

# Consensus Sparse Attention: A Memory and Computation Efficient Mechanism Based on Inter-Head Consensus

Anonymous ACL submission

## Abstract

Large language models have achieved state-of-the-art performance across a wide range of NLP tasks, but their deployment is constrained by the attention mechanism’s quadratic scaling with sequence length, leading to extensive memory requirements. We propose Consensus Sparse Attention (CSA), an efficient and lightweight optimization method that enhances attention calculation performance while maintaining model accuracy, without requiring additional post-training. CSA determines a few representative attention heads to identify a consensus set of salient tokens. These selected tokens are then shared across all remaining heads. This mechanism significantly reduces both computational cost and memory consumption while preserving the model’s contextual understanding and information. CSA integrates seamlessly into existing attention architectures and requires no further adaptation. Experimentally, CSA delivers 2× inference speedup and 50% lower peak memory usage while maintaining 99.7% accuracy on LLaMA-3, 99.8% on Qwen2, and 99% on the Needle In A Haystack benchmark for long-context understanding.

## 1 Introduction

Large Language Models (LLMs) have achieved impressive performance across a broad spectrum of challenging tasks, owing to their ability to model complex dependencies and relationships between tokens (i.e., embedded data) (Zhao et al., 2023; Chang et al., 2024). Their capabilities extend to diverse applications, including multimodal integration (Cha et al., 2024), relation extraction (Wan et al., 2023), code generation (Zhong and Wang, 2024), and agent-based decision-making (Li et al., 2023).

Despite these advances, the dense attention mechanism at the core of LLMs remains a significant barrier to efficient deployment. Its computational and memory complexity scales quadrati-

cally with input sequence length, leading to prohibitive costs as model size and input sequences increase. To mitigate this issue, prior work has explored various sparse attention mechanisms, including Local (Child et al., 2019), Global (Beltagy et al., 2020), Hybrid (Zaheer et al., 2020), Predicted Token Dominated (Tang et al., 2024), and Explicit Sparse Transformers (Zhao et al., 2019).

While these methods exploit the observation that attention scores are often concentrated on a small subset of tokens, many still rely on computing full attention scores or require model post-training, limiting their efficiency in practice. For example, SparQ (Ribar et al., 2023) selects the top- $r$  components of each query vector to extract corresponding components from the salient tokens along the hidden dimension, thereby effectively approximating the attention scores. While this compression reduces computational overhead, the resulting attention matrix still retains the same shape  $[b, h, s, s]$  as in dense attention, offering limited improvements in terms of peak memory usage.

Our approach is motivated by the empirical insight that multiple attention heads often focus on the same subset of certain tokens. Specifically, these particular tokens consistently receive high attention scores across different heads, suggesting a shared notion of salience, illustrated in Figure 1. Building on this finding and motivation, we propose **Consensus Sparse Attention (CSA)**, a novel mechanism for multi-head attention. CSA computes attention scores first using a small subset of  $g$  selected representative heads to identify a consensus set of salient tokens, which are then used by the remaining heads to finish individual attention scores. By restricting attention score computation to only  $[b, g, s, s]$  shape (where  $g \ll h$ ), CSA effectively reduces both computational cost and peak memory usage in the attention layer. On standard benchmark evaluations, CSA achieves 2x inference speedup and 50% reduction in peak memory usage

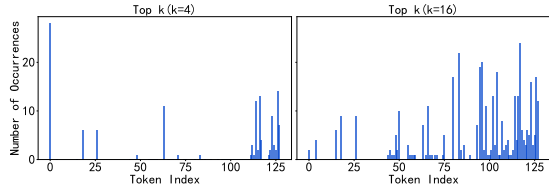


Figure 1: The figure presents the distribution of token occurrences for  $k = 4$  (left) and  $k = 16$  (right). Each bar represents the number of occurrences for a top  $k$  token index. The shared y-axis indicates the frequency of token occurrences, facilitating a direct comparison between the two  $k$  values.

during attention computation. CSA can also seamlessly integrate into existing attention-based model structures without any post-training and without sacrificing model accuracy, such as maintaining 99.7% accuracy on LLaMA-3, 99.8% on Qwen2, and 99% accuracy on long-context understanding. In summary, our contributions are as follows:

- We uncover a consistent consensus among attention heads in selecting salient tokens and validate this phenomenon through empirical analysis.
- We propose Consensus Sparse Attention (CSA), a novel attention mechanism that is fully compatible with existing Transformer architectures. CSA significantly reduces computational and memory costs during the attention score computation stage.
- We evaluate CSA through extensive experiments, showing that it outperforms existing baselines and validating its effectiveness.

## 2 Related Work

Both SparQ and CSA aim to avoid the full computation of attention scores. Therefore, we introduced SparQ for a horizontal comparison and used top- $p$  as the theoretical upper limit for comparison with SparQ and CSA.

### 2.1 Efficient LLMs Inference

LLMs usually require a higher inference cost when processing large amounts of queries, which poses a huge challenge for their deployment. To improve the inference efficiency of LLMs, some current works optimize two important parts in the model, Feed Forward Network (FFN)(Zhang et al., 2021; Gao et al., 2022; Komatsuzaki et al., 2022) and Attention Operation(Shazeer, 2019; Ainslie et al., 2023; Ma et al., 2021), by designing efficient structures or strategies. Some other works

consider applying classical scheduling strategies in the query batching process to handle asynchronous queries more quickly, such as FCFS(Yu et al., 2022), Multi-Level Feedback Queue(Wu et al., 2023) and Continuous Batching(Kwon et al., 2023). Besides, in model compression, quantization is a commonly used method. It reduces the computational and memory costs of LLMs by converting model weights and activations from high to low bit-widths, such as GPTQ(Frantar et al., 2022) minimizes the difference in model output before and after quantization by using a small portion of calibration data for the weight matrix of each layer, AWQ(Lin et al., 2024)selects salient weights based on the activation distribution. Also, some methods (Frantar and Alistarh, 2023; Sun et al., 2023; Kurtic et al., 2022) prune the model parameters, or (Gu et al., 2023; Hsieh et al., 2023; Shridhar et al., 2022) compress the model volume by distilling knowledge into a smaller one. The method studied in this paper is closely related to sparsity in model compression and focuses on the bottleneck of dense self-attention in the inference process.

### 2.2 Sparse Attention Compression

Due to the sparsity of the self-attention matrix, extracting the important parts from it has always been an active research field. For example, methods like Local(Child et al., 2019; Ren et al., 2021), Global(Beltagy et al., 2020), Hybrid(Zaheer et al., 2020) improve the computational efficiency of attention scores by choosing random, adjacent or specially marked tokens during long context processing. LM-Infinite(Han et al., 2023) and StreamingLLM(Xiao et al., 2023) adopt some fixed sparse patterns to select the latest and important tokens. Explicit sparse transformer(Zhao et al., 2019) and FlexGen(Sheng et al., 2023) identify important tokens through the attention scores and Tang et al.(Tang et al., 2024) link the selection of important attention scores and the currently predicted token together. GQA(Ainslie et al., 2023) groups the query heads and shares the parameters of key heads and value heads within each group. Unlike GQA, DHA(Chen et al., 2024) decouples key heads and value heads and allocates different numbers of key heads and value heads across different layers to achieve a better balance between performance and efficiency. DeJaVu(Liu et al., 2023) predicts efficient heads in layer  $L+1$  based on observations from layer  $L$ . CHAI(Agarwal et al., 2024) clusters similar heads and replaces all heads in a cluster

