	11,408	22,814
	ASAF	3,127	10,110	20,148
	Compression Ratio	12.48%	11.38%	11.69%
Llama-2-70B	QuaRot	6,521	20,536	41,079
	ASAF	5,653	17,943	35,833
	Compression Ratio	13.31%	12.63%	12.77%
Llama-3-8B	QuaRot	1,226	3,722	7,456
	ASAF	1,127	3,432	6,869
	Compression Ratio	8.08%	7.79%	7.87%
Llama-3-70B	QuaRot	6,811	21,295	42,586
	ASAF	5,855	18,136	36,241
	Compression Ratio	14.04%	14.83%	14.90%

Table A5: WikiText-2 perplexity (PPL) and zero-shot accuracy of our ASAF framework for Llama-2 models applying 4- and 8-bits with RTN weights and activation quantization. We use PIQA (PQ), WinoGrande (WG), HellaSwag (HS), ARC-Easy (A-e), ARC-Challenge (A-c), and LAMBADA (LA). We quantize all weights, activations, and caches and introduce adaptive sparsity allocation across layer groups.

Model	Method	Precision	PPL ↓	PQ ↑	WG ↑	HS ↑	A-e ↑	A-c ↑	LA ↑	Average ↑
7B	Baseline	FP16	5.47	79.11	69.06	75.99	74.58	46.25	73.90	69.82
	QuaRot-RTN	INT4	8.37	72.09	60.69	65.40	58.88	35.24	57.27	58.26
		INT8	5.50	78.94	68.67	75.80	74.79	45.39	74.33	69.65
	ASAF-RTN (Ours)	INT4	8.44	71.48	60.23	64.81	58.38	34.98	56.75	57.77
		INT8	5.54	78.31	68.09	75.12	74.19	45.00	73.66	69.06
13B	Baseline	FP16	4.88	80.47	72.22	79.39	77.48	49.23	76.75	72.59
	QuaRot-RTN	INT4	6.09	77.37	67.32	73.11	70.83	43.69	70.66	67.16
		INT8	4.90	80.52	71.59	79.38	77.31	49.15	76.79	72.46
	ASAF-RTN (Ours)	INT4	6.14	76.67	66.71	72.45	70.19	43.30	70.02	66.56
		INT8	4.94	79.84	70.91	78.67	76.65	48.68	76.10	71.81
70B	Baseline	FP16	3.32	82.70	77.98	83.84	80.98	57.34	79.58	77.07
	QuaRot-RTN	INT4	4.14	80.69	75.14	79.63	77.57	51.71	77.02	73.63
		INT8	3.33	82.97	77.98	83.67	80.77	58.11	79.53	77.17
	ASAF-RTN (Ours)	INT4	4.18	79.92	74.46	78.83	76.83	51.24	76.25	72.92
		INT8	3.36	82.14	77.24	82.92	79.96	57.56	78.81	76.44

methods experience some quality degradation, our ASAF framework consistently maintains average performance within 1% of QuaRot results across all model sizes. In both INT4 and INT8 configurations, these findings confirm that layer-group-based adaptive sparsity allocation can be effectively combined with RTN quantization without introducing significant accuracy loss, validating the generalizability of our optimization framework.

APPENDIX A.13: COMPUTATIONAL EFFICIENCY ANALYSIS

A.13.1 UNIFIED COMPLEXITY ANALYSIS

Baseline Brute-Force Complexity. The naive approach requires exhaustive enumeration of all possible layer grouping and sparsity allocation configurations. For a model with L layers and $|S|$ discretized sparsity rates, there are 2^{L-1} ways to partition the layers into consecutive groups and $|S|^G$ sparsity assignments for G groups, where G ranges from 1 to L. The total brute-force complexity is:

$$\mathcal{C}_{\text{brute}} = \sum_{G=1}^{L} \binom{L-1}{G-1} \times |S|^G = |S|\,(1+|S|)^{L-1} = \mathcal{O}(|S|^L \times 2^L) \tag{A20}$$

For Llama-2-7B with $L = 32$ layers and $|S| = 29$ sparsity levels (1% to 15% with 0.5% steps), brute force requires at most $29^{32} \times 2^{31} \approx 10^{56}$ operations.

