GQSA: Group Quantization and Sparsity for Accelerating Large Language Model Inference

Chao Zeng[*], Songwei Liu[*] [†], Shu Yang, Fangmin Chen, Xing Mei [‡], Lean Fu
ByteDance Inc,
{zengchaocs, cfangmin}@gmail.com, {liusongwei.zju, xing.mei}@bytedance.com

Abstract

Model compression has emerged as a mainstream solution to reduce memory usage and computational overhead. This paper proposes **GQSA**, a novel model compression framework specifically designed for LLMs. Traditional methods typically focus exclusively on either quantization or sparsification, but relying on a single strategy often results in significant performance loss at high compression rates. In contrast, GQSA integrates quantization and sparsification in a tightly coupled manner, leveraging GPU-friendly structured group sparsity and quantization for efficient acceleration. Building upon system-algorithm co-design principles, we propose a two-stage sparse optimization strategy that ensures the performance superiority of the compressed model. On the engine side, we introduce a "task-centric" parallel strategy, which, to the best of our knowledge, is the first application in the domain of sparse computing. Compared to the traditional 2:4 sparse method, the GQSA offers a more flexible and adjustable sparsity rate, as well as a higher weight compression rate, and is efficiently compatible with weight-only quantization methods. Experimental results demonstrate that, under the GQSA W4S50% compression setting, the model's accuracy surpasses that of both 2:4 pruning and W2 quantization. Furthermore, at the inference level, GQSA outperforms W2 by 1.26× and 2:4 pruning by 2.35× in terms of speed.

1 Introduction

Sparsity, combined with quantization (Lin et al., 2024; Shao et al., 2023), is a powerful approach to enhance model inference performance, reduce the size of LLMs, and enable their deployment on edge devices such as PCs (Gu et al., 2024; Liu et al., 2022). However, current sparsification strategies exhibit limited acceleration benefits due to the unstructured sparsity patterns typically generated by existing unstructured pruning methods (Han et al., 2015; Sun et al., 2023), which are poorly suited for hardware acceleration. Strategies such as SparseGPT (Frantar and Alistarh, 2023) and Wanda (Sun et al., 2023) address this issue by adopting a 2:4 sparsity pattern, leveraging NVIDIA GPUs' Sparse Tensor Core units for acceleration. Nevertheless, these approaches are constrained by hardware requirements such as a minimum operation shape of [m, n, k] = [16, 8, 16], which restrict their applicability to compute-intensive tasks like GEMM operations. These limitations pose significant challenges in accelerating the decoding process, the primary performance bottleneck in LLMs (Zeng et al., 2024). Unlike GEMM, decoding involves GEMV operations, where the Tensor Core's compute resources are underutilized, with approximately 87.5% of resources being wasted (Mishra et al., 2021). Consequently, SparseGPT and Wanda achieve up to 50% sparsity but remain inefficient in practical scenarios. Furthermore, as shown in Table 1, these methods are incompatible with weight-only quantization because Sparse Tensor Cores require both weights and activations to be in either floating-point or integer formats. Combining sparsification with weight-activation quantization leads to excessive compression of activation value representation, resulting in severe performance degradation. This limita-

Method	NVIDIA	
	Weight-only	Weight-Activation
2:4	✗	✓
GQSA	✓	✓

Table 1: Compatibility of 2:4 and GQSA Methods with Quantization Approaches.

[*]These authors contributed equally.
[†]Project Leader.
[‡]Corresponding author

tion significantly diminishes the practical utility of existing sparsification strategies.

To address these challenges, we propose a novel model compression method called GQSA, designed specifically for the decoding process and efficiently compatible with weight-only per-group quantization. GQSA explores a group sparsity pattern beyond the conventional 2:4 sparsity, achieving a better trade-off between accuracy and speed through a combination of algorithm-level optimizations and a customized software engine. Specifically, we reinterpret weight pruning as a particular form of quantization and introduce a group pruning based on group quantization. Our method incorporates the Block Sparse Row (BSR) format and designs a compact, low-precision weight storage structure to maximize the compression benefits of pruning and quantization. The GQSA method consists of two main stages. The first stage, Block Quantization-Pruning Optimization (BQPO), calibrates model parameters at the block level by optimizing weight distributions within block to minimize performance loss caused by group pruning and quantization. In the second stage, End-to-End Optimized Quantization-Pruning (E2E-OQP), the backbone network's weights are frozen, and only the quantization parameters are fine-tuned to optimize the global network performance. Unlike BQPO, E2E-OQP considers the global error distribution across blocks. Freezing the backbone network can not only reduce memory usage but also improve training efficiency. Extensive experiments demonstrate that GQSA achieves significant advantages in both model accuracy and inference speed, especially when applied to newly released advanced models such as LLaMA-3 and LLaMA-3.1 model family and Qwen2.5 models.

In summary, our contributions are as follows.

- We propose a sparse scheme seamlessly compatible with widely used weight-only and weight-activation quantization, effectively accelerating GEMV operations and reducing memory usage.
- We introduce a task-centric parallel implementation, addressing the workload balancing issue in sparse acceleration.
- We integrate group pruning with low-bit quantization techniques and achieves outstanding model performance through the two-stage optimization process of BQPO and E2E-OQP.

2 Related work

Compressing Large Language Models. Pruning and quantization are the two primary techniques for compressing LLMs. Pruning methods can be classified into structured (Chen et al., 2024; Ma et al., 2023; Ashkboos et al., 2024), semi-structuredcite (Frantar and Alistarh, 2023; Sun et al., 2023; Fang et al., 2024), and unstructured (Han et al., 2016, 2015; Sun et al., 2023) pruning, depending on the granularity of pruning. Structured pruning operates at a coarser granularity and offers significant acceleration, but it often results in a substantial loss of accuracy (Wang et al., 2024), limiting its application in LLMs. Unstructured pruning better preserves accuracy but provides limited improvements in inference speed (Frantar and Alistarh, 2023). Semi-structured pruning strikes a balance between accuracy retention and acceleration, though it is constrained by a sparsity of 50%, reducing its flexibility. Quantization reduces model size by replacing floating-point numbers with low-precision integers, which accelerates memory access during inference. Currently, high-bit quantization techniques such as AWQ (Lin et al., 2024), GPTQ (Frantar et al., 2022), QuIP (Chee et al., 2024), OmniQuant (Shao et al., 2023), and OWQ (Lee et al., 2024) are widely adopted. However, extremely low-bit quantization poses significant challenges, with mainstream methods struggling to maintain performance at low-bit levels. While techniques like AQLM (Egiazarian et al., 2024) and QuIP# (Tseng et al., 2024) aim to enhance low-bit quantization, they rely on vector quantization and complex codebooks, which hinder inference acceleration. Overall, existing model compression techniques continue to face substantial challenges in achieving an optimal balance between flexibility and compression rate.

Advantages of GQSA. Quantization and sparsity address model redundancy in different ways. Quantization reduces the precision of numerical representations, while sparsity compresses the model by eliminating certain neurons. These two techniques are largely orthogonal, and GQSA leverages both dimensions to achieve flexible and high compression rates. Although both GQSA and 2:4 pruning are semi-structured pruning methods, GQSA offers several advantages over 2:4 pruning. First, GQSA supports an adjustable sparsity rate, whereas 2:4 pruning, designed for NVIDIA's 2:4 TensorCore, mandates a 50% sparsity rate by forcing two out of every four weights to be zero. Our group sparsity model, in combination with co-designed operators, enables efficient implementation at various sparsity levels. Second, GQSA achieves a higher weight compression rate. For instance, with a 50% sparsity rate, 2:4 pruning requires additional metadata to identify the positions of retained neurons, which are chosen randomly. In contrast, GQSA stores location information at the group/block level, significantly enhancing compression efficiency. Finally, 2:4 pruning is restricted to NVIDIA's 2:4 TensorCore and is incompatible with mainstream weight-only quantization methods. In contrast, GQSA is highly compatible with weight-only quantization, thanks to its customized two-stage optimization, leading to a substantial increase in overall compression rate.