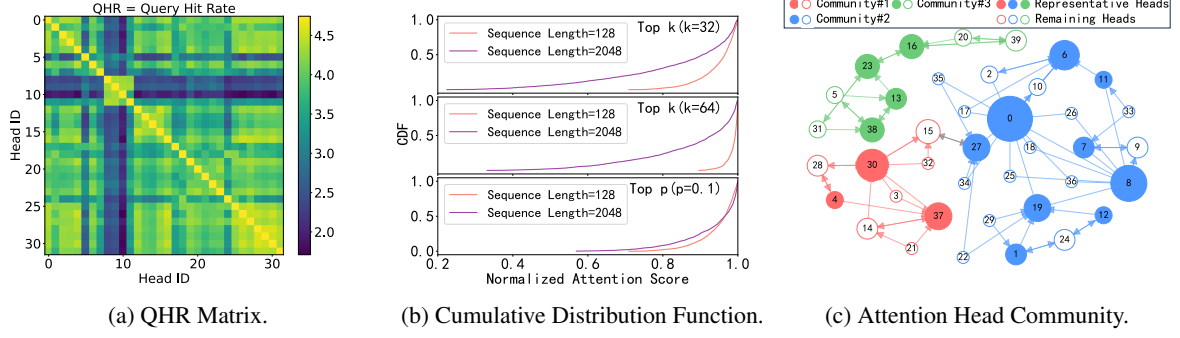


Figure 2: Subfigure (a) QHR Matrix illustrates the QHR scores between any two attention heads. Subfigure (b) presents the distribution of Cumulative Distribution Function (CDF) scores for the top salient tokens at sequence lengths of 128 and 2048. Subfigure (c) demonstrates the results of community division and the selection of representative attention heads; the size of each node indicates its in-degree. The directed edges represent that the destination node has an advantage over the source node in terms of being among the top  $p$  tokens.

with a single representative head. Both approaches perform pruning at the attention head level. That means in these methods, only a limited number of heads contribute to the final attention score calculation with all input tokens. Meanwhile, the eviction strategy maintains a certain size by continuously deleting irrelevant tokens. H<sub>2</sub>O(Zhang et al., 2024) maintains a budget space of size  $k$  by accumulating historical attention weight scores. TOVA(Oren et al., 2024) discards tokens with lower attention scores based on the current query. SparQ(Ribar et al., 2023) reduces the memory bandwidth during the computation process by selecting salient query tokens before computing the attention weights. FastGen(Ge et al., 2023) formulates separate compression strategies for them respectively based on the observation of different heads. Unlike the aforementioned methods, our approach achieves sparse attention by leveraging the consensus among attention heads on certain salient tokens.

### 3 Motivation

In well-trained Transformer-based models, it is commonly observed that most tokens receive negligible attention scores, whereas only a small subset consistently captures high attention. As shown in Figure 1, attention heads are concentrated on a few salient tokens, with stronger concentration for more salient tokens, suggesting that these tokens have a large impact on the model’s output. Figure 2b also supports that the attention distribution in dense models, such as LLaMA-3, is heavily skewed toward a small number of tokens. These findings indicate that focusing solely on these salient tokens might suffice to preserve essential contextual infor-

mation, thereby maintaining the model’s accuracy.

**Challenges with Top- $k$  Token Selection.** Existing approaches such as Explicit Sparse Transformer (Zhao et al., 2019) and SparQ (Ribar et al., 2023) improve inference efficiency by replacing the dense attention mechanism with sparse attention that selects the Top- $k$  salient tokens based on attention scores. While effective in reducing computation, this fixed-size selection strategy becomes problematic as input sequence length increases. As illustrated in Figure 2b, when input sequence length increases from 128 to 2048, attention scores over the Top-32 or Top-64 token selection methods become more diffuse. This means that selected Top- $k$  tokens capture a smaller proportion of total attention, diminishing their utility. Simply increasing  $k$  does not solve this issue, as the score distribution continues to flatten with longer sequences. This makes Top- $k$  selection methods less suitable for long-context scenarios.

**Advantages of Top- $p$  Token Selection.** To address the limitations of fixed size in Top- $k$  selections, we explore a percentage-based alternative - the Top- $p$  method. Instead of selecting a fixed number of tokens, Top- $p$  dynamically selects a proportion of salient tokens based on attention scores. This adaptive approach better preserves overall attention scores across varying sequence lengths. As shown in Figure 2b, Top- $p$  ( $p=10\%$ ) selection method maintains a stable distribution of attention scores, even as sequence length scales from 128 to 2048. By selecting a proportional rather than a fixed-size number of salient tokens, the model retains the most contextual information more effectively.

## 4 Our Approach

Traditional Top- $p$  sparse attention methods typically require computing the full attention matrix before selecting the Top- $p$  proportion of salient tokens with the highest scores. While this reduces the number of attended tokens, it does not eliminate the overhead of computing dense attention scores, which remain a major bottleneck in both computation and memory usage.

In contrast, our proposed method - Consensus Sparse Attention (CSA) - eliminates the need to compute the full attention scores. In each attention layer, by computing attention for only a small number of representative heads, CSA predicts the indices of high-importance salient tokens that are then shared across all heads. This consensus-driven approach allows us to preselect Top- $p$  salient tokens efficiently, significantly reducing computational and memory costs while preserving model performance.

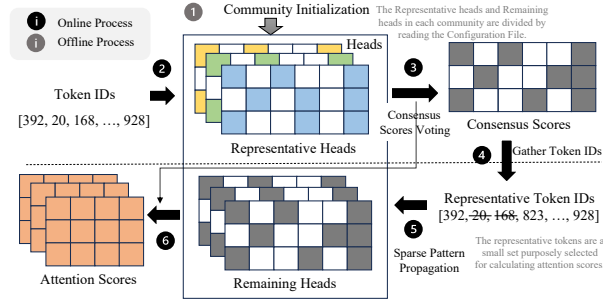


Figure 3: CSA Workflow.

### 4.1 Consensus Sparse Attention

The rationale of CSA is to reduce attention computation by exploiting redundancy across attention heads. CSA first categorizes attention heads into *Communities* based on their similarity in attention behavior. Within each community, a small number of representative heads (denoted as  $g_i$ ) are selected to predict the token indices based on consensus scores for the others.

These representative heads are used to identify a shared set of salient tokens, which are then propagated to the remaining heads in the same community. Instead of computing full attention scores individually for every head on every token, this strategy allows CSA to only compute attention scores on limited salient tokens, thereby significantly reducing computational overhead and memory usage, especially in lone-sequence scenarios.

