

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

Anonymous ACL submission

Abstract

The inference efficiency of large language models (LLMs) is limited by the computational complexity and memory usage of attention layers. To address these challenges, we introduce Consensus Sparse Attention (CSA), a technique that leverages the consensus of a few representative attention heads to select the *Key* tokens for the remaining heads, thereby limiting the attention computation space from all tokens to a small number of potential candidate tokens, effectively reducing computational and peak memory consumption without additional training. Experiments conducted on diverse scale models and varied downstream tasks demonstrate that CSA can offer a significant improvement in computational efficiency with a negligible accuracy decrease. In particular, CSA was able to achieve a two-fold speed increase, along with a half reduction of peak memory usage in the attention layer computation during the prefilling stage on LLaMA-3.

1 Introduction

Due to the efficient ability in modeling complex relationships between different tokens, Large Language Models (LLMs) have recently achieved excellent performance in various fields (Zhao et al., 2023; Chang et al., 2024), including multi-model fusion (Cha et al., 2024), relation extraction (Wan et al., 2023), code generation (Zhong and Wang, 2024) and even agent-based decision-making (Li et al., 2023).

However, the quadratic time and space complexity of dense attention in LLMs create a critical bottleneck for widespread application and efficient deployment as model size and input sequence length increase, significantly raising computational resource and memory demands.

Many previous works have achieved efficiency gains through sparse attention mechanisms, such as 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 transformer (Zhao et al., 2019). The above method approximates dense attention based on the fact that most attention scores are concentrated on a few important tokens. However, they either require retraining or need to compute the complete attention scores before selecting the important tokens, which does not sufficiently reduce the computational load. SparQ (Ribar et al., 2023) finds the r largest components of the query vector and gathers the corresponding components along the hidden dimension of the *Key* tokens to approximate the attention scores. The compression along the hidden dimension reduces the computational load, but the resulting attention scores $[b, h, s, s]$ are of the same size as in the dense condition, which does not yield significant peak memory consumption benefits.

Our work began with the observation that attention heads tend to consistently select some important *Key* tokens. Specifically, certain *Key* tokens receive high attention scores across different attention heads. As shown in Figure 1b, different colors represent different attention heads. It can be noted that tokens at certain index positions receive high attention scores across various attention heads, indicating that these tokens are valued by the majority of the attention heads. Building on this insight, we propose the **Consensus Sparse Attention (CSA)** method for multi-head attention. By computing a small number of g representative heads, we use their consensus to identify important *Key* tokens for the others, thus avoiding the computation of all *Key* tokens. Since only the maximum $[b, g, s, s]$ ($g < h$) attention scores need to be calculated, CSA effectively reduces both the computational load and peak memory usage of the attention layer.

In mainstream benchmark tests presented in Table 1, CSA maintained comparable performance to Dense models. In high-batch and long-sequence scenarios, CSA achieved a true two-fold speedup

083 and reduced memory peak by 50% when comput- 131
084 ing the attention layer. These results highlight 132
085 CSA’s advantages in both efficiency and perfor- 133
086 mance. 134

087 In summary, we make the following contribu- 135
088 tion: 136

- 089 • We reveal the consensus exhibited by multiple 137
090 attention heads when selecting important *Key* 138
091 tokens and validate this phenomenon through 139
092 experiments. 140
- 093 • We propose **Consensus Sparse Attention** 141
094 (**CSA**), leveraging the consensus among a few 142
095 representative attention heads to select *Key* 143
096 tokens for the others, thus reducing computa- 144
097 tional load and memory usage in the attention 145
098 layer without extra training. 146
- 099 • We evaluated CSA through experiments, and 147
100 the results show that it outperforms existing 148
101 benchmarks across multiple key performance 149
102 indicators, fully validating its effectiveness. 150

103 2 Related Work 152

104 2.1 Efficient LLMs Inference 153

105 LLMs usually require a higher inference cost when 154
106 processing large amounts of queries, which poses a 155
107 huge challenge for its deployment. To improve the 156
108 inference efficiency of LLMs, some current works 157
109 optimize two important parts in the model, Feed 158
110 Forward Network (FFN)(Zhang et al., 2021; Gao 159
111 et al., 2022; Komatsuzaki et al., 2022) and Atten- 160
112 tion Operation(Shazeer, 2019; Ainslie et al., 2023; 161
113 Ma et al., 2021), by designing efficient structure 162
114 or strategies. Some other works consider apply- 163
115 ing classical scheduling strategies in query batch- 164
116 ing process to handle asynchronous queries more 165
117 quickly, such as FCFS(Yu et al., 2022), Multi-Level 166
118 Feedback Queue(Wu et al., 2023) and Continuous 167
119 Batching(Kwon et al., 2023). Besides, in model 168
120 compression, quantization is a commonly used 169
121 method. It reduces the computational and mem- 170
122 ory costs of LLMs by converting model weights 171
123 and activations from high to low bit-widths, such 172
124 as GPTQ(Frantar et al., 2022) minimizes the differ- 173
125 ence in model output before and after quantization 174
126 by using a small portion of calibration data for 175
127 the weight matrix of each layer, AWQ(Lin et al., 176
128 2024)selects salient weights based on the activa- 177
129 tion distribution. Also, some methods (Frantar and 178
130 Alistarh, 2023; Sun et al., 2023; Kurtic et al., 2022)

131 prune the model parameters, or (Gu et al., 2023; 132
133 Hsieh et al., 2023; Shridhar et al., 2022) compress 134
135 the model volume by distilling knowledge into a 136
137 smaller one. The method studied in this paper 138
139 is closely related to sparsity in model compression 140
141 and focuses on the bottleneck of dense self- 142
143 attention in inference process. 144