3 GQSA

In this section, we provide a detailed exposition of GQSA. Section 3.1 begins with an examination of weight quantization and salient weight selection principles. Building upon these foundations, Section 3.2 introduces the innovative GQS Layer, designed to maximize the compression advantages from both quantization and pruning. The subsequent sections (3.3 and 3.4) detail our two-stage optimization algorithm, which delivers exceptional model performance. Concluding the section, 3.5 proposes a novel task-centric parallel strategy for efficient inference acceleration.

3.1 Preliminary

Weight Quantization. LLM quantization maps floating-point values to a lower-bit discrete value space, significantly reducing model size, enhancing computational efficiency, and accelerating inference. The process typically involves two steps: determining the quantization parameters (scale and zero-point) and computing the corresponding quantized tensor. For uniform asymmetric quantization, which is used in this paper, the scale s and zero-point z are determined by:

$$s = \frac{\max(W)-\min(W)}{2^n-1}, z = -\left\lfloor \frac{\min(W)}{s} \right\rceil, \quad (1)$$

where W represents the model weights and n denotes the quantization bit-width. The elements of the quantized tensor can be computed as follows:

$$\tilde{W} = \text{clamp}(\left\lfloor \frac{W}{s} \right\rceil + z, 0, 2^n - 1), \quad (2)$$

where $\lfloor \cdot \rceil$ represents the rounding operation, and $\tilde{W}$ represents the quantized integer weights. When it is necessary to update the quantized weights of the model, the weights are converted back to full precision during the forward propagation phase, as shown below:

$$\hat{W} = (\tilde{W} - z) \cdot s, \quad (3)$$

where $\hat{W}$ denotes the dequantized weights utilized in the forward computation. The processes of quantization (as shown in Equation (2) and dequantization (as shown in Equation (3) are integrated into the computational graph, enabling quantization-aware optimization through gradient descent.

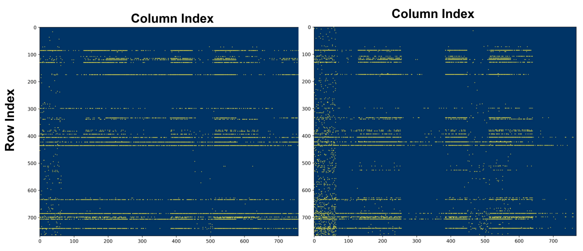


Figure 1: The distribution of the top 1% significant weights in the Hessian matrix, derived from the k_{proj} and q_{proj} distributions in the LLaMA-7B model.

Salient Weight. In LLMs, different weights exhibit different importance. By pruning unimportant weights, memory usage can be greatly reduced while maintaining nearly unchanged performance. Early studies used the absolute value of weights to evaluate weight importance, but ignored the role of activation. The Hessian metric combines weights and activations and is a more effective metric that has been verified by multiple methods (Shang et al., 2023; Frantar and Alistarh, 2023). Therefore, this paper uses the Hessian matrix to evaluate weight importance.

$$s_i = \frac{w_i^2}{[H^{-1}]_{ii}^2}, \quad (4)$$

where H represents the Hessian matrix of each layer, and w_i denotes the weight values. In the subsequent sections, s_i refers to the criteria for identifying salient weights. AWQ (Lin et al., 2024) demonstrates that the top 1% of salient weights in the model are crucial to performance, so accurately retaining these weights is key to performance. Figure 1 visualizes the distribution of salient weights in the OPT model, revealing a segmented pattern along the rows. Consequently, group by rows and selecting salient weight group emerges as a natural optimization strategy.

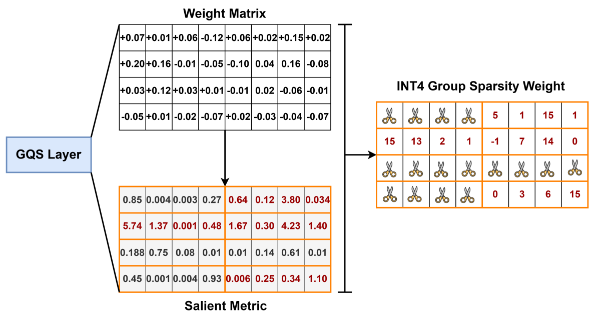


Figure 2: GQSA computes saliency metrics based on weights and activations, grouping the weights along the row dimension (illustrated with groups of four elements). Group pruning is then applied based on the average saliency metrics within each group, resulting in the formation of the GQS layer.

3.2 GQS Layer

Weight-only per-group quantization has gained significant recognition in both academia (Lin et al., 2024; Shao et al., 2023; Frantar et al., 2022) and industry (Gerganov, 2024). To enable efficient sparse acceleration compatible with weight-only per-group quantization approach, we conduct a comprehensive analysis of the distribution of salient weights within the model. As depicted in Figure 1, we observe that salient weights exhibit distinct segmented distribution patterns. Based on this observation, we introduce a novel structured group pruning method that goes beyond the conventional 2:4 sparsity pattern, leveraging the segmented distribution characteristics of salient weights. As illustrated in Figure 2, we begin by grouping weights along the row dimension, assuming a group size of 4 for simplicity. For each group, we compute a salient metric using the Hessian matrix. Based on this metric, we prune non-salient weight groups and quantize the remaining salient groups to 4 bits, thereby further compressing the model size. Additionally, by adopting the BSR sparse format, we convert the compression gains from pruning into actual storage savings. The specific storage structure in Figure 2 is shown below:

```
rowIndex = {0, 1, 3, 3, 4}
 groups  = {1, 0, 1, 1}
 values  = {5, 1, 15, 1, 15, 13, 2, 1,
           -1, 7, 14, 0, 0, 3, 6, 15}
```

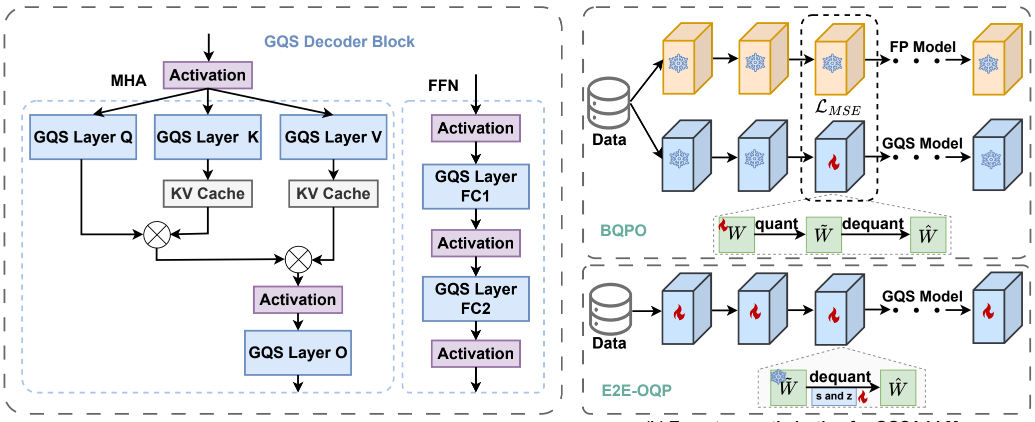
where `rowIndex[i]` represent the offset of each row `i`, where `i` belongs to the range $[0, \text{rows}]$. The difference `rowIndex[r+1] - rowIndex[i]` indicates the number of non-zero groups in the `i`-th row. Additionally, `rowIndex[rows]` represents the total number of non-zero groups. The array `groups[i]` stores the indices of the non-zero groups; for instance, if `groups[1] = 0`, it means that the second group is located in the 0th column (in terms of group units). Finally, `values` stores the values of the non-zero groups for each row.