### Algorithm 1 Consensus Sparse Attention

**Input:**  $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{b \times h \times s \times d}$ , Selection Parameters  $p$ , Head Community  $\mathcal{C}$   
**Output:** Attention output  $\mathbf{O} \in \mathbb{R}^{b \times h \times s \times d}$

- 1: Initialize output tensor  $\mathbf{O}$
- 2: **for** each community  $c \in \mathcal{C}$  **do**
- 3:   Categorize heads of community  $c$  into the representative  $g_i$  heads and the remaining  $h_{c-g_i}$  heads
- 4:   Extract  $\mathbf{Q}_{g_i}, \mathbf{K}_{g_i}$  from representative heads
- 5:    $\mathbf{S}_{g_i} \leftarrow \text{softmax} \left( \frac{\mathbf{Q}_{g_i} \mathbf{K}_{g_i}^\top}{\sqrt{d}} + \text{Mask} \right)$
- 6:    $\mathbf{O}_{g_i} \leftarrow \mathbf{S}_{g_i} \mathbf{V}_{g_i}[:, p]$
- 7:    $\mathbf{S}_{g_i} \leftarrow \mathbf{S}_{g_i} \odot \text{Mask}_p$  /\* mask non Top- $p$  tokens (step 2 in Fig 3) \*/
- 8:    $\hat{\mathbf{S}}_c \leftarrow \sum_{g_i} \mathbf{S}_{g_i}$  /\* consensus scores voting (step 3 in Fig 3) \*/
- 9:    $\mathcal{P}_{\text{select}} \leftarrow \text{top}_p(\hat{\mathbf{S}}_c)$  /\* find Top- $p$  indices (step 4 in Fig 3) \*/
- 10:    $\mathbf{K}_{c-g_i}, \mathbf{V}_{c-g_i} \leftarrow \text{gather}(\mathcal{P}_{\text{select}})$  /\* step 5 in Fig 3 \*/
- 11:    $\mathbf{S}_{c-g_i} \leftarrow \text{softmax} \left( \frac{\mathbf{Q}_{c-g_i} \mathbf{K}_{c-g_i}^\top}{\sqrt{d}} \right)$
- 12:    $\mathbf{O} \leftarrow \mathbf{O} + \mathbf{S}_{c-g_i} \mathbf{V}_{c-g_i} + \mathbf{O}_{g_i}$  /\* step 6 in Fig 3 \*/
- 13: **end for**
- 14: **return**  $\mathbf{O}$

As illustrated in Figure 3, CSA consists of the following five primary phases:

1. **Community Initialization:** Community initialization reads the community information required by the CSA online process from an offline-generated configuration file. The generation of the configuration file, as shown in Figure 4, primarily consists of two parts: Head Community Clustering and Representative Heads Selection.

- (a) **Head Community Clustering:** Different attention heads often exhibit similar behavior, tending to focus on similar contextual information. To exploit this redundancy, CSA groups attention heads into communities based on their similarity, identified using a clustering algorithm (in §4.2).

- (b) **Representative Heads Selection:** In each community, a subset of heads is designated as representative heads (in §4.3). These representative heads are responsible for identifying high-importance salient token indices, which are then shared with the remaining heads in the community, thereby reducing computational effort without compromising performance.

2. **Consensus Scores Voting:** To enforce sparsity, CSA masks the non Top- $p$  elements/tokens in the attention matrices of the representative heads, thereby generating Top- $p$  vot-



311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329

- 317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329

330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341



After calculating the QHR matrix(line 1 in Algorithm 2), we identify the most representative heads for each attention head  $h_m$  by selecting the  $j$  heads with the highest QHR values. These selected heads statistically best represent  $h_m$ . To model this relationship, we construct a directed graph by drawing  $j$  directed edges from  $h_m$  to the selected heads(line 2 in Algorithm 2), as illustrated in Figure 2c.

In this graph, certain nodes naturally accumulate a higher in-degree than others, indicating that they are recognized as representative attention heads by a larger number of nodes. To identify the most influential heads, we first select the attention head node with the highest in-degree and remove it from the community, along with its associated edges. This process is repeated iteratively until the required number of representative heads is obtained for each community(line 7-10 in Algorithm 2). Finally, we select the  $g$  attention heads based on descending in-degree. These  $g$  heads are deemed the representative attention heads. The complete selection procedure is described in Algorithm 2.

To ensure that the representative heads effectively capture long-context information, each community is proportionally assigned  $g_i$  representative heads, guaranteeing that representatives are distributed across all communities.

#### 4.4 Complexity Analysis

**Computational Complexity.** The standard attention mechanism primarily involves  $QK^T$  Matrix Multiplication and Value Projection. The  $QK^T$  includes  $\mathcal{O}(2bhs^2d)$  operations for  $[b, h, s, d] \times [b, h, d, s]$  tensor contraction. The Value Projection also includes  $\mathcal{O}(2bhs^2d)$  operations for  $[b, h, s, s] \times [b, h, s, d]$  tensor product. In total, it is  $\mathcal{O}(4bhs^2d)$ .

The CSA attention mechanism introduces a sparse-based calculation, reducing the computational complexity. For  $QK^T$ , CSA experiences  $\mathcal{O}(2bgs^2d)$  and  $\mathcal{O}(2b(h-g)ps^2d)$  operations for representative and remaining heads, respectively. For Value Projection, CSA reduces to  $\mathcal{O}(2bhps^2d)$  operations. The total operations in CSA are  $\mathcal{O}(2bgs^2d + 2b(h-g)ps^2d + 2bhps^2d)$ .

Due to  $g \ll h$  and  $p$  being a percentage less than 1, CSA reduces computational cost by a factor of  $\frac{g+(h-g)p+hp}{2h}$ . For example, when  $h$  is 32,  $g$  is 8, and  $p=0.1$ , it saves 79%.

**Memory Complexity.** Memory constraints are a major barrier to deploying LLMs in resource-

limited environments. CSA addresses this challenge by leveraging sparsity in its attention calculation, significantly reducing peak memory usage during the attention computation process. In the default attention mechanism, the peak memory usage is  $\mathcal{O}(bhs^2)$  for storing the full attention matrix  $[b, h, s, s]$ . CSA reduces the peak memory usage to  $\mathcal{O}(\max(bgs^2, b(h-g)ps^2))$  where  $\mathcal{O}(bgs^2)$  memory is for representative heads and  $\mathcal{O}(b(h-g)ps^2)$  the remaining heads.

Typically,  $g > (h-g)p$  holds. CSA can reduce the peak memory by a factor of  $\frac{g}{h}$  through  $\frac{bgs^2}{bhs^2}$ .

## 5 Evaluation

We address the following research questions in our experiments:

- RQ1: How does the CSA method perform compared to baseline approaches across various NLP tasks and model scales?
- RQ2: To what extent do the Representative Heads Selection and Heads Community Clustering mechanisms contribute to the effectiveness of CSA?
- RQ3: Can CSA maintain consistent performance as the input sequence length increases?
- RQ4: Does CSA offer substantial improvements in inference speed and memory efficiency?

### 5.1 Environment Setting

We assess memory and time for various batch sizes and sequence lengths on core matrix computations with aligned PyTorch implementations (see §A.4).

**Benchmark Suites.** To evaluate CSA in NLP tasks, we use the OpenCompass(Contributors, 2023) framework: MMLU(Hendrycks et al., 2020) and Ceval(Huang et al., 2023) assess language capabilities, HumanEval(Chen et al., 2021) assesses coding, GSM8K(Cobbe et al., 2021) evaluates math reasoning, TriviaQA(Joshi et al., 2017) checks knowledge, SQuAD2.0 tests reading comprehension, and 'needle-in-a-haystack'(Li et al., 2024) handles long-text processing.

**Models.** We tested the CSA on popular open-source models like Qwen2 14B-Chat(Yang et al., 2024), LLaMA-3-8B-Chat(Grattafiori et al., 2024), and LLaMA-3-70B-Chat(Grattafiori et al., 2024). For long-text experiments, we used the ChatGLM-6B-32k(GLM et al., 2024) model, specifically trained for this purpose.

**Baselines.** We selected vanilla Top- $p$  and SparQ (Ribar et al., 2023) as baseline methods. SparQ approximates attention scores by selecting key vector components and then identifies salient tokens based on these approximated scores. Note: To ensure a fair comparison, we modified SparQ to use a Top- $p$  method like CSA and enabled it to function in the prefilling stage, as CSA supports both stages.

