Attention スコアの分布類似性を用いた大規模言語モデルの動作効率化および省メモリ化

谷口 令[1] 肖 川[1,2] 董 于洋[3] 小山田 昌史[3] 鬼塚 真[1]
[1] 大阪大学大学院情報科学研究科 [2] 名古屋大学 [3] NEC
{taniguchi.rei,chuanx,onizuka}@ist.osaka-u.ac.jp

概要

Transformer-デコーダをベースとした大規模言語モデルは、文脈理解・回答における精度の高さから活用が広がっている。近年では長文入力への需要が高まっているが、自己注意処理 (Attention) における計算量が入力長の 2 乗に比例すること、KV キャッシュのサイズが入力長に比例するという理由により、入力長に対して制限が存在する。本研究では Transformer 内の各レイヤ・各ヘッドについて注意スコアの分布を調査し、同一レイヤにおいてスコアを共有できるヘッドが存在すること、また各レイヤについてスコアが大きなトークンが共通していることを明らかにした。これを元に自己注意処理の効率化と KV キャッシュの削減を行い、削減前と比較してベンチマークにおける精度低下が限定的であることを確認した。

1 はじめに

現在、GPT や Llama, Mistral といった LLM(Large Language Model) が開発され、一部は OSS で誰でも利用・改良できるようになっている。

これらのモデルは活性化関数やレイヤ正規化の手法、学習データ数の差などはあれど基本的な構造は Transformer-デコーダと変わらず、Attention ヘッドと呼ばれる処理機構を並列に持つ Transformer ブロックが複数個直列に繋がれたモデルである。

基本的な Transformer-デコーダモデルの Attention ヘッド 1 つあたりの Prefill 処理時 (初回のプロンプト入力時) の Attention 処理の計算は、以下のような式で表される。

$$\text{Output} = \text{Softmax}\left(\frac{QK^\top + \text{Mask}}{\sqrt{d}}\right)V \tag{1}$$

ここで **Q**,**K**,**V** はそれぞれクエリ、キー、バリュー行列であり、**d** は Query ベクトルの次元数、Mask は値が 1、大きさが入力長と等しい下三角行列である。

KV キャッシュを利用した Decode 手法では式 1 は以下のように置き換えられる。

$$K_{\text{cache}} \leftarrow [K_{\text{cache}}, k], \quad V_{\text{cache}} \leftarrow [V_{\text{cache}}, v] \tag{2}$$

$$\text{Output} = \text{Softmax}\left(\frac{qK_{\text{cache}}^\top}{\sqrt{d}}\right)V_{\text{cache}} \tag{3}$$

これにより Decode 時には **q**,**k**,**v** ベクトルのみを計算すればよいため生成速度の向上が期待できる。反面、式 2 のキャッシュサイズは入力長に応じて線形に増加するため、プロンプトが長大な場合はメモリ (GPU を利用する場合は VRAM) を圧迫する課題が存在している。

また **Q**,**K**,**V** の作成や FeedForward 処理の計算量は $\mathbf{O}(\mathbf{n})$ であるのに対して、式 1,3 の計算量は $QK^T \in \mathbb{R}^{N\times N}$ のため $\mathbf{O}(\mathbf{n^2})$ となる。そのため既存の Transformer モデルでは、入力長に応じてメモリ使用量が線形に増加し、また計算量が入力長の二乗に比例して大きくなる課題が存在している。

近年では LLM の活用方法として、本や映画の脚本の要約・複数書類を参照対象とした質問タスクなどが期待されていて、その要求に対して長大なコンテキスト長で学習した LLM モデルも発表されている。しかし上記のような問題により、長文タスクを解く場合はメモリ量・計算力ともに強大な計算機を所持する必要があり、LLM の長文タスクへの応用の妨げとなっている。

2 事前実験

省メモリ化、動作速度効率化を実現する手法を作成する前段階として、Attention 処理の挙動について llama3 を用いて観察を行った。それにより確認したことを以下に示す。