3.3 BQPO

In the first stage (Figure 3(b)), we apply the BQPO method to optimize the GQS model, aiming to mitigate the accuracy degradation caused by group quantization and pruning. This is achieved by adjusting the weight parameters within each block. Traditional QAT methods typically optimize the entire network's weights in an end-to-end fashion, as illustrated in Equations (2) and (3). Similarly, most pruning approaches adopt a global end-to-end strategy to update the remaining unpruned parameters. However, such methods often demand substantial computational resources and large-scale training datasets. To improve optimization efficiency, BQPO adopts a block-wise optimization strategy. Prior studies, such as OmniQuant and AffineQuant, have shown that block-wise optimization can significantly reduce both training time and memory consumption. Unlike OmniQuant and AffineQuant, which primarily optimize quantization parameters (inter-channel smoothing factors and weight clipping thresholds), GQSA suffers from more severe performance degradation due to its combination of high structured sparsity and low-precision quantization. As a result, BQPO focuses on optimizing the remaining weights to recover performance under extreme compression settings. This block-wise approach enables significant performance restoration with minimal additional training cost compared to global optimization techniques.

3.4 E2E-OQP

Compared to BQPO, E2E-OQP not only performs intra-block optimization but also accounts for the overall error across the entire network, thereby capturing cross-block dependencies. As illustrated in Figure 3(b), E2E-OQP differs from conventional quantization-aware training (QAT) methods. Assuming that BQPO has already yielded a well-optimized model in the first stage, E2E-OQP initializes training using the BQPO-optimized weights. During this phase, we freeze the primary network



(a) One basic block of the GQSA LLM

(b) Two-stage optimization for GQSA LLM

Figure 3: Overview of GQSA. (a) We propose a group quantization and sparse LLMs, where linear layers are replaced by GQS layers. (b) We use the two-stage optimization method BQPO and E2E-OQP to recover the performance of the extremely compressed model.

weights $\tilde{W}$ and optimize only the quantization parameters s and z to further refine model performance. The design of E2E-OQP underscores the advantages of the GQSA framework. Specifically, during fine-tuning, we employ the block-sparse row (BSR) format: the remaining group weights are quantized to low bit-width and frozen, while pruned groups are discarded entirely. This strategy enables effective fine-tuning of the quantization parameters without requiring sparse masks, thereby restoring the performance of the GQSA model under extreme compression. Overall, E2E-OQP achieves substantial memory savings by focusing solely on the quantization parameters of the remaining groups while maintaining 4-bit quantization across the main network. A detailed comparison of the resource consumption of BQPO and E2E-OQP is provided in Appendix A, demonstrating the efficiency advantages of the GQSA approach.

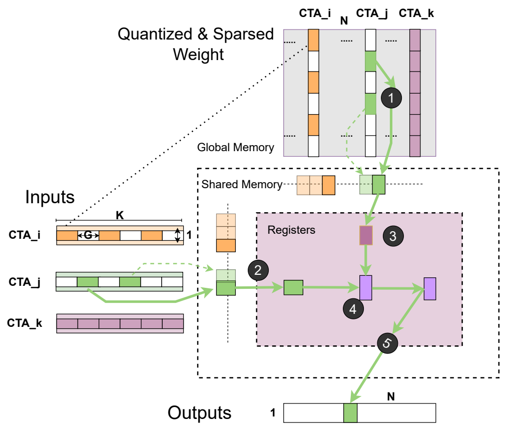


Figure 4: A simplified view of GQSA's operator calculation flow. G represents sparse and quantized group size.

3.5 Custom Software Engine

GPU has many processing elements called Streaming Multiprocessors (SMs) and uses a large number of threads to perform computing tasks in parallel. Threads are structured into thread blocks (CTAs), which become the smallest scheduling execution unit on SMs. Therefore, the computation target is decomposed and mapped to each thread block, called CTA, to achieve parallel computing. As shown in Figure 4, for a GEMV task of shape 1×N×K, each thread block is responsible for computing a 1×BN output tile, which is decomposed into $\frac{K}{BK}$ sub-GEMV tasks of shape 1×BN×BK. In offline pre-processing, quantized weights are grouped by size G and saved as gguf format along with scaling factors and zero points. This means that each sub-GEMV task computes $\frac{BK}{G} * BN$ non-sparse groups held by one or more output channels. It should be noted that the logical addresses between non-sparse groups are not necessarily consecutive, so the corresponding activation group needs to be accessed according to the real group index of each group. ❶ The thread-block issues asynchronous copy instructions to fetch small chunks of input data (tiles) from global memory to shared memory. ❷ As soon as a tile arrives in shared memory, it is further sliced into smaller chunks (fragments) and copied into registers. ❸ Once all necessary components are in the registers, the quantized matrix undergoes dequantization. ❹ The dequantized matrix and inputs are then processed by TensorCores (MMA) or CudaCores (FMA) instructions. ❺ Finally, the accumulated results are written back from the registers to the outputs in global memory.

Setting	Method	LLaMA-7B		LLaMA-13B		LLaMA-2-7B		LLaMA-2-13B		LLaMA-3-8B		LLaMA-3.1-8B	
		WikiText2	C4	WikiText2	C4	WikiText2	C4	WikiText2	C4	WikiText2	C4	WikiText2	C4
W2	GPTQ	44.01	27.71	15.60	15.29	36.77	33.70	28.14	20.97	210	4.1e4	250	80.3
	QUIP	29.74	33.74	12.48	21.94	39.73	31.94	13.48	16.16	84.97	1.3e2	-	-
	PB-LLM	24.61	49.73	17.73	26.93	25.37	29.84	49.81	19.82	44.12	79.2	-	-
	Omniquant	15.47	24.89	13.21	18.31	37.37	90.64	17.21	26.76	2.1e4	6.0e4	7.3e3	1.3e4
	LeanQuant	15.65	17.62	9.64	10.93	16.98	17.89	10.32	11.73	41.78	36.50	-	-
	SliM-LLM	14.58	32.91	**8.87**	13.85	16.01	16.00	9.41	**9.41**	39.66	1.1e2	-	-
2:4	SparseGPT	**11.20**	13.59	9.14	11.34	10.95	13.56	8.32	11.30	16.56	22.99	16.62	23.22
2:4	Wanda	11.53	14.41	9.58	12.07	11.02	15.07	8.27	12.12	25.27	36.40	23.93	36.24
w4s20%	GQSA	6.58	8.30	5.75	7.57	6.57	8.32	5.86	7.51	8.43	12.54	8.40	12.37
w4s30%		7.91	9.74	6.72	8.32	7.56	9.49	6.87	8.49	9.79	14.58	9.69	14.32
w4s40%		9.10	11.24	7.70	9.57	8.43	11.31	7.13	9.53	11.80	17.61	11.56	17.32
w4s50%		11.33	**12.03**	9.21	**10.85**	**10.64**	**12.82**	**7.80**	10.93	**13.81**	**20.85**	**13.56**	**20.43**

Table 2: Wikitext2 and C4 perplexity (↓) for LLaMA-1, LLaMA-2, LLaMA-3 and LLaMA-3.1 models, with a context length of 2048.

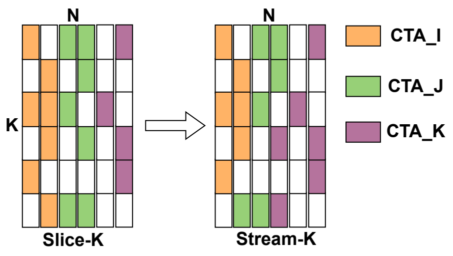


Figure 5: Workload balancing through parallel task partitioning.