## 5.2 Results

### 5.2.1 Performance on NLP Tasks (RQ1)

Table 1 presents the performance of CSA under the condition of  $p = 0.1$  across various model sizes. The experimental results demonstrate that CSA consistently outperforms the baselines across almost all datasets, closely approaches the theoretical upper bound of the Top- $p$  method, and performs comparably to the dense model. Remarkably, on certain datasets, our method even surpasses the dense model. We attribute this improvement to the Top- $p$  mechanism, which effectively filters out low-attention tokens, thereby concentrating attention on more relevant tokens and enhancing model performance.

Additionally, Table 2 presents a comparison of baseline methods on the PIQA and HellaSwag datasets, demonstrating that CSA maintains the original accuracy performance without any degradation.

### 5.2.2 Representative Heads Selection (RQ2)

We evaluated CSA’s representative head selection against random selection. Table 3 shows CSA consistently outperforms random selection, with Figure 5 revealing differences in consensus voting (Random vs. CSA). Random selection reduces low-hit regions via consensus score voting, whereas CSA further compresses low-value areas and increases high-value density, demonstrating its superior capability in selecting representative attention heads and ensuring stable voting. These results confirm CSA’s representative head selection enhances Top- $p$  token prediction stability and improves performance across datasets by optimizing consensus voting quality.

### 5.2.3 Community Size (RQ2)

To assess how community size impacts model performance and validate our parameter choices for  $g$  and  $c$ , we conducted experiments outlined in Table 4. The results show that community partitioning boosts CSA performance. However, too many

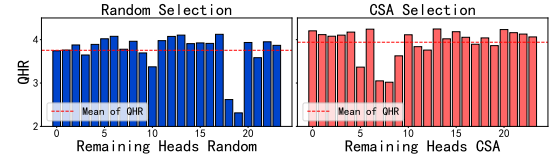
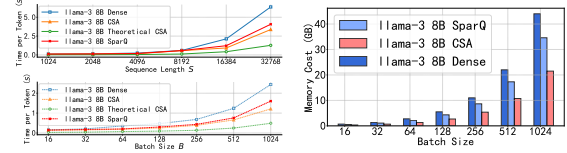


Figure 5: Representative Head Selection Strategies: Random vs. CSA.



(a) Long Sequence Test (b) Peak Memory Usage.

Figure 6: (a) and (b) are both set on LLaMA3-8B. (a) set batch size to 1 and (b) set sequence length to 512.

partitions do not always improve results, as they reduce sampled attention heads per community and lead to reliance on dominant heads. Similarly, an excessively high number of heads offers minimal performance benefits once the optimal number is reached and significantly increases computational costs. Thus, a careful balance of community number and head selection is essential for optimizing model accuracy and efficiency.

### 5.2.4 Sequence Length Scaling (RQ3)

We tested CSA on long-sequence problems using the "Needle in a Haystack" benchmark. We created a single lengthy context by concatenating SQuAD dataset contexts and inserted a "text needle" at a specific depth. CSA’s ability to handle long texts was evaluated by its effectiveness in finding this needle. For consistency, we followed methods in (gkamradt, 2023) and (Contributors, 2023). As shown in Table 5, CSA exhibited robust stability with increasing sequence lengths from 4k to 16k.

### 5.2.5 Computational Efficiency and Memory Optimization(RQ4)

We evaluated attention operations in layers using PyTorch on A100 80G. According to Figure 6a, CSA achieves nearly double the speed of dense models in high-batch, long-text scenarios, indicating effective sparsity. By skipping full attention matrix computation, CSA greatly cuts memory use. Experiments with fp16 precision in Figure 6b, show up to 50% memory savings, enabling larger batch sizes and efficient inference, making CSA ideal for resource-limited settings.

Models	Methods	DataSets						
		MMLU	HumanEval@5	Ceval	Gsm8k	TriviaQA	SQuAD	Avg
LLaMA-3 8B	Dense	63.2	73.0	54.0	78.9	75.4	53.0	66.2
	Top	63.4	73.1	53.2	79.2	74.8	52.9	66.1
	SparQ	61.7	72.5	51.5	76.5	72.9	49.4	64.1
	CSA	<b>63.2</b>	<b>73.0</b>	<b>52.7</b>	<b>78.2</b>	<b>74.9</b>	<b>52.5</b>	<b>65.8</b>
Qwen2 14B	Dense	49.6	74.4	62.9	65.5	65.3	21.2	56.5
	Top	50.0	75.6	62.7	65.6	65.1	20.3	56.5
	SparQ	47.1	74.4	62.2	<b>65.9</b>	65.1	<b>20.3</b>	55.8
	CSA	<b>50.1</b>	<b>75.0</b>	<b>62.4</b>	65.6	<b>65.1</b>	20.0	<b>56.4</b>
LLaMA-3 70B	Dense	77.5	84.1	67.5	92.6	88.7	56.9	77.9
	Top	77.9	84.7	66.3	92.3	88.7	56.8	77.8
	SparQ	75.4	<b>84.1</b>	58.3	89.9	88.3	52.0	74.6
	CSA	<b>77.8</b>	83.9	<b>66.3</b>	<b>92.4</b>	<b>88.7</b>	<b>56.8</b>	<b>77.7</b>

Table 1: Performance on NLP tasks. We used Pass@5 metric on HumanEval and Accuracy metric on the others. 'Dense' method represents the Dense model, 'Top' method represents the vanilla Top- $p$ , 'SparQ' is one of baselines, and CSA represents our proposed method.

Method	PIQA	HellaSwag
DHA-7B-50%	0.979	0.935
DHA-7B-25%	0.962	0.886
GQA-7B-50%	0.983	0.835
GQA-7B-25%	0.951	0.951
DejaVu-10%	0.951	0.938
DejaVu-30%	0.833	0.756
DejaVu-50%	0.691	0.333
SpAtten	0.481	0.441
CHAI-static	0.950	0.943
CHAI	0.975	0.957
CSA	<b>1.000</b>	<b>0.996</b>

Table 2: Performance on the PIQA and HellaSwag benchmarks. Normalized results: values above 1 show improvements over the baseline, while below 1 indicate degradation. Higher values mean better performance.

Models	Methods	DataSet		
		MMLU	HumanEval@5	Gsm8k
LLaMA-3 8B	Random	62.6	72.5	77.8
	CSA	<b>63.2</b>	<b>73.0</b>	<b>78.2</b>
Qwen2 14B	Random	47.7	71.0	62.1
	CSA	<b>50.1</b>	<b>75.0</b>	<b>65.6</b>
LLaMA-3 70B	Random	76.0	82.6	92.0
	CSA	<b>77.8</b>	<b>83.9</b>	<b>92.4</b>

Table 3: Head Selection Strategies: Random vs. CSA.

## 6 Conclusion

In this work, we propose Consensus Sparse Attention (CSA), a novel mechanism designed to accelerate the inference of large language models (LLMs). CSA leverages consensus voting among representative attention heads, enabling efficient prediction of salient tokens across the remaining heads. By adopting a Top- $p$  token selection strategy, CSA effectively mitigates the decline in attention concentration commonly observed in long-text scenarios.

Models	Community Size					
	g4 c1	g8 c1	g8 c2	g16 c1	g16 c2	g32 c4
LLaMA-3 8B	62.4	63.2	63.0	63.2	63.2	63.3
Qwen2 14B	41.7	48.2	39.8	49.0	50.1	50.3
LLaMA-3 70B	76.1	76.9	76.5	77.2	77.8	77.9

Table 4: Impact of  $g$  and  $c$  on head community clustering.