145 2.2 Sparse Attention Compression 150

146 Due to the sparsity of self-attention matrix, ex- 151
147 tracting the important parts from it has always 152
148 been an active research field. For example, meth- 153
149 ods like Local(Child et al., 2019; Ren et al., 154
150 2021), Global(Beltagy et al., 2020), Hybrid(Zaheer 155
151 et al., 2020) improve the computational efficiency 156
152 of attention scores by choosing random, adja- 157
153 cent or specially marked tokens during long con- 158
154 text process. LM-Infinite(Han et al., 2023) and 159
155 StreamingLLM(Xiao et al., 2023) adopt some fixed 160
156 sparse patterns to select the latest and important 161
157 tokens. Top-k(Zhao et al., 2019) and FlexGen(Sheng 162
158 et al., 2023) identify important tokens through the 163
159 attention scores and Tang et al.(Tang et al., 2024) 164
160 link the selection of important attention scores and 165
161 currently predicted token together. Meanwhile, the 166
162 eviction strategy maintains a certain size by con- 167
163 tinuously deleting irrelevant tokens. H₂O(Zhang 168
164 et al., 2024) maintains a budget space of size k 169
165 by accumulating historical attention weight scores. 170
166 TOVA(Oren et al., 2024) discards tokens with 171
167 lower attention scores based on the current query. 172
168 SparQ(Ribar et al., 2023) reduces the memory 173
169 bandwidth during the computation process by se- 174
170 lecting key query tokens before computing the at- 175
171 tention weights. FastGen(Ge et al., 2023) formu- 176
172 lates separate compression strategies for them re- 177
173 spectively based on the observation of different 178
174 heads. Unlike the aforementioned methods, our ap- 179
175 proach achieves sparse attention by leveraging the 180
176 consensus among attention heads on key tokens. 181

182 3 Background and Motivation 187

183 In well-trained Transformer models, it is often 188
184 observed that most tokens receive low attention 189
185 scores, while the primary attention is concentrated 190
186 on a small subset of tokens, as depicted in Figure 191
187 1b. This suggests that only a minority of tokens 192
188 significantly impact model performance. Statistical 193
189 results in Figure 1c show that attention scores in 194
190 Dense models are largely concentrated on a few 195
191 important tokens, indicating that focusing on these 196
192

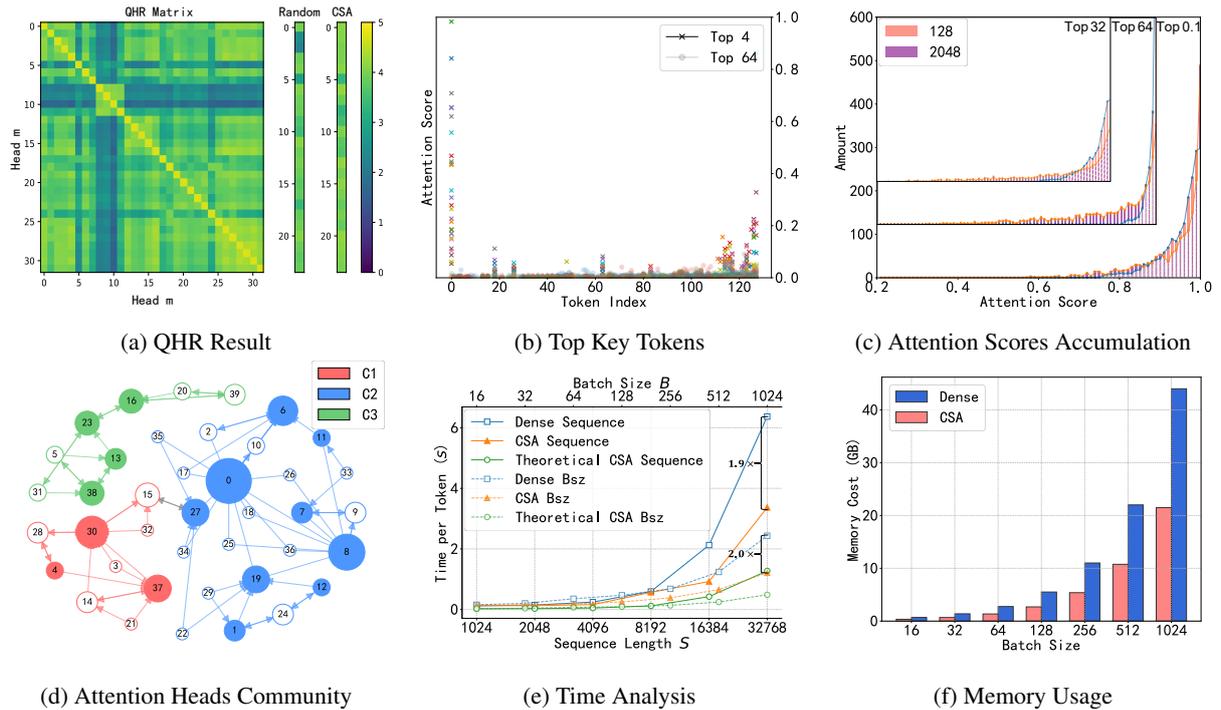


Figure 1: Subfigure (a) QHR Matrix illustrates the QHR scores between any two attention heads. Both Random and CSA show the QHR scores of the remaining heads as voted by the representative attention heads. Subfigure (b) displays the distribution of Attention Scores for the Top 64 *Key* tokens and Top 4 *Key* tokens across different heads, with each color representing a distinct head. Subfigure (c) presents the distribution of cumulative attention scores for the top *Key* tokens at sequence lengths of 128 and 2048. Subfigure (d) demonstrates the results of community division and the selection of representative attention heads; the size of each node indicates its in-degree, with C1, C2, and C3 representing the divided communities, and solid nodes indicating representative nodes. Subfigure (e) illustrates the token generation time across different batch sizes (top x-axis, dotted line) and sequence lengths (bottom x-axis, solid line). In particular, for the experiments conducted on the LLaMA3 8B model, the sequence length experiment was performed with a batch size (bsz) of 1, while the batch size experiment used a sequence length of 512. Subfigure (f) illustrates the peak memory usage of the attention layer when computing the first token on LLaMA3 8B with a sequence length of 512.

tokens can effectively capture sufficient attention information.