Furthermore, to enhance the efficiency of sparse computing, we introduced Stream-K (Osama et al., 2023). As show in Figure 5, the classic Slice-K (Guo et al., 2024) assigns output tiles independently to thread blocks. Each thread block processes one or more rows of the left operand and one or more columns of the right operand to compute the corresponding output tile by slicing along the internal K dimensions. However, when the weight matrix exhibits high sparsity, the uneven distribution of workloads can result in the "straggler" problem, where small workloads cause inefficiencies. Stream-K addresses this issue by decomposing the workload at a finer granularity, allowing multiple thread blocks to collaborate in computing a single output tile.

4 Experiments

4.1 Experimental Settings

Models and Tasks. We selected the LLaMA (Touvron et al., 2023a), LLaMA-2 (Touvron et al., 2023b), LLaMA-3, LLaMA-3.1 (Dubey et al., 2024) and OPT (Zhang et al., 2022) models to benchmark our method. Following previous studies, we evaluated the model's language modeling capability on the WikiText2 (Merity et al., 2016) and C4 (Raffel et al., 2020) datasets. To assess performance on zero-shot tasks, we selected several mainstream benchmarks, including PIQA (Bisk et al., 2020), ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), and Winogrande (Sakaguchi et al., 2021), and conducted evaluations using lm-eval.

Baselines. We conducted a comprehensive comparison of our method with several recently published techniques in both structured and semi-structured pruning. Given that our implementation achieved INT4 along with 50% structured pruning, we also compared our approach with pure INT2 quantization. For structured pruning, we compared our results with LLMPruner (Ma et al., 2023), SliceGPT (Ashkboos et al., 2024) and ShortGPT (Men et al., 2024). For semi-structured pruning, we utilized SparseGPT (Frantar and Alistarh, 2023) and Wanda (Sun et al., 2023) for comparison. Additionally, we selected OmniQuant (Shao et al., 2023), QuIP (Chee et al., 2024), PB-LLM (Shang et al., 2023), GPTQ (Frantar et al., 2022), LeanQuant (Zhang and Shrivastava, 2024), and SliM-LLM (Huang et al., 2024) as benchmarks for W2 quantization.

Implementation Details. To evaluate the performance of GQSA across various configurations, we implemented sparsity levels of 20%, 30%, 40%, and 50%, using 4-bit weight-only per-group quantization. To strike a balance between model performance and inference speed, a group size of 16 was selected as the optimal configuration. The AdamW optimizer (Loshchilov, 2017) with a learning rate of 1e-5 was employed to optimize both BQPO and E2E-OQP. The optimization data was randomly sampled from the WikiText2 and C4 datasets, consisting of 4,096 samples, each containing 2,048 tokens. BQPO was trained for 5 epochs, while E2E-OQP was trained for 2 epochs.

4.2 Evaluation on Language Generation Tasks

To assess the performance of GQSA under extreme compression conditions, we first compared its perplexity against baseline method. As shown in Table 2, GQSA surpasses the performance of current state-of-the-art weight-only per-group quantization methods, including GPTQ, QuIP, OmniQuant, LeanQuant, under a 50% structured pruning combined with INT4 quantization. It also surpasses mixed-precision quantization models like PB-LLM and SliM-LLM. Furthermore, GQSA achieves comparable results to 2:4 semi-structured pruning while delivering substantial improvements in compression ratio and speedup. Similar results are presented in Table 16 for Qwen2.5 and Table 17 for OPT models, where GQSA consistently matches or surpasses baseline methods, even under more stringent compression settings. Furthermore, we observe that existing model compression methods often experience significant performance degradation on the latest large language models (e.g., LLaMA-3 and LLaMA-3.1). In contrast, GQSA demonstrates robust performance even in scenarios where other methods encounter substantial performance degradation.

Model	Setting	Method	Zero-Shot Accuracy				
			PIOA	ARC-E	ARC-C	Hellaswag	Winogrande
LLaMA-2-7B	25%	ShortGPT	60.1	31.0	41.7	44.0	60.8
		SliceGPT	67.5	34.5	55.6	55.1	62.9
		LLM-Pruner	**75.7**	**37.2**	62.0	60.1	62.2
	W4S30%	GQSA	74.32	34.98	**66.04**	**64.40**	**65.98**
	40%	ShortGPT	50.7	27.7	25.6	30.1	50.3
		SliceGPT	58.5	27.3	43.5	43.6	57.9
		LLM-Pruner	70.7	**31.3**	50.7	53.5	56.1
	W4S40%	GQSA	**71.27**	30.72	**61.32**	**58.48**	**61.48**
LLaMA-2-13B	25%	ShortGPT	73.1	41.9	60.1	60.6	**70.5**
		SliceGPT	69.6	40.2	61.5	59.4	67.0
		LLM-Pruner	**79.4**	**43.5**	67.8	65.4	63.5
	W4S30%	GQSA	75.68	39.85	**71.55**	**70.45**	66.54
	40%	ShortGPT	62.4	32.2	44.8	47.8	62.8
		SliceGPT	59.9	29.2	44.1	49.6	61.6
		LLM-Pruner	75.3	35.4	56.3	60.2	57.8
	W4S40%	GQSA	**75.30**	**35.82**	**66.50**	**65.40**	**65.98**

Table 3: Zero-shot performance between LLaMA-2-7B and LLaMA-2-13B models under 25% and 40% structured pruning, GQSA with 30% and 40% structured pruning along with INT4 quantization.

4.3 Evaluation on Zero-Shot Tasks

To further validate our model, we conducted a detailed comparison of its zero-shot accuracy against baseline methods. Given the limited data availability from these baselines methods, we selected LLaMA-2-7B and LLaMA-2-13B for the analysis. Table 3 compares GQSA with structured pruning, where GQSA achieved substantial performance gains at equivalent or higher pruning rates, with these benefits becoming more pronounced at higher pruning levels. Table 4 compares GQSA with semi-structured pruning and W2 weight-only per-group quantization. Compared to W2 per-group quantization, GQSA consistently delivered superior performance improvements at the same compression ratio. Under the conditions of 50% structured pruning with INT4 quantization, GQSA outperformed OmniQuant W2 per-group quantization, yielding average accuracy gains of 5.4% for LLaMA-2-7B and 5.7% for LLaMA-2-13B. Given that GQSA operates in a more challenging compression setting than semi-structured pruning, we compare GQSA W4 40% with semi-structured pruning. Experimental results reveal that GQSA achieves superior performance even with a compression rate 3× higher than that of 2:4 pruning. Furthermore, GQSA demonstrates significant advantages in both speed and accuracy compared to 2:4 pruning. Considering its compression efficiency and flexibility, GQSA emerges as the clear superior choice.

Model	Setting	Method	Zero-Shot Accuracy				
			PIOA	ARC-E	ARC-C	Hellaswag	Winogrande
LLaMA-2-7B	W2	OmniQuant	64.52	26.10	44.94	49.27	54.53
		LeanQuant	65.4	24.7	44.2	-	57.4
	W4S50%	GQSA	**68.01**	**29.01**	**58.33**	**52.72**	**58.41**
	2:4	SparseGPT	70.13	29.35	61.14	56.89	**63.14**
		Wanda	70.12	30.55	61.32	55.34	62.83
	W4S40%	GQSA	**71.27**	**30.72**	**61.32**	**58.48**	61.48
LLaMA-2-13B	W2	OmniQuant	68.06	30.03	57.07	56.56	52.95
		LeanQuant	70.6	28.2	56.7	-	60.7
	W4S50%	GQSA	**72.47**	**33.28**	**63.01**	**62.11**	**62.28**
	2:4	SparseGPT	72.74	32.59	66.04	62.78	66.54
		Wanda	73.72	34.39	66.33	63.12	**66.93**
	W4S40%	GQSA	**75.30**	**35.82**	**66.50**	**65.40**	65.98

Table 4: Zero-shot performance between LLaMA-2-7B and LLaMA-2-13B under W2 quantization method, 50% semi-structured pruning, and GQSA with 40% and 50% structured pruning along with INT4 quantization.