ASAF Framework Complexity. Our two-phase approach decomposes the problem into manageable subproblems. The coarse-grained phase (Phase 1) has complexity $\mathcal{O}(G_{\max} \times B \times |\mathcal{I}|)$, where $G_{\max} = L$ is the maximum number of groups, B is the number of discretized accuracy budget levels, determined by the granularity of sensitivity measurements, and $|\mathcal{I}|$ is the number of candidate sparsity intervals. The fine-grained phase (Phase 2) has complexity $\mathcal{O}(L^2 \times G^* \times B \times |S'|)$, where L^2 accounts for the nested loops over layer positions and group boundaries, G^* is the optimal number of groups from Phase 1, and $|S'| \ll |S|$ is the size of the refined sparsity space. Tabulation preprocessing has complexity $\mathcal{O}(L^2 \times |S| \times T_{\text{eval}})$, where T_{eval} is the evaluation time for sensitivity measurement per configuration.

Total ASAF Complexity. Summing the three terms gives

$$\mathcal{C}_{\text{ASAF}} = \mathcal{O}(L^2 \times |S| \times T_{\text{eval}} + G_{\max} \times B \times |\mathcal{I}| + L^2 \times G^* \times B \times |S'|) \tag{A21}$$

whose dominant term is the tabulation preprocessing, $\mathcal{O}(L^2 \times |S| \times T_{\text{eval}})$.

Practical Complexity Comparison. For the Llama-2-7B configuration, $L = 32$, $|S| = 29$, $G_{\max} = 32$, $|\mathcal{I}| = 15$, $G^* \approx 8$ and $|S'| \approx 3$. The accuracy budget discretization is $B = \lceil \delta_{\max}/\Delta \rceil = \lceil 1\%/0.5\% \rceil = 2$ levels for the experiments in this paper. In general, B scales with the granularity of accuracy degradation tracking: finer-grained sensitivity measurements may require $B \in O(10)$ to $O(100)$ levels to capture detailed degradation patterns, though our experiments show that $B = 2$ is sufficient for the $\delta_{\max} = 1\%$ constraint. The ASAF total is then $32^2 \times 29 \times T_{\text{eval}} + O(G_{\max} \times B \times |\mathcal{I}|) + O(L^2 \times G^* \times B \times |S'|) \approx 3 \times 10^4 \times T_{\text{eval}} + O(10^3)$ operations with $B = 2$, and the ratio $\mathcal{C}_{\text{brute}}/\mathcal{C}_{\text{ASAF}} \approx 10^{52}/T_{\text{eval}}$ shows an exponential reduction in complexity.

Memory Complexity. The dynamic programming tables require $\mathcal{O}(G_{\max} \times B)$ for the coarse-grained phase and $\mathcal{O}(L \times G^* \times B)$ for the fine-grained phase, and the tabulation requires $\mathcal{O}(L^2 \times |S|)$ for each of the FLOP and sensitivity tables. The total memory is therefore

$$\text{Memory}_{\text{ASAF}} = \mathcal{O}(L^2 \times |S| + L \times G^* \times B) \tag{A22}$$

For Llama-2-7B, the tabulation tables dominate memory usage with ∼232 KB for storing precomputed FLOP costs and sensitivity measurements, while DP state tables contribute additional tens of KB. Total overhead remains under 300 KB, which is negligible (less than 0.01%) compared to model parameters.

Scalability Analysis. The polynomial complexity ensures graceful scaling: a 12-layer model requires 10^{21} brute-force operations against 10^4 for ASAF, and a 48-layer model 10^{85} against 10^5, so the complexity reduction ratio grows exponentially with model size.

A.13.2 Memory Efficiency Analysis

Dynamic Programming State Management. The memory requirements for our dynamic programming approach consist of three primary components. The coarse-grained and fine-grained DP tables maintain states for tracking optimal solutions across different grouping configurations and accuracy budgets. The memory requirements scale with model size and accuracy budget discretization, typically consuming tens of KB for both DP tables in FP32 format. The tabulation mechanism maintains two lookup tables: the FLOP cost table $H[i][\text{len}][s]$ and the accuracy degradation table $\Xi[i][\text{len}][s]$, both with dimensions $[L] \times [L] \times [|S|]$. The combined memory requirement becomes $2 \times L \times L \times |S|$ entries, specifically resulting in $2 \times 32 \times 32 \times 29 = 59,392$ entries for Llama-2-7B, consuming approximately 232.0 KB in FP32 format (59,392 entries × 4 bytes/entry). Combined with the DP state tables, the total memory overhead for all data structures amounts to approximately 250-300 KB, which remains negligible (less than 0.01%) compared to the model parameter memory requirements of several gigabytes.