The Current Top-k Tokens Mechanism Faces Several Challenges. Existing studies like Explicit sparse transformer (Zhao et al., 2019) and SparQ (Ribar et al., 2023) have implemented Sparse Attention by selecting a fixed number (k) of *Key* tokens with the highest attention scores to replace Dense Attention. This approach accelerates inference while preserving model capabilities. However, using a fixed number of Top k *Key* tokens results in a sparser distribution of attention scores as the input sequence length increases. As shown in Figure 1c, when the input sequence length grows from 128 to 2048, the attention score distribution for the top 32 *Key* tokens becomes more dispersed. Although the top 64 *Key* exhibit a higher concentration, the overall trend of declining concentration persists. This indicates that regardless of adjust-

ments to the value of k , the top k method struggles with declining attention score concentration, posing challenges for handling long-sequence tasks.

Percentage-based Top p Tokens Mechanism demonstrates superior performance. To address the decline in attention concentration caused by selecting a fixed number of important *Key* tokens (Top k), we did experiments to find a simple effective percentage-based method. This approach involves selecting a fixed proportion of tokens as *Key* tokens to maintain stable accumulated attention scores. To differentiate this method from the existing Top k approach, which uses a fixed number of tokens, we refer to it as the Top p method. As illustrated in Figure 1c, as the input sequence length increases from 128 to 2048, the Top p method consistently maintains a high concentration of attention scores. This demonstrates that the Top p method can sustain stable model performance as sequence

length scales, as verified in Section 5.3.3.

4 Methodology

Unlike the traditional top- p method, which requires first calculating the complete attention scores and then selecting the top p *Key* tokens, we aim to propose a method that does not require calculating the full attention scores. This method utilizes the consensus of attention heads on important tokens to predict the indices of the top p *Key* tokens.

