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[2] 董 于洋[3]
小山田 昌史[4] 鬼塚 真[5]

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。近年では映画の脚本や小説、複数の社内資料などを入力として要約や QA タスクを行わせる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。自己注意処理の効率化・省メモリ化手法として、全てのトークンを利用する代わりに厳選して保存・利用する、クエリに類似したキーのみを利用するといった手法が提案されているが、これらは各ヘッドにおいて生成時に別途計算処理を行う必要があり得られる効果は限定的である。本研究では Transformer-デコーダを採用した大規模言語モデル内の各レイヤ・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った上で、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

1.1 Transformer 概要

2017 年に発表された Transformer アーキテクチャ [18] は、翻訳を主な目的として開発された。しかし現在ではその用途は翻訳にとどまらず、文章生成や画像認識、制御など多岐にわたっている。

このようなモデル発表から数年しか経過していないにもかかわらず爆発的に普及している理由として、それまで自然言語処理で利用されていた時系列モデルと比べて情報処理能力と学習能力に優れていたことが一因であると考えられる。

時系列モデルの代表例であり古くから存在している RNN(Recurrent Neural Network) では入力をリカレント層に記憶することにより時系列データを処理していたが、この手法では長期的な入力に対して適切に記憶することが難しく、また時系列方向の誤差逆伝播をする際に勾配消失・発散が発生してしまうことが問題とされていた。これを解決する手法として記憶部分にゲート処理機能を付与した LSTM(Long Short Term Memory) [9] や GRU(Gated Reccurent Unit) [4] が提案され、さらに LSTM を用いて翻訳を行うための手法として seq2seq [17] が開発された。

seq2seq はエンコーダとデコーダを組み合わせた系列変換モデルであるが、エンコーダから送られる情報は固定長ベクトルであるため長文入力を適切に伝搬させることが難しい問題があった。そこでエンコーダの出力を全て利用する手法として相互注意機能 (Cross-Attention) を備えた seq2seq with attention [2] が提案された。その後、並列学習の妨げになっている LSTM を排除し自己注意機能 (Self-Attention) のみで構成された系列変換モデルである Transformer が提案された。

Transformer は処理構造が簡素でありながら帰納バイアスが小さく汎用性があること、また Self-Attention を行うヘッドの数や Transformer ブロック数を増やすことにより回答精度が向上すること (スケーリング則) が確認されたこと、さらに並列処理に適した GPU の急速な大メモリ化、処理高速度化といったことより、発表以降自然言語・画像・音声などのタスク向けの大規模モデルが開発されている。

自然言語処理分野では特に、OpenAI によって Transformer-デコーダのみを利用し Attention mask を用いて文章生成を行う GPT(Generative Pretrained Transformer) [15] やそれに続く GPT2 が開発されて以降、Transformer-デコーダベースの大規模言語モデル (Large Language Model, LLM) が企業や研究機関により開発されている。これらの LLM は Instruction-Tuning や人間のフィードバックによる強化学習 (Reinforcement Learning from Human Feedback、RLHF) が適用されることにより人間と寸分違わぬやり取りが可能になり、実際の業務や QA タスクなどで用いられるようになっている。

1.2 Transformer を利用した言語モデルの構造と課題

近年は GPT や Llama, Mistral といった大規模言語モデルが開発され、一部はオープンソースモデルとして誰でも利用・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、

- Tokenizer による入力文文章の分割とトークン化
- 複数回にわたる Transformer ブロック処理 (位置表現追加・MLP 処理、Self-Attention 処理、残差結合 など)
- 確率計算による出力トークンの決定

のような処理が反復されることで文章が生成される。具体例としてオープンソースモデルである Llama3 の Transformer ブロックの計算処理は以下のものである。ここで $\mathbf{x}$ は Transformer ブロックへの入力テンソルを表す。

$$h = x + \mathrm{Attention}(\mathrm{RMSNorm}(x)) \tag{1}$$

[1] 学生会員 大阪大学情報科学研究科
taniguchi.rei@ist.osaka-u.ac.jp
[2] 正会員 大阪大学・名古屋大学
chuanx@ist.osaka-u.ac.jp
[3] 正会員 SB Intuitions
yuyang.dong@sbintuitions.co.jp
[4] 正会員 NEC
[5] 正会員 大阪大学情報科学研究科
onizuka@ist.osaka-u.ac.jp

$$\text{Out} = h + \text{FeedForward}(\text{RMSNorm}(h)) \tag{2}$$

また 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、**W** を重み行列として以下のような式で表される。

$$Q = xW_q, \quad K = xW_k, \quad V = xW_v \tag{3}$$

$$Q, K = \text{apply\_rotary\_emb}(Q, K) \tag{4}$$

$$\text{Output} = \text{Softmax}\left(\frac{QK^\top + \text{Mask}}{\sqrt{d}}\right)V \tag{5}$$

ここで **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。また回転位置埋め込みの処理を apply_rotary_emb と表した。

式 3 における計算について、入力 **x** はプロンプトとこれまでに生成した単語トークンに依存しているため、Decode 時 (Prefill 時を除いた単語生成時) の n+1 ステップ目の式 3 の計算は、この時の入力ベクトル **x** を元に新たに作成された **q**,**k**,**v** ベクトルの計算を除いて n ステップ目の計算と全く同じになる。そこで再計算を減らし生成速度を向上させる手法として、既に計算した Key, Value を保存し次の単語生成時に再利用する KV キャッシュがある。

KV キャッシュを利用した Decode 手法では式 3,5 は以下のように置き換えられる。

$$q = xW_q, \quad k = xW_k, \quad v = xW_v \tag{6}$$

$$K_{\text{cache}} \leftarrow [K_{\text{cache}}, k], \quad V_{\text{cache}} \leftarrow [V_{\text{cache}}, v] \tag{7}$$

$$\text{Output} = \text{Softmax}\left(\frac{qK_{\text{cache}}^\top}{\sqrt{d}}\right)V_{\text{cache}} \tag{8}$$