Model Memory Reduction. The adaptive sparsity allocation strategy achieves memory efficiency through strategic pruning of redundant parameters. For a layer group i with sparsity rate s_i, the memory reduction is directly proportional: $\text{Memory}_{\text{saved}} = s_i \times \text{Memory}_{\text{original}}$. Our experimental results demonstrate progressive memory reduction scaling with model size. The Llama-2-7B model achieves 7.43% memory reduction (from 3,255 MB to 3,013 MB at sequence length 2048), while the Llama-2-13B model attains 8.29% reduction (from 5,753 MB to 5,276 MB) at an average sparsity of 7.9%. Larger models exhibit more substantial benefits, with the Llama-2-30B achieving 11.38% reduction (from 11,408 MB to 10,110 MB) and the Llama-2-70B reaching 12.63% reduction (from 20,536 MB to 17,943 MB). This scaling behavior reflects the increased redundancy present in larger architectures, enabling more aggressive sparsification without accuracy degradation.

A.13.3 Runtime Performance Characteristics

Preprocessing and Optimization Time. The tabulation construction phase constitutes the primary preprocessing overhead, with time complexity $\mathcal{O}(L^2 \times |S| \times T_{\text{eval}})$, where T_{eval} represents the evaluation time for a single configuration. The preprocessing time scales quadratically with the number of layers but remains acceptable for deployment scenarios requiring one-time optimization.

The dynamic programming phases then operate on the precomputed tables through $O(1)$ lookups, so tabulation dominates the preprocessing cost. This preprocessing cost is amortized across all subsequent inference runs, making it negligible compared to the cumulative inference-time savings enabled by ASAF.

Inference Acceleration Analysis. The FLOP reduction achieved by optimal allocation $\{(\mathcal{L}_i, s_i)\}_{i=1}^{G^*}$ can be quantified as:

$$\text{FLOP}_{\text{reduced}} = \sum_{i=1}^{G^*} \sum_{l \in \mathcal{L}_i} \phi_l \times s_i, \tag{A23}$$

where ϕ_l represents the original FLOP count for layer l. Experimental evaluation across different batch sizes and model configurations reveals consistent acceleration patterns. The Llama-2-7B model achieves speedups ranging from 2.08× at batch size 1 to 2.27× at batch size 32 during the prefill stage. Larger models demonstrate more substantial improvements, with the Llama-2-13B reaching 2.63× at batch size 32, the Llama-2-30B achieving 3.13×, and the Llama-2-70B reaching peak acceleration of 3.63× at batch size 32. This scaling behavior indicates that both increasing model complexity and batch size amplify the gains from adaptive allocation, as larger computational workloads increasingly overshadow memory bandwidth limitations.

Where the Speedup Comes From. The allocated sparsity is realized by reducing the GEMM dimensions of each layer group, so ASAF and the quantized baseline execute through an identical kernel stack. The speedups reported above therefore follow from smaller matrix multiplications and the correspondingly reduced weight traffic, not from any change to the kernels themselves.

A.13.4 Scalability Properties

Layer Count Scaling. The polynomial complexity $\mathcal{O}(L^2 \times |S|)$ ensures graceful scaling with increasing model sizes, while brute-force complexity $\mathcal{O}(|S|^L \times 2^L)$ grows exponentially. For a model with L layers and $|S|$ sparsity levels, ASAF requires $\mathcal{O}(L^2 \times |S|)$ operations compared to $\mathcal{O}(|S|^L \times 2^L)$ brute-force operations, with the complexity reduction factor growing exponentially with model depth. For instance, moving from a 32-layer model to a 64-layer model increases ASAF's tabulation cost by a factor of 4 (quadratic scaling), while brute-force cost increases by a factor of about 10^{56} (exponential scaling).