4.1 Consensus Sparse Attention

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

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:   Identify representative heads  $g_i$  and remain
   heads  $h_{c-g_i}$  in community  $c$ 
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:    $\mathbf{S}_{g_i} \leftarrow \mathbf{S}_{g_i} \odot \text{Mask}_p$ 
8:    $\hat{\mathbf{S}}_c \leftarrow \sum_{g_i} \mathbf{S}_{g_i}$ . // consensus scores voting
9:    $\mathcal{P}_{selected} \leftarrow \text{top}_p(\hat{\mathbf{S}}_c)$ 
10:   $\mathbf{K}_{c-g_i}, \mathbf{V}_{c-g_i} \leftarrow \text{gather}(\mathcal{P}_{selected})$ 
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:   $\mathbf{O} \leftarrow \mathbf{O} + \mathbf{S}_{c-g_i} \mathbf{V}_{c-g_i} + \mathbf{O}_{g_i}$ 
13: end for
14: return  $\mathbf{O}$ 

```

In Figure 1b, we observe a clear consensus among different attention heads in selecting key tokens: Important key tokens are typically the focus of most attention heads, with stronger consensus for more significant tokens. This observation is further supported by the QHR matrix analysis in Figure 1a. Experimental results indicate that most QHR scores (defined in Equation 2) exceed 2, compared to a score of only 0.41 for random selection, demonstrating the effectiveness of consensus-based inference. However, consensus levels vary among different attention heads, and some heads show a clear advantage in guiding others.

Based on these insights, we propose an efficient important *Key* token prediction method called Consensus Sparse Attention (CSA). Specifically,

instead of computing the important *Key* tokens for each attention head individually, we first select representative attention heads g and then use their consensus on important *Key* tokens to predict the important *Key* tokens for the remaining heads. This strategy avoids the need to compute the complete attention scores for each head, thereby reducing computational overhead and enhancing the efficiency of inference.

The consensus sparse attention mechanism operates through the following key phases, as shown in Figure 2

1. **Community Initialization:** For each attention head community (introduced in Section 4.3), identify representative heads (introduced in Section 4.2) that capture dense attention patterns, as well as the remaining heads that use sparse attention patterns. See details in Algorithm 2.
2. **Representative Head Processing:** Representative heads use standard-scaled dot-product attention to compute full attention scores. These scores preserve complete sequence information within the representative heads.
3. **Consensus Scores Voting:** Enforce sparsity by masking nontop- p elements in the attention matrices of the representative heads to form top- p votes for each attention head. Sum the masked scores across the representative heads to generate a consensus score matrix through voting, highlighting strongly attended tokens.
4. **Sparse Pattern Propagation:** Use the consensus scores to select top- p indices with the highest aggregate attention. These indices define sparse attention patterns propagated to remain heads through *Key* selection.
5. **Attention Computation:** remaining heads compute attention using only the selected tokens. The final output combines results from both representative and remain heads.

The above process demonstrates how CSA achieves sparse attention through consensus among representative attention heads, thereby simultaneously reducing computational and peak memory consumption.

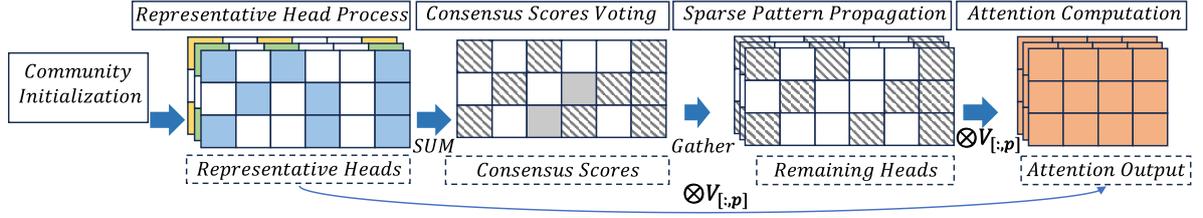


Figure 2: Overview of CSA framework

Algorithm 2 Community and Heads Selection

Input: Query I , h attention heads, representative heads number g , temperature t

Output: A set of representative attention heads

- 1: Compute $HR_i(h_m, h_n); \forall i \in I, m \neq n$.
// (h_m, h_n) is a pair of $head_m$ and $head_n$
- 2: Compute $QHR(h_m, h_n); m \neq n$
- 3: Form a directed graph G with h nodes using $QHR(h_m, *)$ as weight to select edges
- 4: $G' \leftarrow f(G)$. // f is Spectral Clustering
- 5: **for** each community C_i **do**
- 6: $g_i \leftarrow size(C_i) / h * g$. // g_i refers to representative heads for community C_i
- 7: **while** not all g_i heads are selected **do**
- 8: Select the head (node) with the highest in-degree
- 9: Remove the selected node along with its associated edges
- 10: **end while**
- 11: **end for**
- 12: **return** A set of attention heads $\{g_i\}$

4.2 Representative Heads Selection

As mentioned, the selection of representative attention heads significantly influences the prediction of top p tokens. To address this, we propose an adaptive method for selecting representative heads based on prior knowledge of different input queries. Specifically, for a token i in the input query I , we define the *Hit Rate*, denoted as $HR_i(h_m, h_n)$, to represent the weighted accuracy of using the top p *Key* tokens of attention head h_n to predict the important *Key* tokens of attention head h_m on token i , which indicates the effectiveness of h_n in predicting h_m . The calculation is as follows:

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

Where function M returns the indices of the top p *key* tokens that have the highest attention scores

for a *Query* token i in attention head h_n . The function $GetRank$ returns the rank of the attention scores for the *key* tokens at index s among all *key* tokens in attention head h_m . The t is a temperature parameter that is used to smooth the influence of the ranking order.

For an input query sequence I of length L , we use the QHR to quantify the advantage of an attention head h_n over other heads in predicting the important *Key* tokens of attention head h_m . The calculation formula is as follows:

$$QHR(h_m, h_n, I) = \frac{1}{L} \sum_{i \in I}^i (HR_i(h_m, h_n)) \quad (2)$$

QHR is the average of HR scores on the Query I . Obviously, the higher the $QHR(h_m, h_n)$, the more advantageous h_n is in inferring the important *Key* tokens of h_m compared to other attention heads, indicating that h_n is regarded as a representative attention head for h_m .

Afterward, we calculate the QHR between any pair of heads through Equation 2. For each head h_m , we select the top j attention heads with the highest QHR , identifying those that statistically best represent h_m . We then construct j directed edges from h_m to these heads, forming a directed graph, as illustrated in Figure 1d. In this graph, some nodes have a higher in-degree than others, indicating they are recognized by more nodes as representative attention heads. We then select g attention heads in descending order of in-degree. These g attention heads are the representative attention heads we seek.

4.3 Heads Community Clustering

However, treating all attention heads as a whole may introduce representational biases. As shown in Figure 1d, some sections (green parts) of the directed graph are independent. Thus, when selecting representative nodes for the green parts, heads from the blue and red sections should not be considered. To enhance selection accuracy, we use a spectral

clustering algorithm for community detection, dividing all heads into c communities and assigning g_i representative heads to each community based on its size, with the total number being g .

Specifically, after partitioning the communities, within each community, we first select the attention head node with the highest in-degree and remove it from the community, along with its corresponding edges. We repeat this process until a sufficient number of representative attention heads are identified within each community. The detailed procedure is provided in Algorithm 2.

4.4 Complexity and Memory Analysis

4.4.1 Computational Complexity

The baseline dense attention mechanism involves:

- *QK^T Multiplication*: $2bhs^2d$ operations for $[b, h, s, d] \times [b, h, d, s]$ tensor contraction
- *Value Projection*: $2bhs^2d$ operations for $[b, h, s, s] \times [b, h, s, d]$ tensor product

The CSA mechanism includes:

- *QK^T Multiplication*: $2bgs^2d$ operations for g representative heads, $2b(h - g)ps^2d$ operations for remaining heads with p
- *Value Projection*: $2bhps^2d$ operations for compressed value projection

The total complexity for the dense model is $4bhs^2d$, while for CSA it is $2bgs^2d + 2b(h - g)ps^2d + 2bhps^2d$. Thus, CSA reduces computational cost by a factor of $\frac{g+(h-g)p+hp}{2h}$.

4.4.2 Memory Utilization

The CSA mechanism can effectively reduce the peak memory usage during the attention computation process.

Baseline: Peak memory $\mathcal{O}(bhs^2)$ for storing the full attention matrix $[b, h, s, s]$