2.1 同一レイヤ内における各ヘッドの Attention Map の類似性

llama3[1)] を用いて longbench[1] 中のタスクを解く際の prefill 時における各レイヤ・各ヘッドの

1) https://ai.meta.com/blog/meta-llama-3/

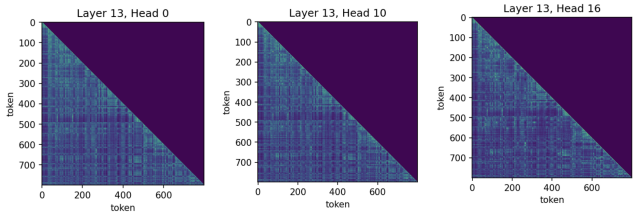

図 1 llama3 の 13 レイヤー目の Attention 行列 (第 0,10,16 のヘッドのみ)

Attention 行列を出力し、その分布について確認を行った。例として１３層目の Transformer レイヤの Attention 行列の出力を付録 A の図 5 に示した。ここで llama3 は 32 個の Transformer レイヤを持ち、各レイヤについて 32 個の Attention ヘッドを持つ。

各レイヤの Attention 行列を観察した上で、同一レイヤ内では類似した分布を持つように見られる Attention 行列が複数存在している事が確認できた。

このような類似したスコア分布を持つヘッドが存在する理由として、ヘッド数はハイパーパラメータであり事前に決定されるが、単語生成において重要なトークンはどのヘッドにおいても共通しているため冗長性が発生し、一部のヘッドは学習の過程で同一の分布となるようにパラメータが更新されたためと考えられる。

類似した Attention 行列分布を持つヘッドを特定することを考える。Attention 行列同士の類似性を値 distance として表すために式 4 を利用することを考える。ここで入力長を N, 類似度を比較したい Attention 行列同士をそれぞれ $\mathbf{A}$,$\mathbf{A}'$ と表した。式 4 では行列内の各要素ごとの差を累積して正規化を行い、この値を distance と定義している。この distance がしきい値 th(thresh_hold) 以下であれば類似しているとみなす。

$$\text{distance} = \frac{\sqrt{\sum (A - A')^2}}{\sqrt{N}} \tag{4}$$

例として、第 13 レイヤ目にて式 4 を用い th=0.18 として同一レイヤ内の Attention 行列のクラスタリングを行った場合はインデックス [0, 10, 16] のヘッドによる Attention 行列の類似度が th を下回った。これらのヘッドの Attention 行列を図 1 に示した。

これらの Attention 行列は、目視では類似した分布を持っているように見受けられる。

以上より、式 4 を用いることで類似度により Attention 行列をクラスタリングできることが確認できた。また llama3 は MHA に GQA(Grouped Query Attention)[2] を利用しているが、各グループ中のヘッドの Attention 行列について類似したスコア分布が顕著に現れるということは無く、類似したスコアを持つヘッドの番号に規則性は確認できなかった。

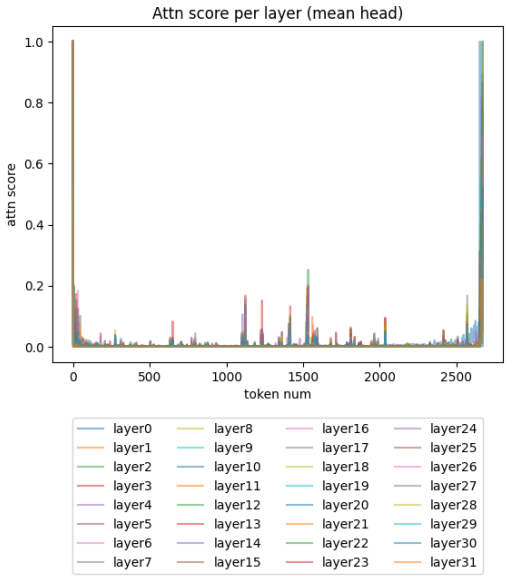

図 2 各レイヤの平均 Attention スコアの分布。初期トークンに Attention が集中しているのは、これらのトークンが Attention sink[4] として動作しているためと考えられる。