Sparsity Resolution Impact. The impact of sparsity discretization resolution on computational requirements follows a linear relationship. Finer discretization (e.g., from 0.5% to 0.25% steps, doubling $|S|$ from 29 to 57) proportionally increases the tabulation construction time by approximately 2× while slightly increasing the optimization overhead.

Accuracy Budget Sensitivity. The relationship between the final accuracy constraint $\delta_{\max}$ and optimization complexity is influenced by the internal budget discretization granularity B. While the framework maintains a strict $\delta_{\max} = 1\%$ constraint as reported in the main experiments, the internal DP search space uses finer-grained budget tracking to accurately capture cumulative degradation across layer groups. This design ensures that the coarse-grained phase efficiently prunes suboptimal configurations, maintaining overall polynomial complexity. The framework remains computationally tractable even when exploring different accuracy-efficiency trade-off points, with preprocessing time dominated by the $\mathcal{O}(L^2 \times |S| \times T_{\text{eval}})$ tabulation cost rather than the DP search itself.

Appendix A.14: NVIDIA 4090 GPU Performance Analysis

Performance Evaluation on NVIDIA RTX 4090s. We further evaluate our ASAF framework on NVIDIA RTX 4090 GPUs to assess scalability across hardware generations. Figure A9 demonstrates acceleration performance across various batch configurations with 2048-token sequences on Llama-2 models. On RTX 4090, our framework achieves peak acceleration of 3.89× on the LLaMA-2-70B model, representing a 7.2% improvement over the 3.63× achieved on RTX 3090. The relative improvement of ASAF over QuaRot baseline remains consistent across both GPU generations, with our method providing 6 to 8% additional acceleration across all model sizes.

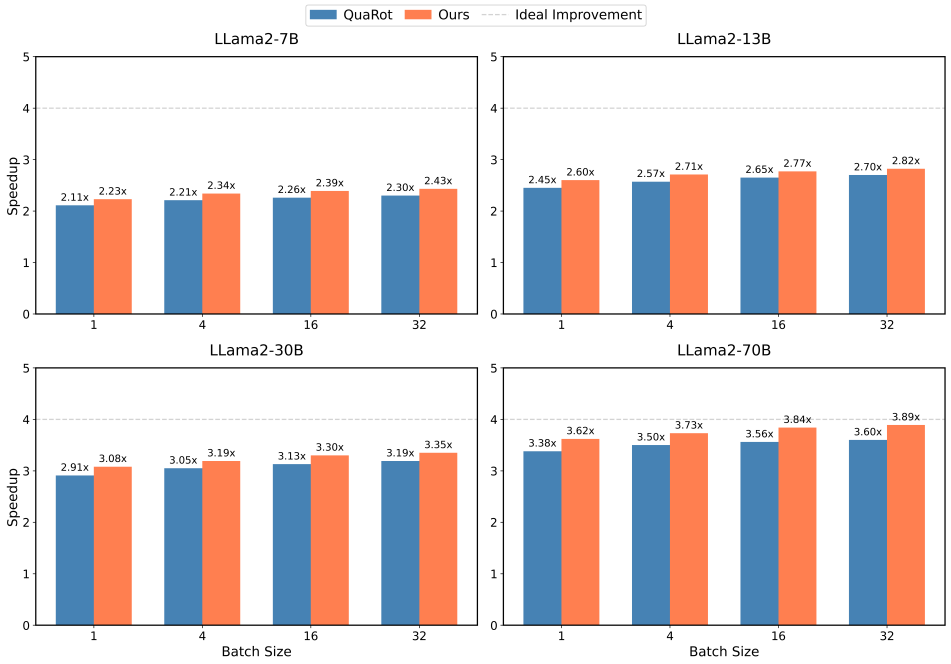


Figure A9: Prefill-stage performance of ASAF on a single transformer block of Llama-2 models using NVIDIA RTX 4090 GPUs. We evaluate using sequence length 2048 with different batch sizes.

APPENDIX A.15: LLAMA-3 EXPERIMENTAL RESULTS

Table A6: Zero-shot accuracy of Llama-3 models with our framework on PIQA (PQ), WinoGrande (WG), HellaSwag (HS), ARC-Easy (A-e), ARC-Challenge (A-c), and LAMBADA (LA).