CSA: $\mathcal{O}(bgs^2)$ memory for $[b, g, s, s]$ matrices in representative heads, $\mathcal{O}(b(h - g)ps^2)$ memory for $[b, h - g, s, ps]$ matrices in remaining heads

The peak memory is $\max(bgs^2, b(h - g)ps^2)$. 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 Experiment

We aim to answer the following research questions in our experiments:

- RQ1: Compared to baseline methods, how does the CSA method perform across different NLP tasks and model scales?
- RQ2: Do Representative Heads Selection and Heads Community Clustering contribute to the effectiveness of CSA?
- RQ3: Can CSA maintain consistent performance as the sequence length scales?
- RQ4: Does CSA provide significant improvements in inference speed and memory consumption?

5.1 Experiment Setting

To validate the broad effectiveness of our method, we tested mainstream models of various sizes and architectures. Since Instruct models are currently the most widely used, we selected the Instruct version of the corresponding models for our experiments. We tested our method on multiple mainstream datasets and tasks.

5.1.1 Task

To validate the performance of CSA in practical NLP tasks, we refer to the OpenCompass(Contributors, 2023) framework to construct a multi-dimensional evaluation task set: we use MMLU(Hendrycks et al., 2020) and Ceval(Huang et al., 2023) to evaluate the model’s comprehensive capabilities in both languages. For specialized capabilities, we employ HumanEval(Chen et al., 2021) to assess coding, GSM8K(Cobbe et al., 2021) for mathematical reasoning, TriviaQA(Joshi et al., 2017) for knowledge understanding, SQuAD2.0 for reading comprehension, and "needle-in-a-haystack"(Li et al., 2024) for long-text processing. In terms of model efficiency, to control for implementation differences, we measure memory and time consumption for different batch sizes and sequence lengths on the core matrix computation components under aligned PyTorch implementations. Further details are provided in Section B.

5.1.2 Models

We evaluated the CSA on mainstream open-source models, including Qwen14B-chat, LLaMA-3-8B-Chat, and LLaMA-3-70B-Chat. In the long-text experiments, we utilized the chatglm-6b-32k model, which was trained specifically for long-text experiment.

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

Table 1: The experimental results of CSA on different models are shown above. We used Pass@5 as the metric on HumanEval and accuracy on the other datasets. Dense represents the Dense model, Top represents the original top p method, SparQ is the implementation of the SparQ method, and CSA represents our proposed method. In all the experiments, p=0.1. For LLaMA-3 8b, g=8 and c=1; for Qwen 14B, g=16 and c=2; for LLaMA-3 70B, g=16 and c=2.

5.1.3 Baseline

For the baseline, we considered the original top p and SparQ (Ribar et al., 2023). SparQ computes approximate attention scores through vector components selection and selects *Key* tokens based on these scores. 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. Considering that SparQ select a fixed number of *Key* tokens, whereas CSA employs a top p mechanism based on percentages, to ensure a fair comparison, we also adopted the top p method for testing in SparQ. Since CSA supports prefilling but the code provided for SparQ only supports the decoding stage, we modified SparQ to enable it to function during the prefilling stage for comparative purposes.

5.2 Main Result(RQ1)

Table 1 shows the results of our method running under the condition of p=0.1 on different scales of Instruct models. Here, Dense represents the test results of the Dense model, serving as the actual benchmark; Top denotes the results achieved using the original top p method, representing the theoretical upper limit; SparQ is the baseline method at the same compression ratio. Experimental results indicate that, our method outperforms the baseline method on almost all datasets, approaches the theoretical upper limit of the top p method, and is nearly equivalent to the Dense model. Notably,

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

Table 2: Random and CSA selection mechanism.

on some datasets, our method even surpasses the Dense model. We believe this is because the top p method eliminates interference from tokens with lower attention scores, thereby concentrating attention and enhancing model performance.

5.3 Ablations

5.3.1 Heads Selection Ablation(RQ2)

To validate the effectiveness of the representative heads selection in CSA, we conducted comparative experiments with random selection. As shown in Table 2, CSA outperforms random selection in all scenarios. Figure 1a, through *QHR* scores (Random and CSA), further reveals that compared to single-head inference, random selection significantly reduces low-hit regions (cool tones) through consensus scores voting, while CSA further compresses low-value areas and increases the density of warm tones, demonstrating its ability to select more representative attention heads and produce more stable and reliable voting results. These ablation studies indicate that representative heads selection

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
Qwen 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 3: Comparison on the Selection of Parameters g and c.

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	99.1	99.0
CSA	100	99.3	99.0	99.0

Table 4: Sequence Length Scaling.

enhances the stability of top p token prediction and the algorithm’s performance across multiple datasets by optimizing the quality of consensus voting.

5.3.2 Community Size Ablation(RQ2)

To demonstrate the impact of community size on model performance and to support our parameter choices for g and c, we conducted community size ablation experiments, as shown in Table 4. The strategy of community partitioning effectively enhances the performance of CSA. Moreover, we observed that more partitions are not always better, as over-partitioning leads to a reduction in the number of attention heads sampled per community, thereby diluting the consensus of attention heads to a few determinative heads. Similarly, the number of selected heads does not need to be excessively high; after a certain number is reached, the model’s performance no longer improves significantly, while the computational cost increases markedly. Based on experimental results, we selected $g=8, c=1$, for LLaMA-3 8B, $g=16, c=2$, for LLaMA-3 70B, and $g=16, c=2$ for Qwen 14B.

5.3.3 Sequence Length Scaling(RQ3)

We assessed the performance of CSA on long-sequence problems using the "needle in a haystack" experiment. We concatenating the contexts from SQuAD to form a context of a specified length, and subsequently inserting a "text needle" at a specific depth within this context. The model’s capability for long-text processing was evaluated by retrieving the text needle from the context. The detailed

implementation of this task was kept consistent with the methods described in (gkamradt, 2023) and (Contributors, 2023). As shown in Table 5.3.3, experiments show that CSA can maintain good stability under Sequence Length Scaling conditions.

5.3.4 Computational Efficiency and Memory Optimization(RQ4)

To evaluate practical computational consumption, we measured the attention operations in attention layers using uniform PyTorch implementations. As shown in Figure 1e, CSA achieves a near 2-fold acceleration over Dense models in high-batch and long-text scenarios, with further optimization potential given nonoptimal attention implementations. Currently, by avoiding the computation of the full attention matrix, CSA significantly reduces peak memory usage - our fp16 experiments in Figure 1f demonstrate up to 50% memory savings, enabling higher batch sizes for efficient inference.

6 Limitations

When calculating attention, CSA needs to compute the representative attention heads and the remaining attention heads in a serial manner. This approach may not fully utilize the computational power of the machine in low-batch, short-text scenarios.

7 Conclusion

In this work, we propose the Consensus Sparse Attention (CSA) mechanism, a new technique for accelerating the inference of large language models (LLMs). By leveraging consensus voting among representative attention heads for important tokens, CSA predicts potential key tokens in the remaining attention heads. CSA employs a top-p tokens mechanism, reducing the decline in attention concentration under long text conditions. By utilizing consensus among attention heads, CSA can simultaneously reduce computational load and peak memory consumption, providing significant efficiency gains. In the experimental section, we demonstrate the robustness of CSA across numerous tasks and models, indicating that CSA has the potential to become a reliable technique for reducing inference time and memory consumption in the future.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai.