2.2 各レイヤにおける Attention スコアの類似性

同一レイヤ内の全てのヘッドの Attetion スコアの平均は、その Transformer レイヤがどのトークンに注意を向けているのかを表す指針であると考えることができる。

llama3 のレイヤ 0 から 31 までの各レイヤの平均スコアをグラフとして図 2 に表した。ここで longbench[1] を入力とした。

また先行研究 [3] により、実際の重要度としての Attention スコアは Value のノルムの存在も重要でであることが提案されているため、グラフには Attention スコアに Value ノルムを乗算した。さらに可視性を高めるためにスコアの最大値で正規化を行った。

図より、多くのレイヤにおいて Attention スコアが大きいトークンは共通していることが確認できる。これは Prefill 処理だけでなく、その後の Decode 処理においても同じ傾向が見受けられた。また初期レイヤ (0-10) の分布と比べると後半レイヤ 11-31) のレイヤにおける Attention スコアの分布の方が分布が類似している点が見受けられた。これは初期レイヤで重要トークンが決定された後、後半レイヤではそれら重要トークンに重点をあててデータの変換処理・及び出力処理が行われるためであると考えられる [5]。

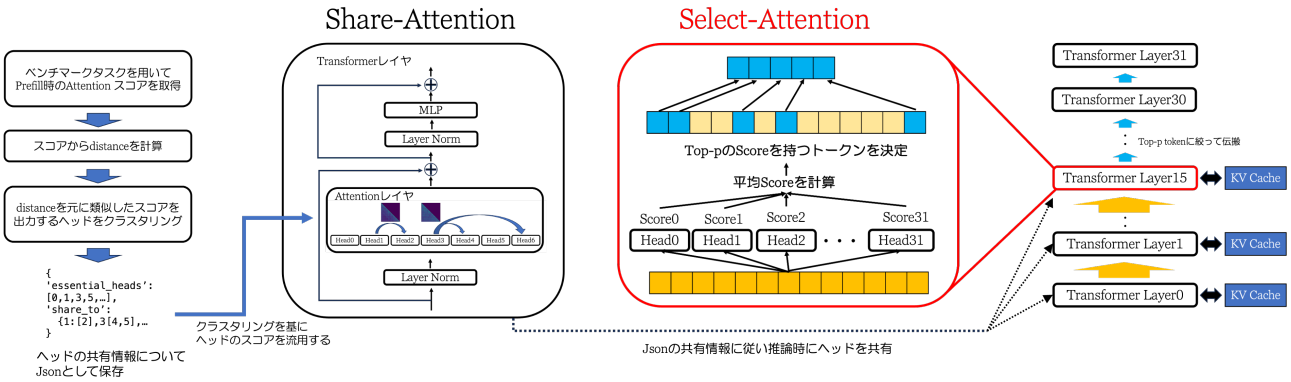

図 3　提案手法 (Share-Attention と Select-Attention) の動作概要

3　提案手法

前章において確認した事項を元に、動作効率化のための手法として **Share-Attention**、メモリ使用量削減のための手法として **Select-Attention** を提案する。

提案手法の概要を図 3 に示した。

3.1　Share-Attention

図 1 に示したように、同一レイヤ内には類似した Attention スコアを出力するヘッドが複数存在している。そこでこれらのヘッドについてはスコアの共有が可能と仮定し、1 つのヘッドのみ計算を行いその結果を共有することを考える。

具体的な流れは以下のようである。

まずオフラインで共有できるヘッドを決定する。

1. ベンチマーク (longbech) を回答中の LLM の各レイヤ・ヘッドの Attention 行列を保存する。
2. 式 4 を用いて distance の計算を行う。
3. distance を基に類似したスコアを出力するヘッドをクラスタリングし、Attention 処理を行うヘッド (図中では essential_heads と記載) と処理結果をどのヘッドに共有するか (図中では share_to と記載) を示した json を作成する。