Model	Method	PQ ↑	WG ↑	HS ↑	A-e ↑	A-c ↑	LA ↑	Average ↑
Llama3-8B	FP16	81.28	72.14	78.32	78.71	52.13	76.84	73.24
	QuaRot	79.16	69.82	75.47	75.33	48.29	73.95	70.34
	ASAF (Ours)	78.52	69.26	74.81	74.73	47.94	73.40	69.78
Llama3-70B	FP16	85.42	81.77	86.15	85.23	64.42	82.91	80.98
	QuaRot	84.28	80.34	84.61	83.76	62.88	81.24	79.52
	ASAF (Ours)	83.65	79.77	83.94	83.05	62.47	80.63	78.92

Zero-Shot Tasks on Llama-3 Family. We repeat the zero-shot evaluation on the Llama-3 family, whose FP16 baseline is stronger than Llama-2's. We evaluate our framework on the same six zero-shot benchmarks: PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021), HellaSwag (Zellers et al., 2019), LAMBADA (Radford et al., 2019), and the ARC-Easy and ARC-Challenge datasets (Clark et al., 2018). Table A6 demonstrates that our method maintains competitive accuracy across Llama-3 model sizes with performance degradation consistently below 1% compared to QuaRot, even on these more advanced models with higher baseline performance.

APPENDIX A.16: COMPARISON WITH NAIVE HEURISTIC EXPLORATION

We compare ASAF with a naive search-time-matched heuristic that uniformly assigns sparsity levels across layer groups, without using sensitivity information. Table A7 reports the comparison. Both methods receive the same search budget, the same tabulation and the same evaluation protocol, so the contrast isolates the allocation strategy itself. Across every coarse group partition, ASAF discovers configurations that satisfy the accuracy budget, whereas the uniform heuristic exceeds the

Table A7: Comparison between ASAF and a naive search-time-matched heuristic on the WikiText-2 validation set under different coarse group partitions. "Groups" denotes the number of layer groups used in the coarse-grained DP stage.

Groups	Method	Accuracy Drop (%)	Feasible ($\leq 1\%$ Budget)
12	ASAF (ours)	0.62	✓
12	Naive heuristic	1.37	×
16	ASAF (ours)	0.74	✓
16	Naive heuristic	1.25	×
20	ASAF (ours)	0.58	✓
20	Naive heuristic	1.09	×

$\leq 1\%$ budget in all of them. These results show that naive uniform exploration is inadequate for this coupled optimization space.

APPENDIX A.17: ABLATION STUDY FOR ACCURACY BUDGET AND SPARSITY RANGE

We conduct explicit ablation studies by systematically varying the accuracy degradation budget δ and the sparsity search range $[\alpha, \beta]$.

A.17.1 SENSITIVITY TO ACCURACY DEGRADATION BUDGET δ

Table A8 reports performance under varying δ values from 0.5% to 2.0% on Llama-2-7B. Since δ serves as a hard constraint in our DP formulation, the actual perplexity degradation must remain strictly below the specified budget. The results demonstrate predictable and monotonic behavior: relaxing the accuracy constraint enables the DP solver to explore higher sparsity configurations, yielding greater acceleration while the actual degradation approaches but never exceeds the budget.

Table A8: Sensitivity to accuracy budget δ on Llama-2-7B (batch=32, seq=2048). All configurations satisfy the hard constraint $\Delta\text{PPL} < \delta$.

δ (%)	PPL	Δ vs QuaRot	Average Sparsity (%)	Speedup	Memory Reduction (%)
0.5	6.12	0.33%	4.2	2.24×	4.51
1.0	**6.14**	**0.66%**	**7.1**	**2.31**×	**7.43**
1.5	6.17	1.15%	8.4	2.38×	8.67
2.0	6.22	1.97%	9.5	2.44×	9.72

A.17.2 CROSS-MODEL CONSISTENCY UNDER VARYING δ

To verify that the observed behavior generalizes beyond a single model scale, we repeat the δ sweep on Llama-2-70B (Table A9). The larger model achieves higher sparsity under identical δ constraints, consistent with our finding that larger architectures contain more redundancy. Crucially, both models exhibit the same structural allocation pattern and monotonic accuracy-efficiency trade-off.