Methods	0k - 4k	4k - 8k	8k - 12k	12k - 16k
Dense	100	100	99.7	99.6
Top	100	100.0	99.5	99.2
SparQ	99.5	99.3	<b>99.1</b>	99.0
CSA	<b>100</b>	<b>99.3</b>	99.0	<b>99.0</b>

Table 5: Sequence Length Scaling.

This consensus-driven approach not only reduces computational load but also significantly lowers peak memory consumption, resulting in substantial efficiency gains.

## 7 Limitations

When calculating attention, CSA requires computing the representative attention heads and the remaining attention heads sequentially. This serial processing may underutilize the machine’s computational resources, particularly in low-batch, short-text scenarios.

## References

Saurabh Agarwal, Bilge Acun, Basil Hosmer, Mostafa Elhoushi, Yejin Lee, Shivaram Venkataraman, Dimitris Papailiopoulos, and Carole-Jean Wu. 2024. Chai:



579	Clustered head attention for efficient llm inference.	Ze-Feng Gao, Peiyu Liu, Wayne Xin Zhao, Zhong-	632
580	<i>arXiv preprint arXiv:2403.08058</i> .	Yi Lu, and Ji-Rong Wen. 2022. Parameter-efficient	633
581	Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury	mixture-of-experts architecture for pre-trained lan-	634
582	Zemlyanskiy, Federico Lebrón, and Sumit Sanghai.	guage models. <i>arXiv preprint arXiv:2203.01104</i> .	635
583	2023. Gqa: Training generalized multi-query trans-		
584	former models from multi-head checkpoints. <i>arXiv</i>	Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang,	636
585	<i>preprint arXiv:2305.13245</i> .	Jiawei Han, and Jianfeng Gao. 2023. Model tells you	637
		what to discard: Adaptive kv cache compression for	638
		llms. <i>arXiv preprint arXiv:2310.01801</i> .	639
586	Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020.	gkamradt. 2023. Llmtest needle in a haystack - pressure	640
587	Longformer: The long-document transformer. <i>arXiv</i>	testing llms. <a href="https://github.com/gkamradt/LLMTest_NeedleInAHaystack">https://github.com/gkamradt/</a>	641
588	<i>preprint arXiv:2004.05150</i> .	<a href="https://github.com/gkamradt/LLMTest_NeedleInAHaystack">LLMTest_NeedleInAHaystack</a> .	642
589	Junbum Cha, Wooyoung Kang, Jonghwan Mun, and	Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chen-	643
590	Byungseok Roh. 2024. Honeybee: Locality-	hui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Han-	644
591	enhanced projector for multimodal llm. In <i>Proceeed-</i>	lin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Ji-	645
592	<i>ings of the IEEE/CVF Conference on Computer Vi-</i>	adai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie	646
593	<i>sion and Pattern Recognition</i> , pages 13817–13827.	Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu,	647
		Lucen Zhong, Mingdao Liu, Minlie Huang, Peng	648
594	Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu,	Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shu-	649
595	Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi,	dan Zhang, Shulin Cao, Shuxun Yang, Weng Lam	650
596	Cunxiang Wang, Yidong Wang, et al. 2024. A sur-	Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan	651
597	vey on evaluation of large language models. <i>ACM</i>	Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu,	652
598	<i>Transactions on Intelligent Systems and Technology</i> ,	Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan	653
599	15(3):1–45.	An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li,	654
600	Mark Chen, Jerry Tworek, Heewoo Jun, Qiming	Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang,	655
601	Yuan, Henrique Ponde De Oliveira Pinto, Jared Ka-	Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan	656
602	plan, Harri Edwards, Yuri Burda, Nicholas Joseph,	Wang. 2024. <a href="#">Chatglm: A family of large language</a>	657
603	Greg Brockman, et al. 2021. Evaluating large	<a href="#">models from glm-130b to glm-4 all tools</a> . <i>Preprint</i> ,	658
604	language models trained on code. <i>arXiv preprint</i>	<i>arXiv:2406.12793</i> .	659
605	<i>arXiv:2107.03374</i> .		
606	Yilong Chen, Linhao Zhang, Junyuan Shang, Zhenyu	Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri,	660
607	Zhang, Tingwen Liu, Shuhuan Wang, and Yu Sun.	Abhinav Pandey, Abhishek Kadian, Ahmad Al-	661
608	2024. Dha: Learning decoupled-head attention from	Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten,	662
609	transformer checkpoints via adaptive heads fusion.	Alex Vaughan, et al. 2024. The llama 3 herd of mod-	663
610	<i>arXiv preprint arXiv:2406.06567</i> .	els. <i>arXiv e-prints</i> , pages arXiv–2407.	664
611	Rewon Child, Scott Gray, Alec Radford, and	Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang.	665
612	Ilya Sutskever. 2019. Generating long se-	2023. Knowledge distillation of large language mod-	666
613	quences with sparse transformers. <i>arXiv preprint</i>	els. <i>arXiv preprint arXiv:2306.08543</i> .	667
614	<i>arXiv:1904.10509</i> .		
615	Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian,	Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng	668
616	Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias	Ji, and Sinong Wang. 2023. Lm-infinite: Simple	669
617	Plappert, Jerry Tworek, Jacob Hilton, Reiichiro	on-the-fly length generalization for large language	670
618	Nakano, et al. 2021. Training verifiers to solve math	models. <i>arXiv preprint arXiv:2308.16137</i> .	671
619	word problems. <i>arXiv preprint arXiv:2110.14168</i> .		
620	OpenCompass Contributors. 2023. Opencompass:	Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou,	672
621	A universal evaluation platform for foundation	Mantas Mazeika, Dawn Song, and Jacob Steinhardt.	673
622	models. <a href="https://github.com/open-compass/opencompass">https://github.com/open-compass/</a>	2020. Measuring massive multitask language under-	674
623	<a href="https://github.com/open-compass/opencompass">opencompass</a> .	standing. <i>arXiv preprint arXiv:2009.03300</i> .	675
624	Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Mas-	Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh,	676
625	sive language models can be accurately pruned in	Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner,	677
626	one-shot. In <i>International Conference on Machine</i>	Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister.	678
627	<i>Learning</i> , pages 10323–10337. PMLR.	2023. Distilling step-by-step! outperforming larger	679
		language models with less training data and smaller	680
		model sizes. <i>arXiv preprint arXiv:2305.02301</i> .	681
628	Elias Frantar, Saleh Ashkboos, Torsten Hoeftler, and	Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei	682
629	Dan Alistarh. 2022. Gptq: Accurate post-training	Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu,	683
630	quantization for generative pre-trained transformers.	Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu,	684
631	<i>arXiv preprint arXiv:2210.17323</i> .	Maosong Sun, and Junxian He. 2023. C-eval: A	685
		multi-level multi-discipline chinese evaluation suite	686
		for foundation models. In <i>Advances in Neural Infor-</i>	687
		<i>mation Processing Systems</i> .	688