ここで作成した json を基に、LLM を動作させる時にヘッドの共有を行う。

Share-Attention を行うことで Attention 処理の計算回数を減らし、生成速度を早くすることができると考えられる。

3.2　Select-Attention

前章にて、Attention スコアは各レイヤにおいて類似性が見られること、また特に後半レイヤの類似性が大きいを確認した。

Attention スコアがスパースであること・一部の重要トークンが大きなスコアを持つことを踏まえると、中間層レイヤにおいて Attention スコアが大きいトークンを上から順に選択した後に、これらのトークンのみを後半レイヤに伝搬させることで Attention 処理を近似できると考えられる。

このようなトークンのフィルターとして動作するレイヤについて、ベンチマークでの検証により llama3 では 15 層目が最も適切であることが確認できた。そのため今後の検証ではこのレイヤをフィルターとして用いる。

また先行研究 [5] ではトークンを厳選するために Top-k 手法 (スコアが高いものから順に k 個取得する) を利用していたが、Top-K では入力が長くなったときに適切に重要なトークンを取り出すことが難しいと考えられる。そのため本研究では Top-p 手法を採用する。これはスコアが Softmax で正規化されていて合計が 1 になることを前提として、合計スコアが p 以上になるまでトークンを取り出す手法である。

本手法では前半レイヤのみ KV キャッシュを持ち、後半レイヤはキャッシュを持たない。そのためキャッシュの大きさが半分となるため、VRAM の削減につながると考えらえる。

4　検証

前章で提案した Share-Attention および Select-Attention の有用性をベンチマークを用いて検証を行う。検証には llama3 を長文入力 (最大 128k トークン) に対応させたモデルである llama3.1[2] を用いた。

まず Share-Attention について検証を行った。

th の値を大きくすれば共有できるヘッドの数は多

2)　https://ai.meta.com/blog/meta-llama-3-1/

表 1　Share-Attention による Longbench の結果

手法	Multi_news	Qasper	2wikiqa	Hotpotqa
初期状態	26.79	43.88	40.62	50.97
ヘッド維持率 81.6%	26,27	40.25	38.65	45.54
ヘッド維持率 69.1%	25.84	37.05	34.43	43.59

くなるが、代わりにスコアの近似値が低下するため回答精度が悪くなると考えられる。そこで MMLU の high_school_european_history_test を解く際に 1 つのレイヤのみ th=0.9 と極端に高い値に設定して共有するヘッド数を多くし、それ以外は th=0 として共有しない設定にて回答精度を比較した。その結果を付録 B の図 6 に示した。

その結果、前半レイヤ (0 から 14) について共有ヘッド数を多くすると回答精度は低下し、後半レイヤ (15 31) についてはそこまで低下しないことが確認できた。

また最終レイヤ (29 から 31) について th=0.9 とした場合は四択問題である MMLU では問題なかったが、長文回答が要される場合のある Longbench では上記の設定では回文や散文を生成してしまう現象が見受けられた。これは最終レイヤは回答精度についてはそこまで重要ではないが、回答フォーマットについては重要な決定権を持っているためと考えられる。

上記考察を踏まえ、Longbench を実行する際は前半と後半レイヤについては th を小さく・中盤レイヤでは大きく設定を行い共有ヘッド数を変化させて検証を行った。このときの各レイヤにおけるヘッド維持率 (retention rate, 100× 計算を行うヘッド数/本来のヘッド数) を付録 B の図 7 に、検証結果を表 1 に表した。

表より、QA タスクである Qasper,2wikiqa,Hotpotqa ではヘッド維持率が低下するにつれてスコアが下がったが、要約タスクである Multi_news では精度が維持されていることが確認できた。また図 7 より前半・中〜後半レイヤはヘッドの共有率が大きくても問題ないことが、逆に前半〜中盤レイヤでは共有できるヘッドが少ないことが確認できた。特に平均ヘッド維持率が 69.1%の場合において 20〜29 レイヤにて 80%程度のヘッドを削減した場合でも動作に問題がないことが確認できた。