4.4 Comparison with Joint Quantization and Sparsity Methods

To further compare GQSA with co-design methods for quantization and sparsity, we integrate W4 quantization with 2:4 semi-structured sparsity into GPTQ, AWQ, and OmniQuant. This ensures a fair comparison with GQSA under equivalent bit-width and sparsity constraints. As shown in Table 5, GQSA consistently achieves higher zero-shot accuracy than these baselines, demonstrating the superiority of our co-optimization algorithm. More-

Model	Setting	Method	Zero-Shot Accuracy				
			PIOA	ARC-E	ARC-C	Hellaswag	Winogrande
LLaMA-2-7B	W4 2:4	GPTQ	65.94	49.45	26.37	49.09	58.00
		AWQ	66.97	54.42	28.98	52.06	58.08
		OmniQuant	67.24	55.84	**29.01**	52.64	56.14
	W4 S50%	**GQSA**	**68.01**	**58.33**	**29.01**	**52.72**	**58.41**
LLaMA-2-13B	W4 2:4	GPTQ	70.02	59.64	32.56	59.72	60.80
		AWQ	71.16	61.36	331.4	61.06	51.59
		OmniQuant	71.06	59.97	32.90	60.11	61.43
	W4 S50%	**GQSA**	**72.47**	**63.01**	**33.28**	**62.11**	**62.28**
LLaMA-3-8B	W4 2:4	GPTQ	62.24	43.52	25.94	40.52	53.12
		AWQ	64.74	51.18	29.12	51.45	59.54
		OmniQuant	63.93	47.81	26.88	49.62	59.83
	W4 S50%	**GQSA**	**68.06**	**52.32**	**29.10**	**52.05**	**60.89**
LLaMA-3.1-8B	W4 2:4	GPTQ	61.86	42.63	25.00	35.88	56.99
		AWQ	64.80	49.28	28.50	50.60	60.67
		OmniQuant	65.94	52.36	28.05	49.39	60.01
	W4 S50%	**GQSA**	**68.34**	**52.80**	**29.33**	**52.92**	**61.75**

Table 5: Comparison of Zero-Shot Accuracy: GQSA versus joint quantization and sparsity methods.

over, the elegant integration of group sparsity with weight-only quantization in GQSA also leads to significantly better inference performance than the W4 2:4 compression method.

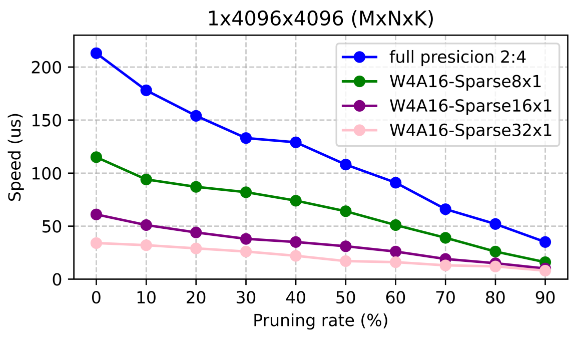


Figure 6: Comparison of GEMV acceleration of our GQSKernel on RTX 4080.

4.5 Inference Engine Evaluation

Kernel Benchmark. We compared GQSKernel with the 2:4 sparse kernel on a (1, 4096) × (4096, 4096) dimension. Due to the flexibility of GQSKernel, it can accommodate varying group sparsity sizes. The experimental results, presented in Figure 6, show that as sparsity increases, the GEMV computation speed improves. Moreover, GQSKernel consistently outperforms the 2:4 sparse mode across all group granularity settings. At 50% sparsity, GQSA achieves a 3× inference speedup compared to the 2:4 sparse mode.

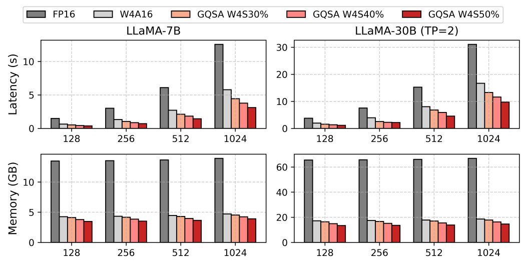


Figure 7: Inference latency (top) and memory usage (bottom) on an NVIDIA A800-40GB GPU with a fixed input length of 15. W8 results are provided in the Appendix Table 18.

End-to-end throughput. The acceleration of quantization primarily results from memory access savings, whereas sparsity acceleration arises from both memory access and computational savings. We integrated the GQSKernel into FastTransformer and compared it with the FP16 implementation. The experimental results, as shown in Figure 7, indicate that GQSA achieves a 4× reduction in inference latency on the LLaMA-7B model under the GQSA W4S50% setting with a 1024 output length. Additionally, as presented in Appendix Table 12, GQSA further enhances the acceleration potential of the compressed model compared to separate quantization or sparsity methods by simultaneously reducing redundancy in both dimensions of LLMs. For instance, the inference speeds of LLaMA-7B for S50%, W2, and W4S50% are 878.90 ms, 475.55 ms, and 377.98 ms, respectively. Overall, GQSA demonstrates the most significant performance improvement.

SeqLen	Method	Latency (ms)
128	W4A16	642.24
	W4 2:4 Pruning	513.79
	GQSA W4 S50%	**377.98**
256	W4A16	1312.91
	W4 2:4 Pruning	1112.96
	GQSA W4 S50%	**699.26**
512	W4A16	2707.26
	W4 2:4 Pruning	1966.45
	GQSA W4 S50%	**1433.43**
1024	W4A16	5786.8
	W4 2:4 Pruning	4118.36
	GQSA W4 S50%	**3110.54**

Table 6: The inference latency and memory usage of GQSA and 2:4 pruning are compared on an NVIDIA A800-40GB GPU with a fixed input length of 15.

Additionally, we compared GQSA's performance with that of state-of-the-art sparse schemes, such as SparseGPT and Wanda's 2:4 sparse scheme. The experimental results, presented in Table 6, demonstrate that GQSA outperforms these methods in terms of inference latency and accuracy.

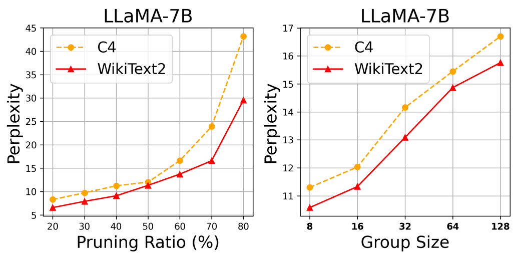


Figure 8: The ablation studies on the LLaMA-7B model to evaluate the impact of different structured pruning group sizes (right) and sparsity levels (left) on model performance.

4.6 Ablation Experiments

We investigated the impact of group size and sparsity on the performance of the GQSA model. As shown in Figure 8 (left), GQSA demonstrates robust performance at sparsity levels of 50% or lower. When sparsity exceeds 60%, a noticeable performance degradation occurs. However, even at an extreme sparsity level of 80%, GQSA achieves a perplexity below 30, avoiding performance collapse. Figure 8 (right) illustrates the relationship between group size and model performance. Overall, model performance exhibits a clear correlation with group size. Based on performance considerations, we selected 16 as the default group size for the model.

5 Conclusion

We propose GQSA, an efficient sparse acceleration method for the decoding process, compatible with weight-only per-group quantization. Through a comprehensive analysis of LLMs weights, we investigated group sparse modes beyond the 2:4 sparsity mode. To enhance model performance, we implemented a two-stage sparse optimization strategy, comprising BQPO and E2E-OQP. Based on the BSR format, we then developed an efficient sparse inference engine to fully leverage the synergistic benefits of quantization and sparsity. Extensive experimental results demonstrate that GQSA effectively integrates at both the algorithmic and system levels, offering a superior accuracy-speed trade-off compared to traditional 2:4 sparsity and quantization approaches.