566	2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. <i>arXiv preprint arXiv:2305.13245</i> .	Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. Knowledge distillation of large language models. <i>arXiv preprint arXiv:2306.08543</i> .	621
567			622
568			623
569	Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. <i>arXiv preprint arXiv:2004.05150</i> .	Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2023. Lm-infinite: Simple on-the-fly length generalization for large language models. <i>arXiv preprint arXiv:2308.16137</i> .	624
570			625
571			626
572	Junbum Cha, Wooyoung Kang, Jonghwan Mun, and Byungseok Roh. 2024. Honeybee: Locality-enhanced projector for multimodal llm. In <i>Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition</i> , pages 13817–13827.	Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. <i>arXiv preprint arXiv:2009.03300</i> .	627
573			628
574			629
575			630
576			631
577	Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2024. A survey on evaluation of large language models. <i>ACM Transactions on Intelligent Systems and Technology</i> , 15(3):1–45.	Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. <i>arXiv preprint arXiv:2305.02301</i> .	632
578			633
579			634
580			635
581			636
582			637
583	Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. <i>arXiv preprint arXiv:2107.03374</i> .	Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. 2023. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. In <i>Advances in Neural Information Processing Systems</i> .	638
584			639
585			640
586			641
587			642
588			643
589	Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. <i>arXiv preprint arXiv:1904.10509</i> .	Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. <i>arXiv preprint arXiv:1705.03551</i> .	644
590			645
591			646
592			647
593	Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. <i>arXiv preprint arXiv:2110.14168</i> .	Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. 2022. Sparse upcycling: Training mixture-of-experts from dense checkpoints. <i>arXiv preprint arXiv:2212.05055</i> .	648
594			649
595			650
596			651
597			652
598	OpenCompass Contributors. 2023. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass .	Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, and Dan Alistarh. 2022. The optimal bert surgeon: Scalable and accurate second-order pruning for large language models. <i>arXiv preprint arXiv:2203.07259</i> .	653
599			654
600			655
601			656
602	Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In <i>International Conference on Machine Learning</i> , pages 10323–10337. PMLR.	Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In <i>Proceedings of the 29th Symposium on Operating Systems Principles</i> , pages 611–626.	657
603			658
604			659
605			660
606	Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. <i>arXiv preprint arXiv:2210.17323</i> .	Mo Li, Songyang Zhang, Yunxin Liu, and Kai Chen. 2024. Needlebench: Can llms do retrieval and reasoning in 1 million context window? <i>Preprint</i> , arXiv:2407.11963.	661
607			662
608			663
609			664
610	Ze-Feng Gao, Peiyu Liu, Wayne Xin Zhao, Zhong-Yi Lu, and Ji-Rong Wen. 2022. Parameter-efficient mixture-of-experts architecture for pre-trained language models. <i>arXiv preprint arXiv:2203.01104</i> .	Nian Li, Chen Gao, Yong Li, and Qingmin Liao. 2023. Large language model-empowered agents for simulating macroeconomic activities. <i>arXiv preprint arXiv:2310.10436</i> .	665
611			666
612			667
613			668
614	Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model tells you what to discard: Adaptive kv cache compression for llms. <i>arXiv preprint arXiv:2310.01801</i> .		669
615			670
616			671
617			672
618	gkamradt. 2023. Llmtest needle in a haystack - pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack .		673
619			674
620			675

using constant values of 64 and 32 results in a significant performance drop compared to the method that adapts to sequence length proportionally. This indicates that the adaptive method, which adjusts according to sequence length, offers more stable performance.

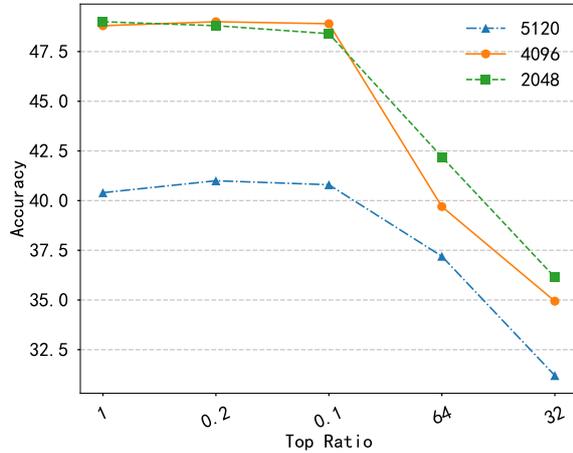


Figure 3: p Ablation

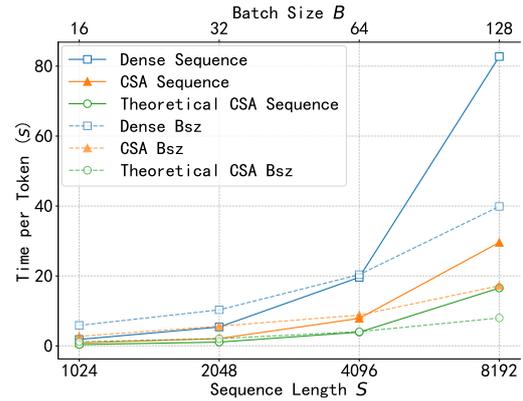
A.2 CSA on Various devices

To comprehensively verify the applicability of the computational efficiency optimization method CSA on different platforms, we have extended the scope of our experiments to other computing environments, building on our analysis of computational load and peak memory tests on the A100 accelerator card (see Chapter 5.3.4). We selected the Intel(R) Xeon(R) Platinum 8163 CPU and the NVIDIA RTX4090 GPU for additional experiments to systematically examine the performance of CSA across different computing architectures. As shown in Figure 4, the experimental data indicate that CSA can still provide effective efficiency improvements on both the CPU platform and the consumer-grade GPU platform.

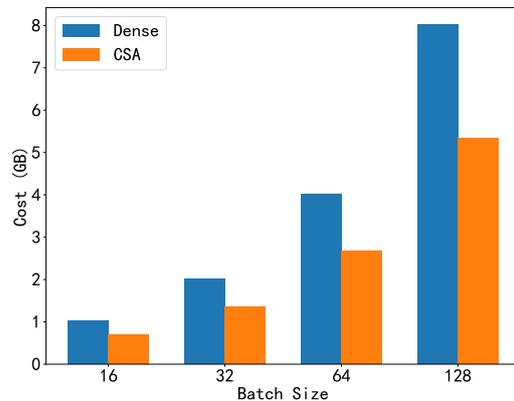
B Experiment Detailed

In this section, we will provide further explanations for all the experimental configurations mentioned in the main text.

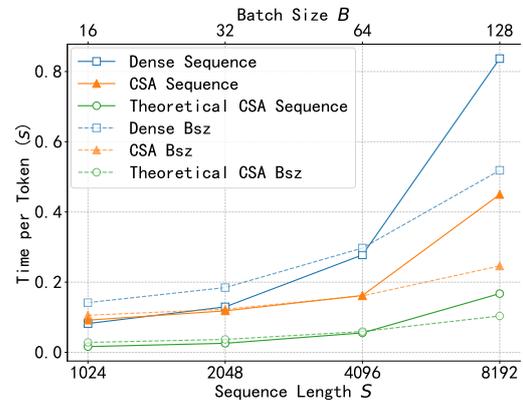
In Section 5.2, we adopted a 5-shots approach for evaluation on MMLU, Ceval, and Gsm8K. Specifically, we constructed a multi-round dialogue prompt, consistent with the approach in opencompass. For evaluation metrics, we used exact match for MMLU and Ceval, while for other evaluations, we used the same Evaluator as in opencompass



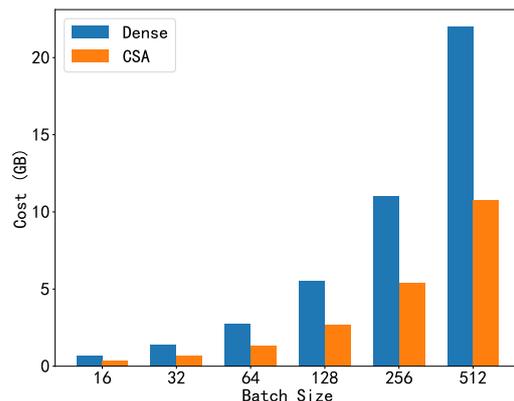
(a) cpu time analysis



(b) cpu memory analysis



(c) RTX 4090 time analysis



(d) RTX 4090 memory analysis

(Contributors, 2023). For the baseline, we set the r value of SparQ to 32 to ensure the same compression ratio as CSA, thereby providing a fair comparison.

In Section 5.3.1, to compare with the representative attention head selection algorithm in CSA, we adopted a method of randomly selecting representative attention heads. Specifically, we randomly selected g attention heads from all attention heads as representative attention heads. This method does not allow further community division, so the number of randomly selected communities is 1.

In Section 5.3.2, we conducted comparative experiments on the selection of g and c on llama3 8b on MMLU, using exact match accuracy as the evaluation metric.

In Section 5.3.3, we selected "needle in haystack" as the evaluation task. Our evaluation followed the approach in opencompass, where we first concatenated a context of specified length from SQuAD, then inserted a needle into the context for evaluation. For evaluation metrics, we directly used the opencompass Evaluator.

In Section 5.3.4, since the performance bottleneck of large language models is mainly concentrated in the prefilling stage, our experiments were also conducted in the prefilling stage. Regarding computational efficiency, since CSA only modifies Core Attention, we focused on the changes in computational efficiency in the Core Attention part. To ensure fairness in comparison, we aimed to compare on the same code implementation. Since CSA provides gains only in the QKV computation, we chose to stat the computational efficiency in the QKV computation. In terms of memory efficiency, CSA provides gains by reducing peak memory, so for the kv Cache of the same size present in both Dense and CSA, we chose to subtract the impact of the KV Cache in both Dense and CSA experiments. Finally, we recorded the time consumed in Core Attention during the first token inference under different Sequence Length and Batch size, as well as the corresponding changes in peak memory.

Since the selection of p is not the main content of CSA, we provided comparative experiments in A.1. The experiments showed that the adaptive method by percentage is more effective than using a fixed constant.

In the preparation of community division and representative attention selection, we chose Queries from multiple evaluation sets when calculating QHR to enhance statistical reliability. For parame-

ter settings, to align with formal experiments, we chose p=0.1. In the calculation of HR and QHR, we chose the temperature coefficient t as 1. When constructing the Graph of attention head nodes, we set j to 2. In community division, we directly used the SpectralClustering implementation of sklearn.

C Case Study

C.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
```