次に Select-Attention について、top-p について p の値を変化させて検証を行った。検証結果を付録 B の表 2 に示した。表には Prefill 時にトークンの厳選を行う手法である GemFilter[5] での結果も記載した。

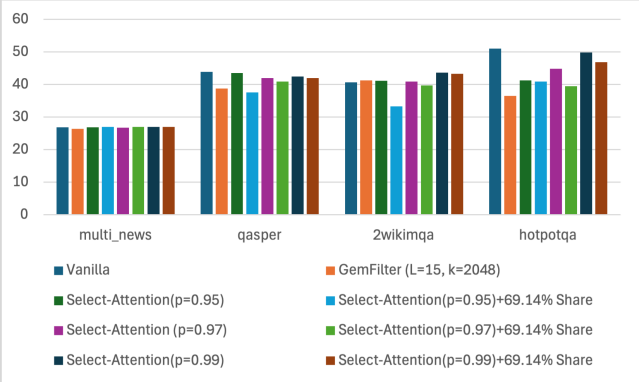

図 4　Vanilla(基準モデル),Gemfilter, Select-Attention 及び Share-Attention を併用した場合の Longbench のスコア比較

表より、Multi_news や Hotpotqa を除く QA タスクについては、全トークン中の 1 割程度のトークンのみの伝搬でもスコアに差が無いことが確認できた。

上記の結果及び Select-Attention,Share-Attention を併用した結果を図 4 に示した。

図より、要約タスク (Multi_news) ではどの手法でもスコアに差がないことが確認できた。また QA タスクにおいても、Share-Attention と Select-Attention を併用した場合については、p=0.95(伝搬するトークンは 1 割程度), ヘッドを 3 割程度削っても基準モデルと比較して 8 割程度のスコアを維持できていることが確認できた。

生成速度については、CUDA に最適化されている手法である Flash_Attention[6] に Share-Attention 機能を追加できていないため計測を行っていない。本手法により計算量は削減されているため CPU 上での動作では高速化が可能であり、また GPU 上での動作においては Flash_Attention を Share-Attention に対応させることで高速化が可能であると思われる。

また本実験では Longbench 内の平均長が 10k 程度のタスクを利用して検証を行った。この理由として、実験環境 (RTX A6000 Ada) においてはこれ以上のトークン数を持つタスクでは OOM(Out of Memory) が発生してしまい検証ができなかったことが挙げられる。この問題についても、Flash_Attention[6] では Attention スコアの計算をバッチに分けて処理を行い長期入力に対応しているため、Flash_Attention を Share-Attention に対応させる改良を行うことにより NarrativeQA(平均長 18k) や Needle-in-a -Haystack[7](最大 128k) においても検証が可能になると思われる。

謝辞

本研究は主にＮＥＣ (日本電気株式会社) の支援を受け，JSPS 科研費 JP23K17456, JP23K25157, JP23K28096 の助成, および JST, CREST, JPMJCR22M2 の支援を受けたものです。

参考文献

[1] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. **arXiv preprint arXiv:2308.14508**, 2023.

[2] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. **arXiv preprint arXiv:2305.13245**, 2023.

[3] Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. Attention Score is not All You Need for Token Importance Indicator in KV Cache Reduction: Value Also Matters, October 2024. arXiv:2406.12335.

[4] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks, April 2024. arXiv:2309.17453.

[5] Zhenmei Shi, Yifei Ming, Xuan-Phi Nguyen, Yingyu Liang, and Shafiq Joty. Discovering the Gems in Early Layers: Accelerating Long-Context LLMs with 1000x Input Token Reduction, September 2024. arXiv:2409.17422.

[6] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. **Advances in Neural Information Processing Systems**, Vol. 35, pp. 16344–16359, 2022.