Limitations

The proposed GQSA extends beyond the 2:4 sparsity pattern to explore group sparsity patterns, enabling efficient compatibility with weight-only per-group quantization. By combining algorithm-level optimizations with a customized inference engine, our approach achieves an improved balance between accuracy and inference speed. However, the current method does not address activation quantization, and due to resource limitations, it has not yet been applied to large language models (LLMs) exceeding 100 billion parameters. These limitations present promising directions for future research, and we are optimistic that they will be addressed in subsequent work.

Ethics Statement

This paper introduces a method to tackle the challenges of compressing large language models (LLMs), with the goal of facilitating their wider application and adoption. In the context of current research, ethical considerations surrounding LLMs have received substantial attention. Our findings indicate that the proposed method does not exacerbate existing biases or compromise ethical standards.

References

Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. Slicegpt: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, 05, pages 7432–7439.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. 2024. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36.

Xiaodong Chen, Yuxuan Hu, and Jing Zhang. 2024. Compressing large language models by streamlining the unimportant layer. *arXiv preprint arXiv:2403.19135*.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. 2024. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*.

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

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

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

M. Gerganov. 2024. llama.cpp: A high-performance implementation of llama. Accessed: 2024-12-12.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2024. Minillm: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*.

Han Guo, William Brandon, Radostin Cholakov, Jonathan Ragan-Kelley, Eric P Xing, and Yoon Kim. 2024. Fast matrix multiplications for lookup table-quantized llms. *arXiv preprint arXiv:2407.10960*.

Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*.

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

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Xianglong Liu, Luca Benini, Michele Magno, and Xiaojuan Qi. 2024. Slim-llm: Salience-driven mixed-precision quantization for large language models. *arXiv preprint arXiv:2405.14917*.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. 2024. Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 13355–13364.

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

Chang Liu, Chongyang Tao, Jiazhan Feng, and Dongyan Zhao. 2022. Multi-granularity structural knowledge distillation for language model compression. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1001–1011.

I Loshchilov. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

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

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

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

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

Mohammad Mozaffari and Maryam Mehri Dehnavi. 2024. Slim: One-shot quantized sparse plus low-rank approximation of llms. *arXiv preprint arXiv:2410.09615*.

Muhammad Osama, Duane Merrill, Cris Cecka, Michael Garland, and John D Owens. 2023. Stream-k: Work-centric parallel decomposition for dense matrix-matrix multiplication on the gpu. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pages 429–431.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

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

Yuzhang Shang, Zhihang Yuan, Qiang Wu, and Zhen Dong. 2023. Pb-llm: Partially binarized large language models. *arXiv preprint arXiv:2310.00034*.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. 2024. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *arXiv preprint arXiv:2402.04396*.

Weilan Wang, Yu Mao, Dongdong Tang, Hongchao Du, Nan Guan, and Chun Jason Xue. 2025. When compression meets model compression: Memory-efficient double compression for large language models. *arXiv preprint arXiv:2502.15443*.

Zixiao Wang, Jingwei Zhang, Wenqian Zhao, Farzan Farnia, and Bei Yu. 2024. Moreaupruner: Robust pruning of large language models against weight perturbations. *arXiv preprint arXiv:2406.07017*.

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

Chao Zeng, Songwei Liu, Yusheng Xie, Hong Liu, Xiaojian Wang, Miao Wei, Shu Yang, Fangmin Chen, and Xing Mei. 2024. Abq-llm: Arbitrary-bit quantized inference acceleration for large language models. *arXiv preprint arXiv:2408.08554*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Tianyi Zhang and Anshumali Shrivastava. 2024. Leanquant: Accurate large language model quantization with loss-error-aware grid. *arXiv preprint arXiv:2407.10032*.

Appendix

A Training Efficiency of GQSA

Table 7 lists the memory and time required to train the Llama-2 model using GQSA. The results show that GQSA requires only minimal resource overhead, and the 7B model only takes less than 10 hours to train with 9.3GB of memory, which is much less than the 14GB memory requirement to load the FP16 model, which is very efficient. It is also significantly better than other pruning methods, such as LLM-Pruner, which requires 18GB and takes less training time.

LLaMA-2	BQPO		E2E-OQP	
	Memory (GB)	Time (h)	Memory (GB)	Time (h)
7B	9.3	5.1	7.6	4.2
13B	14.3	7.3	11.7	6.4

Table 7: Detailed training time and training memory for GQSA at different model sizes and quantization bits on a single A100-40GB GPU.

B The effects of BQPO and E2E-OQP on the model's performance

Table 8 presents the impact of BQPO and E2E-OQP on model performance. BQPO optimizes weights in a block-wise manner, effectively preserving the performance of the GQS model. Finally, E2E-OQP, which accounts for cross-layer errors, yields the best model performance.

Method	LLaMA-13B		LLaMA-2-13B	
	WikiText2	C4	WikiText2	C4
BQPO	12.90	13.39	10.55	13.56
BQPO+E2E-OQP	9.21	10.85	7.80	10.93

Table 8: The effectiveness of BQPO and E2E-OQP methods for compressing LLaMA-13B and LLaMA-2-13B models.

C GQSA performance under weight-activation quantization

Unlike other algorithms that are limited to weight-only quantization, weight-activation quantization or model pruning, GQSA can not only efficiently combine pruning with weight-only quantization, but also support pruning with weight activation quantization. Our GPU-friendly grouped semi-structured sparse solution can be seamlessly combined with weight-only quantization or weight+activation quantization. On the basis of quantization, we can further improve performance by skipping some operations through sparsity.

Model	Settings	WikiText2	C4
LLaMA-2-7B	W4A8S50%	7.84	11.04
LLaMA-2-13B	W4A8S50%	14.09	21.26

Table 9: Performance comparison of GQSA with weight-activation quantization.

As show in Table 9, GQSA effectively preserves model accuracy under W4A8S50% quantization for both LLaMA-2-7B and LLaMA-2-13B architectures, maintaining strong performance despite simultaneous weight and activation quantization with 50% sparsity.

D Comparison of Quantized & Pruned Works

A comparison with SparseGPT's joint sparsification and quantization. In SparseGPT's report, "Joint Sparsification & Quantization" performs worse than "Sparsification-only," so we initially did not include it in our main content. However, for completeness, the following Table 10 presents a direct comparison between GQSA and SparseGPT's "Joint Sparsification & Quantization" on LLaMA-2-13B and LLaMA-3-8B. The results demonstrate that GQSA provides a more significant performance advantage.

Method	LlaMA-2-13B		LlaMA-3-8B	
	WikiText2	C4	WikiText2	C4
SparseGPT 2:4	8.32	11.30	16.56	22.99
SparseGPT 2:4+INT4	9.25	12.74	19.43	26.34
GQSA W4S50%	7.80	10.93	13.84	20.85

Table 10: Performance comparison of GQSA with SparseGPT.

Comparison with contemporaneous works. To further validate the superiority of GQSA, we conduct comparative evaluations with contemporaneous methods including SliM-LoRA (Mozaffari and Dehnavi, 2024) and DC-W8A8 (Wang et al., 2025). **SliM-LoRA** employs 4-bit weight quantization combined with Wanda's 2:4 pruning but fails to overcome the limitations of semi-structured sparsity. Since NVIDIA's 2:4 Tensor Cores do not support weight-only quantization, the inference acceleration benefits remain limited. Additionally, the

2:4 sparse format retains randomly positioned neurons and requires storing an equal amount of metadata to record their locations, preventing effective memory compression. SliM-LoRA also introduces the LoRA-Adapter; however, due to the quantization and sparsity of the main network, the LoRA-Adapter cannot be directly integrated and must be stored separately, increasing inference complexity. According to the SliM paper, its sparse quantization matrix can even reduce inference speed on the A100 GPU. In contrast, GQSA's sparse quantization achieves a 4.3 × inference speedup over FP16, highlighting SliM's shortcomings in both inference acceleration and memory compression. **DC-W8A8** incorporates sparsity into W8A8 quantization but relies on unstructured sparsity with a sparsity rate of only 20%, offering minimal memory compression benefits. As stated in its paper, DC-W8A8 achieves only a 2.2× compression ratio compared to FP16, whereas GQSA achieves a 4.3× compression ratio. Moreover, GQSA significantly outperforms DC-W8A8 in inference acceleration.