Table A9: Cross-model validation on Llama-2-70B (batch=32, seq=2048). All configurations satisfy the hard constraint $\Delta\text{PPL} < \delta$.

δ (%)	PPL	Δ vs QuaRot	Average Sparsity (%)	Speedup
0.5	3.80	0.26%	7.8	3.51×
1.0	**3.82**	**0.79%**	**12.1**	**3.63**×
1.5	3.84	1.32%	13.6	3.72×
2.0	3.86	1.85%	14.9	3.80×

A.17.3 SENSITIVITY TO SPARSITY RANGE $[\alpha, \beta]$

Table A10 examines the effect of the sparsity search range under fixed $\delta = 1\%$. Expanding the upper bound from 15% to 20% or even 30% yields only marginal improvement, indicating that the accuracy constraint δ is the binding factor rather than the range boundary. The results confirm that the default setting $[1\%, 15\%]$ provides sufficient flexibility for the DP solver to discover near-optimal allocations.

Table A10: Sensitivity to sparsity range on Llama-2-7B ($\delta = 1\%$, batch=32). All configurations satisfy the hard constraint $\Delta\text{PPL} < 1\%$.

Range $[\alpha, \beta]$	PPL	Δ vs QuaRot	Average Sparsity (%)	Speedup
$[1\%, 10\%]$	6.13	0.49%	5.8	2.21×
[1%, 15%]	**6.14**	**0.66%**	**7.1**	**2.31**×
$[1\%, 20\%]$	6.15	0.82%	7.6	2.36×
$[1\%, 30\%]$	6.16	0.98%	8.4	2.42×

We also investigate the effect of raising the lower bound α. When α is set to higher values (e.g., 5% or above), the DP solver fails to find any feasible configuration satisfying the $\delta = 1\%$ constraint. This is because certain sensitive layers, particularly the early embedding-adjacent blocks and final output-adjacent blocks, exhibit steep accuracy degradation even at minimal sparsity levels. Forcing these layers to adopt sparsity rates of 5% or higher causes their individual accuracy costs to exceed the total budget, rendering the optimization problem infeasible. This observation further validates our design choice of setting $\alpha = 1\%$, which allows sensitive layers to remain nearly dense while concentrating sparsification on redundant middle layers.

A.17.4 KEY FINDINGS

The ablation studies yield three findings:

1. Varying the accuracy budget produces monotonic trade-offs. The DP solver consistently finds configurations that approach but respect the hard constraint, demonstrating effective utilization of the allowed accuracy budget.
2. Both 7B and 70B models exhibit identical structural patterns under varying δ. Larger models achieve higher sparsity due to increased redundancy, but the relative layer sensitivity rankings remain consistent.
3. The sparsity range $[\alpha, \beta]$ has minimal impact when δ is the binding factor. The default setting $[1\%, 15\%]$ provides sufficient flexibility, and expanding the range yields diminishing returns.

APPENDIX A.18: PRECOMPUTATION OVERHEAD ANALYSIS

Complexity Analysis. The precomputation phase of ASAF consists of tabulation construction and dynamic programming search. The tabulation tables H and Ξ enumerate all consecutive layer subsequences, producing $\sum_{i=1}^{L}(L-i+1) = O(L^2)$ entries. Combined with $|S|$ sparsity levels and per-configuration evaluation cost T_{eval}, the total tabulation complexity is:

$$\mathcal{C}_{\text{tabulation}} = O(L^2 \times |S| \times T_{\text{eval}}). \tag{A24}$$

The subsequent DP search operates on the precomputed tables, so tabulation is the dominant cost.

Scalability. The quadratic dependence on layer count L is exact in the recurrence: doubling L multiplies the tabulation cost by a factor of four. This one-time cost is amortized across all deployment instances: the resulting sparsity configuration is computed once and reused without further computation. The tabulation of different layer-sparsity configurations is embarrassingly parallel and can be distributed across multiple GPUs to reduce wall-clock time.

Where the Cost Sits. Llama-2-70B is the largest case in our evaluation; smaller Llama-2 models complete proportionally faster under the $O(L^2 \cdot |S|)$ complexity formula, with the sensitivity tabulation Ξ as the dominant cost term and the dynamic-programming search itself reduced to table lookups once the tables are populated. The resulting allocation is a model-level artifact: it depends only on the base weights and the target sparsity grid, not on the downstream task or deployment hardware. A single tabulation run therefore amortizes across every downstream serving instance of that model.