[7] Elliot Nelson, Georgios Kollias, Payel Das, Subhajit Chaudhury, and Soham Dan. Needle in the haystack for memory based large language models, 2024.

A　事前準備

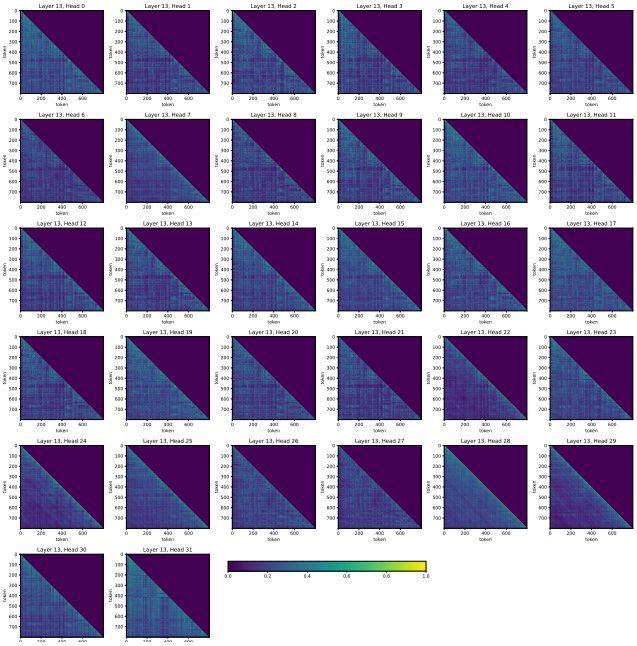

図 5　llama3 の 13 レイヤー目の各ヘッドの Attention 行列

B　実験結果

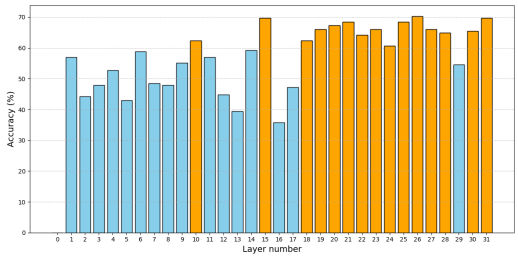

図 6　th=0.9 とするレイヤを変えてベンチマークを行ったときの回答精度。バニラモデルでは回答精度は 69%である。オレンジ色のグラフは精度が 60%を超えていることを表す。

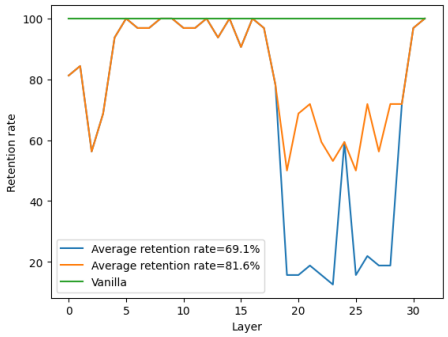

図 7　平均ヘッド維持率が異なる場合の各レイヤにおけるヘッド維持率。

表 2　GemFilter と Select-Attention による Longbench のスコア/トークン維持率。GemFilter は Prefill 時に Top-k トークンを決定して生成時に利用するため、Prefill 時の平均トークン維持率を示している。Select-Attention では生成毎に Top-k トークンを決定しているため、生成時の平均トークン維持率を示している。

手法	Multi_news	Qasper	2wikiqa	Hotpotqa
初期状態	26.79/100%	43.88/100%	40.62/100%	50.97/100%
GemFilter(k=2049)	26.34/74.5%	38.75/55.5%	41.26/40.7%	36.44/22.29%
Select-Attention(p=0.99)	26.9/34.9%	42.48/29.7%	43.65/20.7%	49.78/21.5%
Select-Attention(p=0.97)	26.73/19.2%	41.94/11.4%	40.9/7.0%	44.88/7.16%
Select-Attention(p=0.95)	26.81/13.5%	43.51/6.9%	41.18/3.5%	41.27/3.58%