C.2 LLaMA-3 8b(HumanEval)

```
###Question
HUMAN:
from typing import List\n\n\ndef
separate_paren_groups(paren_string:
str) -> List[str]:\n """ Input to
this function is a string containing
multiple groups of nested
```

917	parentheses. Your goal is to\n	add it to the list and reset the	969
918	separate those group into separate	current group\n if balance == 0:\n	970
919	strings and return the list of those	groups.append(current_group)\n	971
920	.\n Separate groups are balanced (current_group = ""\n\n # Return the	972
921	each open brace is properly closed)	list of separated groups\n return	973
922	and not nested within each other\n	groups\n	974
923	Ignore any spaces in the input	CSA:	975
924	string.\n >>> separate_paren_groups	from typing import List\n\ndef	976
925	(\'() (()) (()())\')\n [\'()\',	separate_paren_groups(paren_string:	977
926	\'(())\', \'((())\')\n	str) -> List[str]:\n ""\n This	978
927	###Answer:	function separates multiple groups	979
928	DENSE:	of nested parentheses into separate	980
929	from typing import List\n\ndef	strings.\n\n Args:\n paren_string (981
930	separate_paren_groups(paren_string:	str): A string containing multiple	982
931	str) -> List[str]:\n ""\n This	groups of nested parentheses.\n\n	983
932	function takes a string containing	Returns:\n List[str]: A list of	984
933	multiple groups of nested	separate strings, each representing	985
934	parentheses as input.\n It separates	a group of nested parentheses.\n	986
935	those groups into separate strings	""\n \n # Remove spaces from the	987
936	and returns the list of those.\n	input string\n paren_string =	988
937	Separate groups are balanced (each	paren_string.replace(" ", "")\n\n #	989
938	open brace is properly closed) and	Initialize an empty list to store	990
939	not nested within each other.\n It	the separated groups\n	991
940	ignores any spaces in the input	separated_groups = []\n\n #	992
941	string.\n\n Parameters:\n	Initialize an empty string to build	993
942	paren_string (str): A string	the current group\n current_group =	994
943	containing multiple groups of nested	""\n\n # Initialize a counter for	995
944	parentheses.\n\n Returns:\n List[open and close parentheses\n	996
945	str]: A list of strings, each	open_count = 0\n close_count = 0\n\n	997
946	containing a group of balanced	# Iterate over each character in	998
947	parentheses.\n ""\n\n # Remove	the input string\n for char in	999
948	spaces from the input string\n	paren_string:\n # If the character	1000
949	paren_string = paren_string.replace	is an open parenthesis, increment	1001
950	(" ", "")\n\n # Initialize an empty	the open count\n if char == "(":	1002
951	list to store the separated groups\n	open_count += 1\n current_group +=	1003
952	groups = []\n\n # Initialize an	char\n # If the character is a close	1004
953	empty string to build the current	parenthesis\n elif char == ")":	1005
954	group\n current_group = ""\n\n #	close_count += 1\n current_group +=	1006
955	Initialize a counter to keep track	char\n # If the current group is	1007
956	of the balance of parentheses\n	balanced, add it to the list and	1008
957	balance = 0\n\n # Iterate over each	reset the current group\n if	1009
958	character in the input string\n for	open_count == close_count:\n	1010
959	char in paren_string:\n # If the	separated_groups.append(1011
960	character is an open parenthesis,	current_group)\n current_group = ""\n	1012
961	increment the balance\n if char ==	n open_count = 0\n close_count = 0\n	1013
962	"(":	# If the current group is not	1014
963	balance += 1\n current_group	balanced, reset the current group\n	1015
964	+= char\n # If the character is a	if open_count < close_count:\n	1016
965	close parenthesis, decrement the	current_group = ""\n open_count = 0\n	1017
966	balance\n elif char == ")":	n close_count = 0\n\n # Add the last	1018
967	balance -= 1\n current_group += char	group to the list if it\'s not	1019
968	\n # If the balance becomes zero, it	empty\n if current_group:\n	1020
	means we have a complete group, so		