Method	OPT		LLaMA-2	
	6.7B	13B	7B	13B
SliM-LoRA	47.08	47.96	54.26	57.85
DC-W8A8	48.55	-	**60.89**	-
GQSA W4S50%	**53.26**	**56.39**	59.36	**64.96**

Table 11: Performance comparison between GQSA and contemporaneous methods.

Table 11 presents the comparative evaluation of average accuracy on zero-shot tasks across different methods. The experimental results demonstrate that GQSA consistently outperforms both SliM and DC-W8A8 in terms of overall performance. Furthermore, GQSA achieves superior acceleration ratios and compression rates, while maintaining competitive accuracy. These advantages make GQSA particularly suitable for edge-side inference scenarios, where both computational efficiency and model compactness are critical.

E A comparison of the effects of pruning and quantization on inference performance

As demonstrated in Table 12, we will highlight the comprehensive performance advantages of GQSA over single pruning and quantization methods from two perspectives.

Setting	WikiText2	C4	Inference speed (ms)
0%	5.47	6.97	1490.50
S20%	7.67	9.10	1370.35
S30%	9.34	11.27	1181.25
S40%	10.84	16.38	1035.15
S50%	14.56	21.09	878.90
S60%	25.76	37.49	671.98
W8	5.50	7.01	868.35
W4	5.72	7.25	642.24
W2	36.43	40.34	475.55
W4S50%	**10.64**	**12.82**	**377.98**

Table 12: Performance comparison of GQSA with naive pruning and naive quantization in the extreme compression setting on LLaMA-2-7B.

From the Perspective of Algorithm Accuracy: Both quantization and sparsity, when applied individually, can lead to significant accuracy degradation under extreme compression settings. For instance, the PPL test results under S60% and W2 configurations demonstrate considerable performance loss. However, combining these two strategies allows for higher compression rates while better preserving model performance compared to using either strategy alone. As an example, using the LLaMA-2-7B WikiText2 benchmark, the results for W2, S60%, and W4S50% are 36.44, 25.76, and 10.64, respectively.
From the Perspective of Inference Speed: The acceleration benefit of quantization primarily arises from reduced memory access, while the acceleration benefit of sparsity stems from both memory and computational savings. For pure quantization or pure sparsity, the acceleration benefit diminishes as the compression rate increases. GQSA, however, enhances the upper limit of the acceleration benefit by simultaneously reducing redundancy in both dimensions (quantization and sparsity). For example, in the case of LLaMA-2-7B, the inference speeds for S60%, W2, and W4S50% are 671.98, 475.55, and 377.98, respectively.

F Combining the advantages of structured pruning and group quantization

As show in Table 13 the acceleration benefits of quantization primarily stem from reduced memory access, while sparsity accelerates inference by saving both memory and computation (as sparse groups do not need to be stored, read, or computed). When applying only the quantization strategy, the

LLM's acceleration benefit does not increase exponentially as the bit-width of W decreases. Instead, it faces diminishing returns, as the performance bottleneck shifts from memory access to computation as the quantization bit-width is reduced. In contrast, GQSA can further accelerate deep quantization models by skipping redundant calculations, thereby pushing the upper limit of acceleration benefits. For example, in the case of LLaMA-2-7B, the measured inference speed is 20% faster with W4S50 than with W2.

Model	Setting	Inference speed (ms)
LLaMA-2-7B	W4	642.24
	W2	475.55
	W4S50%	**377.98**

Table 13: Comparison of inference speed between GQSA and single quantization.

G Comparison of GQSA with Vector Quantization

Some of the latest low-bit quantization methods, such as AQLM (Egiazarian et al., 2024) and QuIP# (Tseng et al., 2024), employ vector quantization (VQ), which differs from uniform quantization techniques like GQSA. VQ constructs codebooks by learning the underlying data distribution, enabling better data preservation and potentially higher model performance. However, VQ methods rely on pre-trained codebooks (e.g., the E8P codebook used in QuIP# and the multi-codebook scheme in AQLM), which introduce considerable computational overhead during both training and inference. This makes them less practical for real-world deployment.

Method	WikiText2	C4	Tokens Per Second
QuIP# W2	6.06	8.07	71.09
AQLM W2	5.60	7.47	68.1
GQSA W4S50%	7.80	10.93	228.95

Table 14: Comparison between GQSA and Vector Quantization

In contrast, GQSA combines uniform quantization with high sparsity to enable efficient inference acceleration in practical scenarios. As show in Table 14, while it may slightly underperform VQ-based methods like QuIP# and AQLM in terms of accuracy, it significantly outpaces them in inference speed—achieving up to 3.3× the speed of vector quantization methods under a small accuracy trade-off. No single method perfectly balances accuracy and computational efficiency; GQSA prioritizes inference speed, accepting a minor compromise in model accuracy to achieve substantial gains in performance.

Setting	LLaMA-7B	LLaMA-13B
FP	92.69	50.68
W8	156.40	95.78
W8S50	**263.64**	**158.99**
W4	202.81	137.92
W4S50	**343.43**	**228.95**

Table 15: Inference throughput (tokens per second) of GQSA on the NVIDIA A100 80GB.

H Inference throughput of GQSA

As show in Table 15, we evaluated the throughput of the GQSA model based on FastTransformer on an Nvidia A100 80 GB GPU. The results demonstrate that, compared to the pure W8 and W4 configurations, GQSA's W8S50% and W4S50% configurations achieved a 60% improvement in throughput.

I Differences from Sparse Methods in Traditional CNNs

Although previous work, such as PatDNN, introduced semi-structured sparsity in CNN networks, we believe our work contributes to the field in two core aspects: **First**, we have significantly advanced the engineering implementation of semi-structured sparsity. Notably, we introduced the "task-centric" parallel strategy, replacing the widely-used "data-centric" parallel approach in the industry. This shift effectively addresses the issue of unbalanced load across computing units, resulting in a substantial speedup of 1.3× to 1.5× for individual operators, thus achieving a new state-of-the-art in engineering performance. **Second**, while the GEMM operator in traditional CNN networks typically adopts the "N×1" sparse mode, we propose the "1×N" sparse mode tailored to the characteristics of LLM models. This innovation better preserves outliers within the channel and is fundamentally different from the traditional "N×1" mode in terms of engineering implementation.

We believe innovation is not solely about proposing "new concepts" or "new strategies" but also about selecting the most appropriate approaches to address real technical challenges and pushing the performance boundaries. Currently, the LLM field faces significant inference cost challenges, and relying exclusively on quantization techniques has nearly reached its performance optimization limits. Our work contributes to further enhancing performance based on quantization models and has led to a SOTA breakthrough in semi-structured sparsity technology within the LLM field. The pursuit of higher performance limits and greater industrial applicability reflects a key aspect of innovation.

J The advantages of group quantization compared to standard per-layer or per-label quantization methods.

From the Perspective of Quantization Accuracy: The primary challenge in quantization LLMs arises from the imbalanced numerical distribution (both between and within channels) and the prevalence of outliers in both weights and activations. Standard per-layer and per-token (or per-channel) quantization methods assume that the entire tensor or the neurons in each channel are identically distributed. This coarser quantization granularity is insufficient to address the issues of uneven distribution and outlier retention. Group quantization, however, further partitions the channels and quantizes the model weights at a finer granularity, effectively mitigating the problem of imbalanced numerical distribution and improving outlier handling, thereby reducing the accuracy loss typically associated with quantization.