689	Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke	Hongyu Ren, Hanjun Dai, Zihang Dai, Mengjiao Yang,	746
690	Zettlemoyer. 2017. Triviaqa: A large scale distantly	Jure Leskovec, Dale Schuurmans, and Bo Dai. 2021.	747
691	supervised challenge dataset for reading comprehen-	Combiner: Full attention transformer with sparse	748
692	sion. <i>arXiv preprint arXiv:1705.03551</i> .	computation cost. <i>Advances in Neural Information</i>	749
		<i>Processing Systems</i> , 34:22470–22482.	750
693	Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp,	Luka Ribar, Ivan Chelombiev, Luke Hudliss-Galley,	751
694	Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie,	Charlie Blake, Carlo Luschi, and Douglas Orr. 2023.	752
695	Yi Tay, Mostafa Dehghani, and Neil Houlsby.	Sparq attention: Bandwidth-efficient llm inference.	753
696	2022. Sparse upcycling: Training mixture-of-	<i>arXiv preprint arXiv:2312.04985</i> .	754
697	experts from dense checkpoints. <i>arXiv preprint</i>		
698	<i>arXiv:2212.05055</i> .	Noam Shazeer. 2019. Fast transformer decoding:	755
699	Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Fran-	One write-head is all you need. <i>arXiv preprint</i>	756
700	tar, Mark Kurtz, Benjamin Fineran, Michael Goin,	<i>arXiv:1911.02150</i> .	757
701	and Dan Alistarh. 2022. The optimal bert surgeon:		
702	Scalable and accurate second-order pruning for large	Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuo-	758
703	language models. <i>arXiv preprint arXiv:2203.07259</i> .	han Li, Max Ryabinin, Beidi Chen, Percy Liang,	759
704	Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying	Christopher Ré, Ion Stoica, and Ce Zhang. 2023.	760
705	Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gon-	Flexgen: High-throughput generative inference of	761
706	zalez, Hao Zhang, and Ion Stoica. 2023. Efficient	large language models with a single gpu. In <i>Inter-</i>	762
707	memory management for large language model serv-	<i>national Conference on Machine Learning</i> , pages	763
708	ing with pagedattention. In <i>Proceedings of the 29th</i>	31094–31116. PMLR.	764
709	<i>Symposium on Operating Systems Principles</i> , pages		
710	611–626.	Kumar Shridhar, Alessandro Stolfo, and Mrinmaya	765
711	Mo Li, Songyang Zhang, Yunxin Liu, and Kai Chen.	Sachan. 2022. Distilling reasoning capabilities	766
712	2024. <a href="#">Needlebench: Can llms do retrieval and rea-</a>	into smaller language models. <i>arXiv preprint</i>	767
713	<a href="#">soning in 1 million context window?</a> <i>Preprint</i> ,	<i>arXiv:2212.00193</i> .	768
714	<i>arXiv:2407.11963</i> .		
715	Nian Li, Chen Gao, Yong Li, and Qingmin Liao. 2023.	Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico	769
716	Large language model-empowered agents for simu-	Kolter. 2023. A simple and effective pruning ap-	770
717	lating macroeconomic activities. <i>arXiv preprint</i>	proach for large language models. <i>arXiv preprint</i>	771
718	<i>arXiv:2310.10436</i> .	<i>arXiv:2306.11695</i> .	772
719	Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-	Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao,	773
720	Ming Chen, Wei-Chen Wang, Guangxuan Xiao,	Baris Kasikci, and Song Han. 2024. Quest: Query-	774
721	Xingyu Dang, Chuang Gan, and Song Han. 2024.	aware sparsity for efficient long-context llm inference.	775
722	Awq: Activation-aware weight quantization for on-	<i>arXiv preprint arXiv:2406.10774</i> .	776
723	device llm compression and acceleration. <i>Proceed-</i>		
724	<i>ings of Machine Learning and Systems</i> , 6:87–100.	Zhen Wan, Fei Cheng, Zhuoyuan Mao, Qianying	777
725	Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang	Liu, Haiyue Song, Jiwei Li, and Sadao Kurohashi.	778
726	Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang,	2023. Gpt-re: In-context learning for relation ex-	779
727	Yuandong Tian, Christopher Re, et al. 2023. Deja	traction using large language models. <i>arXiv preprint</i>	780
728	vu: Contextual sparsity for efficient llms at infer-	<i>arXiv:2305.02105</i> .	781
729	ence time. In <i>International Conference on Machine</i>		
730	<i>Learning</i> , pages 22137–22176. PMLR.	Bingyang Wu, Yinmin Zhong, Zili Zhang, Gang Huang,	782
731	Xuezhe Ma, Xiang Kong, Sinong Wang, Chunting Zhou,	Xuanzhe Liu, and Xin Jin. 2023. Fast distributed	783
732	Jonathan May, Hao Ma, and Luke Zettlemoyer. 2021.	inference serving for large language models. <i>arXiv</i>	784
733	Luna: Linear unified nested attention. <i>Advances</i>	<i>preprint arXiv:2305.05920</i> .	785
734	<i>in Neural Information Processing Systems</i> , 34:2441–		
735	2453.	Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song	786
736	Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi,	Han, and Mike Lewis. 2023. Efficient streaming	787
737	and Roy Schwartz. 2024. Transformers are multi-	language models with attention sinks. <i>arXiv preprint</i>	788
738	state rnns. <i>arXiv preprint arXiv:2401.06104</i> .	<i>arXiv:2309.17453</i> .	789
739	F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel,	An Yang, Baosong Yang, Binyuan Hui, Bo Zheng,	790
740	B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer,	Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan	791
741	R. Weiss, V. Dubourg, J. Vanderplas, A. Passos,	Li, Dayiheng Liu, Fei Huang, Guanting Dong, Hao-	792
742	D. Cournapeau, M. Brucher, M. Perrot, and E. Duch-	ran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian	793
743	esnay. 2011. Scikit-learn: Machine learning in	Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin	794
744	Python. <i>Journal of Machine Learning Research</i> ,	Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang	795
745	12:2825–2830.	Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang,	796
		Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng	797
		Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin,	798
		Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu,	799
		Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng,	800
		Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin	801

- Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.
- Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538.
- Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297.
- Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2021. Moeification: Transformer feed-forward layers are mixtures of experts. *arXiv preprint arXiv:2110.01786*.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuan-dong Tian, Christopher Ré, Clark Barrett, et al. 2024. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36.
- Guangxiang Zhao, Junyang Lin, Zhiyuan Zhang, Xuancheng Ren, Qi Su, and Xu Sun. 2019. Explicit sparse transformer: Concentrated attention through explicit selection. *arXiv preprint arXiv:1912.11637*.
- Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.
- Li Zhong and Zilong Wang. 2024. Can llm replace stack overflow? a study on robustness and reliability of large language model code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 21841–21849.

## A Appendix

In this appendix, we provide comprehensive details on the parameter  $p$ , various hardware platforms, and the specific hyperparameter settings used in our experiments.

### A.1 Top- $p$ Ablation

This section conducts an ablation study on parameter  $p$  to evaluate its impact on model performance across different sequence lengths. The test set consists of newly constructed datasets created by concatenating SQuAD contexts with varying sequence lengths. The performance of the Top  $p$  (or Top  $k$ ) method was tested under different  $p$  (or  $k$ ) values, where  $p = 1$  corresponds to the Dense model.

As shown in Figure 7, in experiments with different sequence lengths, the proportionally adaptive Top  $p$  method maintains performance close to that of the Dense model across varying  $p$  values. In contrast, using Top  $k$  with fixed values (e.g.,  $k = 64$  or  $32$ ) significantly degrades model performance, and this trend becomes more pronounced as  $k$  decreases from 64 to 32. Therefore, the Top  $p$  approach provides more stable performance across different context lengths.

Figure 8 illustrates the changes in time and memory costs under different  $p$  (or  $k$ ) values. It can be observed that at  $p = 0.1$ , compared to the Dense model, there is a significant improvement in both time and memory efficiency. Moreover, when compared to Top  $k$  ( $k = 64$ ), the Top  $p$  method achieves a clear performance gain without introducing substantial additional time or memory overhead. Thus, the Top  $p$  method outperforms the Top  $k$  approach, and  $p = 0.1$  is a relatively optimal choice.