1021
1022
1023

1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072

```
separated_groups.append(  
current_group)\n\n return  
separated_groups\n
```

C.3 LLaMA-3 8b(GSM8k)

```
###Question  
HUMAN:  
Question: \nJosh decides to try flipping  
a house. He buys a house for $80  
,000 and then puts in $50,000 in  
repairs. This increased the value of  
the house by 150%. How much profit  
did he make?\nLet's think step by  
step\nAnswer:  
The cost of the house and repairs came  
out to 80,000+50,000=$  
<<80000+50000=130000>>130,000\nHe  
increased the value of the house by  
80,000*1.5=<<80000*1.5=120000>>120,000\nSo the new value of the house is  
120,000+80,000=$  
<<120000+80000=200000>>200,000\nSo  
he made a profit of 200,000-130,000=  
$<<200000-130000=70000>>70,000\n###  
70000\n...few shots  
HUMAN:  
Question:'  
Janets ducks lay 16 eggs per day. She  
eats three for breakfast every  
morning and bakes muffins for her  
friends every day with four. She  
sells the remainder at the farmers'  
market daily for $2 per fresh duck  
egg. How much in dollars does she  
make every day at the farmers'  
market?  
Let's think step by step  
Answer:  
###Answer:  
DENSE: Janet's ducks lay 16 eggs per day  
. She eats 3 eggs for breakfast and  
bakes muffins with 4 eggs, leaving:\n\n16 - 3 - 4 = 9 eggs\n\nShe sells  
these 9 eggs at the farmers' market  
for $2 per egg, so she makes:\n\n9  
eggs x $2 per egg = $18\n\n### $18  
CSA: Janet's ducks lay 16 eggs per day.  
She eats 3 for breakfast, and bakes  
muffins with 4, leaving her with:\n
```

```
n16 - 3 - 4 = 9 eggs\n\nShe sells  
these 9 eggs at the farmers' market  
for $2 per egg, making:\n\n9 eggs x  
$2 per egg = $18\n\n### 18
```

1073
1074
1075
1076

C.4 LLaMA-3 8b(TriviaQA)

```
###Question  
HUMAN:  
Answer these questions, your answer  
should be as simple as possible,  
start your answer with the prompt '  
The answer is ' .  
Q: Which Lloyd Webber musical premiered  
in the US on 10th December 1993?  
###Answer:  
DENSE: The answer is "Sunset Boulevard".  
CSA: The answer is "Sunset Boulevard".
```

1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090

C.5 LLaMA-3 8b(SQuAD)

```
###Question  
HUMAN:  
The Normans (Norman: Nourmands; French:  
Normands; Latin: Normanni) were the  
people who in the 10th and 11th  
centuries gave their name to  
Normandy, a region in France. They  
were descended from Norse ("Norman"  
comes from "Norseman") raiders and  
pirates from Denmark, Iceland and  
Norway who, under their leader Rollo  
, agreed to swear fealty to King  
Charles III of West Francia. Through  
generations of assimilation and  
mixing with the native Frankish and  
Roman-Gaulish populations, their  
descendants would gradually merge  
with the Carolingian-based cultures  
of West Francia. The distinct  
cultural and ethnic identity of the  
Normans emerged initially in the  
first half of the 10th century, and  
it continued to evolve over the  
succeeding centuries.\nAccording to  
the above passage, answer the  
following question. If it is  
impossible to answer according to  
the passage, start your answer with  
the prompt '\nThe answer is '\.answer  
'impossible to answer`:
```

1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123

1124 Question: In what country is Normandy
1125 located?
1126 ###Answer:
1127 DENSE: France.
1128 CSA: France.