From the Perspective of Quantization Speed: The finer quantization granularity of the per-group approach necessitates additional scaling factors during computation. However, since LLM tasks are memory-intensive rather than computation-bound, this increased granularity does not significantly impact memory access complexity compared to 2:4 sparsity. As a result, the inference speed is not adversely affected. For instance, the widely used reasoning engine, llama.cpp, employs group quantization for model inference.

K Results of GQSA on the Qwen model

To verify the generalization ability of GQSA on different model families, we conduct experiments on Qwen models (base and instruct model). Table 16 shows similar results to the LLaMA model family, where GQSA consistently matches or outperforms the baseline methods even under stricter compression settings.

L Results of GQSA on the OPT model

To verify the generalization ability of GQSA on different model families, we conduct experiments on OPT models (ranging from 1.3B to 13B parameters). Table 17 shows similar results to the LLaMA and Qwen model family, where GQSA consistently matches or outperforms the baseline methods even under stricter compression settings.

M GQSA inference latency and memory consumption

Due to space constraints, detailed inference latency and model memory consumption are provided in Appendix Table 18. Overall, GQSA demonstrates exceptional performance across various settings.

Setting	Method	Qwen2.5-7B		Qwen2.5-14B		Qwen2.5-7B-Instruct		Qwen2.5-14B-Instruct	
		WikiText2	C4	WikiText2	C4	WikiText2	C4	WikiText2	C4
W2	GPTQ	29.22	89.12	23.08	55.43	46.03	289.92	48.42	38.37
	OmniQuant	14.49	22.78	11.98	17.81	17.26	26.37	12.95	17.97
2:4	sparsegpt	**11.25**	17.17	10.13	**15.39**	11.92	17.85	10.95	16.24
	wanda	14.78	22.84	11.74	18.24	15.80	23.83	12.06	18.73
w4s20%	GQSA	8.27	12.74	6.83	10.83	7.99	12.21	6.80	10.76
w4s30%		8.95	13.66	7.75	12.03	9.01	13.78	7.69	11.91
w4s40%		9.70	14.96	8.97	13.81	10.19	15.64	8.90	13.60
w4s50%		11.71	**17.02**	**9.87**	15.93	**11.74**	**17.07**	**10.81**	**15.97**

Table 16: Wikitext2 and C4 perplexity (↓) for Qwen2.5 models, with a context length of 2048.

Setting	Method	OPT-1.3B		OPT-2.7B		OPT-6.7B		OPT-13B	
		WikiText2	C4	WikiText2	C4	WikiText2	C4	WikiText2	C4
W2	GPTQ	130.88	60.88	61.59	33.83	20.18	18.55	21.36	16.34
	QUIP	41.64	-	28.98	-	18.57	-	16.02	-
	PB-LLM	45.92	-	39.71	-	20.37	-	19.11	-
	OmniQuant	23.95	27.33	18.13	21.11	14.43	16.67	12.94	14.92
	SliM-LLM	24.57	-	17.98	-	14.22	-	12.16	-
2:4	SparseGPT	24.54	26.55	17.82	**19.45**	14.23	**16.56**	12.94	**14.88**
	Wanda	28.27	28.54	21.17	22.84	15.90	18.99	15.55	16.18
W4S20%	GQSA	14.49	16.60	12.03	14.54	10.21	12.71	9.93	12.16
W4S30%		16.06	18.44	13.23	15.95	10.94	13.64	10.37	12.85
W4S40%		18.82	21.54	15.39	18.25	12.15	15.12	11.29	13.97
W4S50%		**21.32**	**24.90**	**17.52**	20.81	**13.44**	16.94	**12.16**	15.57

Table 17: Wikitext2 and C4 perplexity (↓) for OPT models, with a context length of 2048.

LLaMA-7B								
sequence length	128		256		512		1024	
	Latency (ms)	Memory (GB)	Latency (ms)	Memory (GB)	Latency (ms)	Memory (GB)	Latency (ms)	Memory (GB)
fp16	1490.5	13.47	3005.95	13.534	6090.97	13.662	12561.82	13.918
w8a16	868.35	7.394	1755.62	7.458	3594.95	7.586	7559.22	7.842
w8a16+sp0.3	688.89	6.296	1261.05	6.361	3005.02	6.489	5814.62	6.745
w8a16+sp0.4	603.23	5.669	1103.08	5.733	2593.76	5.861	5039.33	6.117
w8a16+sp0.5	512.71	5.042	996.59	5.106	2019.1	5.234	4329.32	5.492
w4a16	642.24	4.258	1312.91	4.322	2707.26	4.45	5786.8	4.706
w4a16+g16+sp0.3	518.99	4.101	1041.18	4.165	2113.56	4.293	4437.48	4.549
w4a16+g16+sp0.4	432.05	3.788	855.46	3.852	1828.48	3.977	3772.63	4.233
w4a16+g16+sp0.5	377.98	3.474	699.26	3.528	1433.43	3.653	3110.54	3.909
LLaMA-13B								
sequence length	128		256		512		1024	
	Latency (ms)	Memory (GB)	Latency (ms)	Memory (GB)	Latency (ms)	Memory (GB)	Latency (ms)	Memory (GB)
fp16	2726.66	25.61	5481.96	25.696	11071.81	25.92	22559.77	26.304
w8a16	1439.66	13.524	2900.46	13.62	5922.28	13.844	12257.27	14.228
w8a16+sp0.3	1164.239	11.396	2114.165	11.492	4976.471	11.716	9501.55	12.105
w8a16+sp0.4	1024.199	10.182	1843.61	10.278	4272.72	10.502	8343.77	10.886
w8a16+sp0.5	869.486	8.964	1715.976	9.061	3345.762	9.285	7044.25	9.669
w4a16	999.1	7.444	2020.99	7.54	4155.94	7.764	8750.98	8.148
w4a16+g16+sp0.3	801.203	7.141	1475.175	7.237	3563.465	7.461	6972.12	7.845
w4a16+g16+sp0.4	702.602	6.532	1303.865	6.628	3087.623	6.852	6081.292	7.236
w4a16+g16+sp0.5	603.515	5.924	1104.366	6.02	2374.286	6.244	5099.638	6.628
LLaMA-30B (TP=2)								
sequence length	128		256		512		1024	
	Latency (ms)	Memory (GB)	Latency (ms)	Memory (GB)	Latency (ms)	Memory (GB)	Latency (ms)	Memory (GB)
fp16	3759.08	65.534	7540.17	65.726	15241.36	66.11	31073.23	66.878
w8a16	3032.64	32.418	6111.43	32.642	12371.66	33.026	25477.58	33.794
w8a16+g16+sp0.3	2412.6	27.084	4861.575	27.308	10343.645	27.692	19826.459	28.46
w8a16+g16+sp0.4	2132.65	24.036	3840.98	24.261	8925.685	24.645	17343.09	25.413
w8a16+g16+sp0.5	1797.27	20.988	3472.16	21.212	6950	21.596	14591.638	22.364
w4a16	1938.2	17.178	3924.2	17.402	8011.57	17.786	16680.64	18.554
w4a16+g16+sp0.3	1541.925	16.416	2515.512	16.641	6800.993	17.025	13290.836	17.793
w4a16+g16+sp0.4	1341.315	14.892	2229.65	15.116	5890.861	15.501	11591.38	16.269
w4a16+g16+sp0.5	1122.292	13.368	2180.11	13.592	4526.311	13.816	9720.279	14.584

Table 18: Inference latency and memory usage of the FastTransformer implementation on NVIDIA A800-40GB GPU with a fixed input sequence length of 15, output sequence lengths of 128, 256, 512 and 1024.