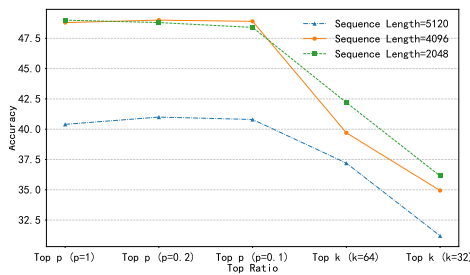


Figure 7: Top- $p$  Ablation

### A.2 CSA on PC-side Hardware

To verify the CSA method’s applicability on various platforms, we expanded our experiments to include other computing environments, based on

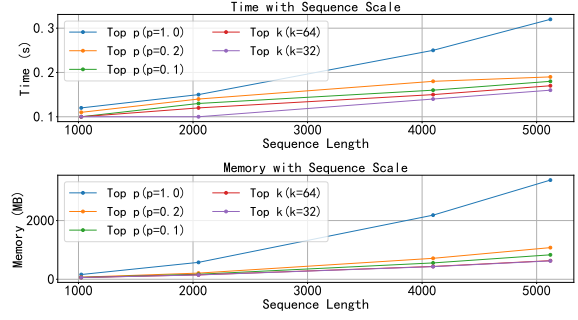


Figure 8: Top- $p$  Time/Memory Ablation

our previous analysis on the A100 accelerator card (see §5.2.5). We tested on the Intel(R) Xeon(R) Platinum 8163 CPU and the NVIDIA RTX4090 GPU to evaluate CSA’s performance across different architectures. The experiments demonstrate that CSA performs exceptionally well on both CPU and consumer-grade GPU platforms. On the CPU, it achieves a 2.79× speedup in inference efficiency (Figure 9a) along with a 44% reduction in memory usage (Figure 9c). On the RTX 4090, it delivers a 1.86× speedup in inference efficiency (Figure 9b) and a 52% reduction in memory consumption (Figure 9d). These results indicate that CSA effectively improves efficiency on both CPU and consumer-grade GPU platforms.

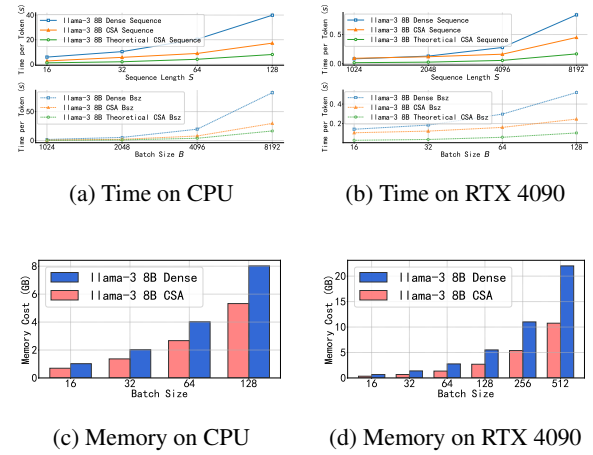


Figure 9: The CSA running results on different architecture devices

### A.3 Derivation of the QHR Formula

The Query Hit Rate (QHR) quantifies the predictive ability of attention head  $h_n$  for the *salient tokens* of another head  $h_m$ . The calculation involves two key steps:

1. Extracting salient tokens from  $h_n$ : For each to-



ken  $i \in I$ , select the Top- $p$  tokens with highest attention scores in  $h_n$  (index set  $M(h_n, p, i)$ ).

2. Validating token importance in  $h_m$ : Evaluate salience through token rankings in  $h_m$ , where lower rank values indicate greater importance.

For a specific token  $i \in I$ , the predictive score(Hit Rate) is:

$$\text{HR}(i, h_m, h_n) = \sum_{s \in M(h_n, p, i)} \left( \frac{1}{\text{GetRank}(s, h_m, i)} \right)^t \quad (2)$$

Where  $\text{GetRank}(s, h_m, i)$  returns the rank (1-based) of token  $s$  in  $h_m$ 's attention scores for token  $i$ . The inverse weighting  $\left(\frac{1}{\text{Rank}}\right)^t$  ensures:(1) Higher-ranked tokens contribute more ( $\frac{1}{1^t} = 1$  for rank 1. (2) Temperature  $t$  controls ranking sensitivity: $t > 1$ : Emphasizes top rankings; $t < 1$ : Smoothes ranking differences.

The final QHR averages HR scores across all tokens:

$$\text{QHR}_I(h_m, h_n) = \frac{1}{|I|} \sum_{i \in I} \text{HR}(i, h_m, h_n) \quad (3)$$

The normalization factor  $\frac{1}{|I|}$  ensures comparability across different query lengths.

#### A.4 Experiment Hyperparameter Setting

This section elaborates on the experimental configurations with details.

In §5.2.1, we adopted a 5-shot approach for evaluating performance on the MMLU, Ceval, and GSM8K datasets. Specifically, we constructed a multi-round dialogue prompt, consistent with the methodology used in opencompass. For evaluation metrics, we used exact match for MMLU and Ceval, while for other evaluations, we employed the same evaluator as used in opencompass (Contributors, 2023). To ensure a fair comparison with the baseline, we set the  $r$  value of SparQ to 32, maintaining the same compression ratio as CSA. The hyperparameter settings for each model were as follows: for LLaMA-3 8B,  $g = 8$  and  $c = 1$ ; for Qwen2 14B,  $g = 16$  and  $c = 2$ ; and for LLaMA-3 70B,  $g = 16$  and  $c = 2$ .

In § 5.2.2, the random selection method does not support further community division; the number of randomly selected communities was set to 1.

In § 5.2.3, we performed comparative experiments with configurations:  $g = 8, c = 1$  for LLaMA-3 8B,  $g = 16, c = 2$  for LLaMA-3 70B, and  $g = 16, c = 2$  for Qwen2 14B. These settings were chosen to achieve the best balance between accuracy and computational efficiency.

In § 5.2.4, we selected the "Needle in a Haystack" task as the evaluation benchmark. Following the methodology used in opencompass, we first concatenated a context of a specified length from the SQuAD dataset and then inserted a "needle" (a specific text segment) into the context for evaluation. To ensure consistency, we utilized the opencompass Evaluator for performance measurement.

In § 5.2.5, we focus on evaluating the computational and memory efficiency of CSA, specifically during the prefilling stage, as this phase represents the primary performance bottleneck for large language models. As CSA primarily improves the efficiency of attention score calculation, we specifically measured the computational gains within  $QKV$ . To isolate the impact of CSA, we subtracted the effect of the KV Cache, which is identical in both Dense and CSA settings. Our experiments were conducted on an A100 80GB GPU, testing the LLaMA-3 8B model. In the sequence length scaling experiments, we kept the batch size fixed at 1, while in the batch size scaling experiments, we maintained a sequence length of 512. For the configurations of CSA and SparQ, we kept the settings consistent with those in §5.2.1.

In § 4.2. During the phases for community clustering and representative heads selection, we selected queries from multiple evaluation sets when calculating QHR to enhance statistical reliability. For parameter settings, we aligned the configuration with formal experimental conditions by setting  $p = 0.1$ . In the calculation of HR and QHR, we used a temperature coefficient  $t = 1$  to maintain consistency. When constructing the graph of attention head nodes, we set  $j = 2$  to determine the number of representative nodes. For community detection, we utilized the Spectral Clustering implementation from scikit-learn.