APPENDIX A.19: SPARSITY ALLOCATION AND SENSITIVITY ANALYSIS

A.19.1 ASAF'S LEARNED LAYER GROUPING AND SPARSITY ALLOCATION

To validate the consecutive-grouping assumption, Figure A10 visualizes ASAF's learned sparsity allocation across all 32 layers of Llama-2-7B. The framework automatically partitions the depth into consecutive groups whose rates form a single inverted-U profile: the first group (layers 1 to 4) and the last group (layers 29 to 32) receive the lowest rates in the allocation, the rate rises with depth to its maximum over the mid-depth layers (15 to 18), and then declines again toward the output. The smooth transitions between consecutive groups confirm that adjacent layers share similar compression tolerance.

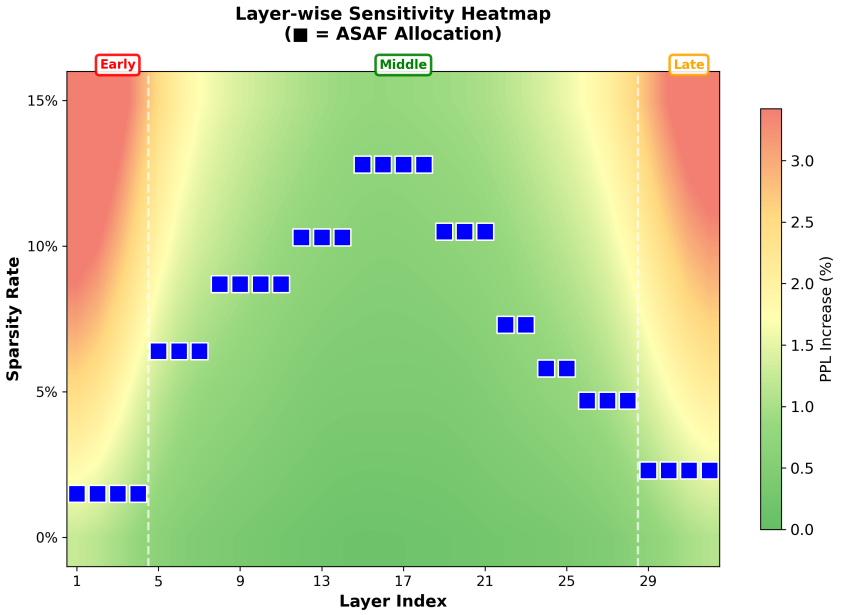


Figure A10: ASAF's learned sparsity allocation showing consecutive layer grouping with adaptive sparsity rates.

A.19.2 LAYER SENSITIVITY ANALYSIS

Figure A11 presents the empirical sensitivity validation. We measure per-layer sensitivity by applying pruning to individual layers while keeping others unchanged. The results reveal smooth depth-wise transitions: early/late layers exhibit higher sensitivity (2 to 3% perplexity increase), while mid-depth layers tolerate more aggressive pruning (0.5 to 1% increase). This sensitivity pattern directly explains ASAF's learned allocation in Figure A10, demonstrating that the framework automatically discovers optimal allocations correlated with empirical layer sensitivity. This pattern aligns with established Transformer behavior: early layers shape core token representations, final layers refine task-specific semantics, and mid-layer MLPs hold the largest structural redundancy.

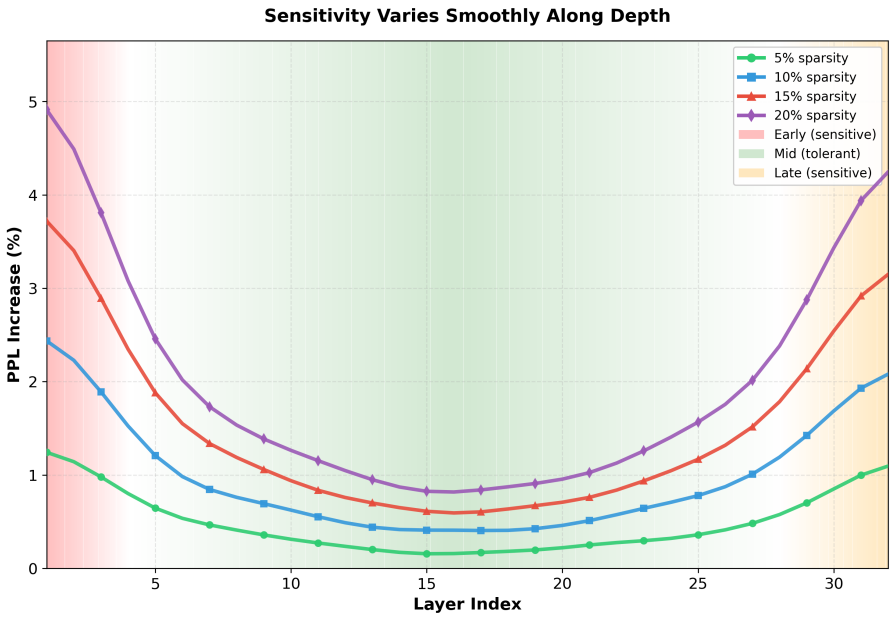


Figure A11: Per-layer sensitivity showing U-shaped pattern that correlates with ASAF's allocation.

Appendix A.20: Models, Software, and Benchmarks Used

This section lists the third-party models, software, datasets, and baseline implementations used in this work, with the licenses their publishers state.

Base models. Llama-2 (Touvron et al., 2023) and Llama-3 are distributed by Meta under the Meta Llama Community License.

Software stack. Our implementation uses PyTorch (BSD-3-Clause), the HuggingFace Transformers library (Apache-2.0), and the LM Evaluation Harness (Gao et al., 2021; 2024) (MIT).

Evaluation benchmarks. Our evaluation uses WikiText-2 (Merity et al., 2017), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021), HellaSwag (Zellers et al., 2019), LAMBADA (Radford et al., 2019), and ARC-Easy/Challenge (Clark et al., 2018).

Baseline implementations. We compare against QuaRot (Ashkboos et al., 2024c), SmoothQuant (Xiao et al., 2023), OmniQuant (Shao et al., 2024), and GPTQ (Frantar et al., 2022), cited at their original publications.

Appendix A.21: Per-Benchmark Results for the Compression Baselines

Table A11 gives the per-benchmark accuracies behind the averages reported in the main paper. The evaluation protocol matches Appendix A.11 throughout.

Table A11: Per-benchmark accuracy of ASAF against recent compression baselines on LLaMA-7B.

Method	WikiText-2 PPL ↓	PQ ↑	WG ↑	HS ↑	A-e ↑	A-c ↑	BoolQ ↑	MMLU ↑	Average ↑
SpinQuant (Liu et al., 2025b)	5.94	76.14	69.58	72.16	69.30	40.37	63.77	30.00	60.19
OWL (Yin et al., 2024)	24.55	62.48	58.72	44.79	45.03	26.19	58.48	26.84	46.08
DSA (Li et al., 2024)	12.50	68.13	65.97	61.84	65.26	39.42	66.84	31.05	56.93
SLIM (Mozaffari et al., 2025)	9.80	75.04	67.83	67.35	67.14	41.38	70.23	29.41	59.77
ASAF (Ours)	5.98	76.00	63.23	71.47	69.17	40.52	74.23	31.87	60.93

APPENDIX A.22: END-TO-END THROUGHPUT AND LATENCY BY CONFIGURATION

Table A12 lists each measurement configuration in its own row rather than in the combined form used in the main paper. Measurements are on a single RTX 3090 under the protocol of Appendix A.11.

Table A12: End-to-end throughput and latency on RTX 3090, the full form of the summary table in the main paper.

Model	Prompt length	Generation length	Batch size	Method	Tokens/s (↑)	Latency (ms) (↓)
Llama-2-7B	512	128	1	QuaRot	909	711
				ASAF	920	703
	2048	512	1	QuaRot	703	3657
				ASAF	720	3570
Llama-2-13B	2048	512	8	QuaRot	2611	7877
				ASAF	2687	7654
	4096	512	8	QuaRot	2129	17554
				ASAF	2203	16961