## A.5 Case Study

### A.5.1 LLaMA-3 8B (MMLU)

```
###Question
HUMAN:
"""There is a single choice question
about global facts. Answer the
question by replying A, B, C or D.
Question: As of 2019, about what
percentage of people from the United
States says homosexuality should be
accepted by society?
A. 52%
B. 62%
C. 72%
D. 82%"""
### Answer:
BOT:
B
...few shots
HUMAN:
"""There is a single choice question
about global facts. Answer the
question by replying A, B, C or D.
Question:
What was GDP per capita in the United
States in 1850 when adjusting for
inflation and PPP in 2011 prices?
A. About $300
B. About $3k
C. About $8k
D. About $15k"""
### Answer:
DENSE: C
CSA: C
```

### A.5.2 LLaMA-3 8B (HumanEval)

```
###Question
HUMAN:
"""from typing import List\n\ndef
separate_paren_groups(paren_string:
str) -> List[str]:\n    Input to
this function is a string containing
multiple groups of nested
parentheses. Your goal is to\n
separate those group into separate
strings and return the list of those
.\n    Separate groups are balanced
(each open brace is properly closed)
and not nested within each other\n
Ignore any spaces in the input
string.\n    >>>
separate_paren_groups('\( ) (( )) ((
))\n [\'(\')\', \'(())\',
\'(()())\']\n """
###Answer:
DENSE:
"""from typing import List\n\ndef
separate_paren_groups(paren_string:
str) -> List[str]:\n    This
function takes a string containing
multiple groups of nested
parentheses as input.\n    It
separates those groups into separate
strings and returns the list of
those.\n    Separate groups are
balanced (each open brace is
```

```
properly closed) and not nested
within each other.\n    It ignores
any spaces in the input string.\n\n
Parameters:\n    paren_string (
str): A string containing multiple
groups of nested parentheses.\n\n
Returns:\n    List[str]: A list of
strings, each containing a group of
balanced parentheses.\n    \n    #
Remove spaces from the input string\n
    paren_string = paren_string.
replace(" ", "")\n\n    # Initialize
an empty list to store the
separated groups\n    groups = []\n\n
    # Initialize an empty string to
build the current group\n
    current_group = ""\n\n    #
Initialize a counter to keep track
of the balance of parentheses\n
    balance = 0\n\n    # Iterate over
each character in the input string\n
    for char in paren_string:\n
        # If the character is an open
parenthesis, increment the balance\n
        if char == "(\n
            balance += 1\n
            current_group += char\n
            # If
the character is a close parenthesis
, decrement the balance\n
        elif char == ")\n
            balance -= 1\n
            current_group += char\n
            #
            If the balance becomes zero, it
means we have a complete group, so
add it to the list and reset the
current group\n
            if
            balance == 0:\n
                groups.append(current_group)\n
                current_group = ""\n\n
    # Return the list of separated
groups\n    return groups\n """
```

```
CSA:
"""from typing import List\n\ndef
separate_paren_groups(paren_string:
str) -> List[str]:\n    This
function separates multiple groups
of nested parentheses into separate
strings.\n\n    Args:\n
    paren_string (str): A string
containing multiple groups of nested
parentheses.\n\n    Returns:\n
    List[str]: A list of separate
strings, each representing a group
of nested parentheses.\n    \n    \n
    # Remove spaces from the input
string\n    paren_string =
paren_string.replace(" ", "")\n\n
    # Initialize an empty list to store
the separated groups\n
    separated_groups = []\n\n    #
Initialize an empty string to build
the current group\n    current_group
= ""\n\n    # Initialize a counter
for open and close parentheses\n
    open_count = 0\n    close_count = 0\n\n
    # Iterate over each character
in the input string\n    for char
in paren_string:\n        # If the
character is an open parenthesis,
```

1123	increment the open count\n	if	and bakes muffins with 4 eggs,	1191
1124	char == "(": \n		leaving: \n\n16 - 3 - 4 = 9 eggs\n	1192
1125	open_count += 1\n		nShe sells these 9 eggs at the	1193
1126	current_group += char\n	# If	farmers' market for \$2 per egg, so	1194
1127	the character is a close parenthesis		she makes: \n\n9 eggs x \$2 per egg =	1195
1128	\n elif char == ")": \n		\$18\n\n#### \$18""	1196
1129	close_count += 1\n		CSA: ""Janet's ducks lay 16 eggs per	1197
1130	current_group += char\n	#	day. She eats 3 for breakfast, and	1198
1131	If the current group is balanced,		bakes muffins with 4, leaving her	1199
1132	add it to the list and reset the		with: \n\n16 - 3 - 4 = 9 eggs\n\nShe	1200
1133	current group\n	if	sells these 9 eggs at the farmers'	1201
1134	open_count == close_count: \n		market for \$2 per egg, making: \n\n9	1202
1135	separated_groups.append(		eggs x \$2 per egg = \$18\n\n#### 18""	1203
1136	current_group)\n		""	1204
1137	current_group = ""\n			
1138	open_count = 0\n			
1139	close_count = 0\n	# If the		
1140	current group is not balanced, reset			
1141	the current group\n	if		
1142	open_count < close_count: \n			
1143	current_group = ""\n			
1144	open_count = 0\n			
1145	close_count = 0\n\n	# Add the		
1146	last group to the list if it's not			
1147	empty\n	if current_group: \n		
1148	separated_groups.append(			
1149	current_group)\n\n	return		
1150	separated_groups\n""			
1151				
	<b>A.5.3 LLaMA-3 8b(GSM8k)</b>		<b>A.5.4 LLaMA-3 8B (TriviaQA)</b>	1205
1152	###Question		###Question	1206
1153	HUMAN:		HUMAN:	1207
1154	""		""	1208
1155	Question: \nJosh decides to try flipping		Answer these questions, your answer	1209
1156	a house. He buys a house for \$80		should be as simple as possible,	1210
1157	,000 and then puts in \$50,000 in		start your answer with the prompt '	1211
1158	repairs. This increased the value		The answer is '.	1212
1159	of the house by 150%. How much		Q: Which Lloyd Webber musical premiered	1213
1160	profit did he make?\nLet's think		in the US on 10th December 1993?	1214
1161	step by step\n		""	1215
1162	Answer:		###Answer:	1216
1163	The cost of the house and repairs came		DENSE: The answer is Sunset Boulevard.	1217
1164	out to 80,000+50,000=\$		CSA: The answer is Sunset Boulevard.	1218
1165	<<80000+50000=130000>>130,000\nHe			
1166	increased the value of the house by			
1167	80,000*1.5=<<80000*1.5=120000>>120,000\n			
1168	nSo the new value of the house is			
1169	120,000+80,000=\$			
1170	<<120000+80000=200000>>200,000\nSo			
1171	he made a profit of 200,000-130,000=			
1172	\$<<200000-130000=70000>>70,000\n###			
1173	70000\n""			
1174	...few shots			
1175	HUMAN:			
1176	""			
1177	Question:			
1178	Janet's ducks lay 16 eggs per day. She			
1179	eats three for breakfast every			
1180	morning and bakes muffins for her			
1181	friends every day with four. She			
1182	sells the remainder at the farmers'			
1183	market daily for \$2 per fresh duck			
1184	egg. How much in dollars does she			
1185	make every day at the farmers'			
1186	market?			
1187	Let's think step by step""			
1188	###Answer:			
1189	DENSE: ""Janet's ducks lay 16 eggs per			
1190	day. She eats 3 eggs for breakfast			