これにより Decode 時には **q**,**k**,**v** 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 7 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する。

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 5,8 の計算量は $QK^T \in \mathbf{R}^{N\times N}$ 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する。例として 8B モデルへの 100 万トークンの入力を Nvidia A100 GPU で行った場合は prefill 処理に約 30 分かかったことが報告されている [10]。

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おり、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 関連研究

ここでは少ないメモリ量・計算量で LLM を動作させるために提案されている手法を説明する。

Transformer-デコーダを利用した大規模言語モデル (LLM) では、

- KV キャッシュの存在がメモリ使用量の線形的な増加を発生
- Attention 内の計算量が $\mathbf{O}(\mathbf{n^2})$

のような理由のため長文入力に対応するのが難しい。

これらの問題は Attention 構造において、n+1 ステップ目のトークン生成においてはこれまでの n 個のトークンの内どれが重要であるかを表した Attention スコアを計算する必要があり、そのために Key とそれに対応した Value を保存する必要があることに由来する。

人が会話をする際にこれまでの会話全てを覚えているのではなく重要な情報のみを記憶して会話をしているように、LLM においても単語生成時において Attention スコアは一様ではなく一部のトークンの値が大きくなることが確認できる。よって Attention 計算を行う前にどのトークンが重要であるかを確認することができれば、重要トークンのみをキャッシュすることでメモリ使用量の削減を、また他トークンを計算しないことで計算の高速化ができると期待される。

このような手法として以下のようなものが提案されている。

2.1 ストリーミングアテンション

Attention スコアは最初期のトークンと近傍のトークン部分について大きくなることが確認されている [20]。近傍トークンについては適切な回答を生成するために必要であり、最初期トークンは Attention スコアのバッファ部分 (必要の無いトークンへのスコアの分布を避ける) として機能すると考えられる。そこで初期の数個のトークンと最近傍の数千個のトークンのみを利用することにより、会話内容の精度を保ちながら LLM の最大入力長までの入力に対して $\mathbf{O}(\mathbf{1})$ の計算量で対応が可能となる。

2.2 Query に類似した Key-Value のみを利用

式 5,8 で表される Attention 処理は内部で Query と Key の内積を計算する。内積はそれぞれのベクトルの類似度が高いほど高い値を示すこと、Softmax 処理により類似度が小さい Query-Key の内積は極めて小さく扱われることを考えると、Query と類似度が高い Key とその Value のみを用いて Attention 処理を行うことで出力を近似できると考えられる。

代表的な例として k-NN などの手法を用いて Key 空間において Query を用いて類似した topk の Key を検索する手法が挙げられるが、この検索自体の計算処理が大きくなってしまい十分な計算速度向上ができない問題が存在する。

そこで Key を適切に量子化して検索計算量を削減する [14]、Query と Key の空間領域の違いを是正して検索効率を向上させる手法 [13] が存在する。

またトークン毎に類似度を計算する代わりに入力をブロックに分割し、ブロックごとに代表トークンを決定して類似度を計算する手法も提案されている [19] [11]。

2.3 Prefill 時に重要トークンを決定

Prefill 時にプロンプト内のトークン全てに対して Attention スコアが計算されるが、このときスコアが大きかったトークンはその後のデコード時においてもスコアが大きい傾向にあることが報告されている [21]。そこで Prefill 時においてスコアが大きいトークンのみを KV キャッシュに保存してデコード時にはキャッシュしたトークンとその後の生成トークンのみを利用することにより、メモリ使用量の削減と動作高速化を行う。

重要トークンを決定する手法として、Prefill 時の Attention スコアを平均する [21], 直前の Attention スコアのみを利用する [16], 近傍の Attention スコアを利用する [12] 手法などが存在する。

3 事前実験

省メモリ化、動作速度効率化のために Attention 処理について Llama3-8B-Instruct モデル (以下 Llama3 と表記)[*1] 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。

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

単一ヘッドでの Attention 計算は式 5,8 で表される。Softmax 関数は合計が 1 になるように入力を正規化して出力するため、単語生成において重要なトークンが複数ある場合でも一部のトークンのみが Attention スコアが大きくなり他の重要なトークンを利用できなくなる問題がある。

そこで Transformer では入力ベクトルを複数個に分割し、分割された各ベクトルに対して Attention を行う手法であるマルチヘッド注意処理 (Multi Head Attention, MHA) を用いることで上記の問題を回避している。

Llama3 を用いて LongBench [3] 中のタスクを解く際の prefill 時の 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 行列の出力を図 1 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。

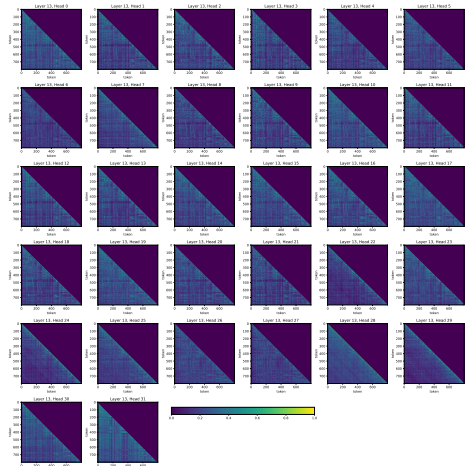

図 1 Llama3 の 13 レイヤ目の Attention 行列

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モデル作成段階において任意に決定されるが、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重みパラメータが学習されたためと考えられる。

次に類似した 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 9 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 9 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。

$$\text{distance} = \frac{\sqrt{\sum(A - A')^2}}{\sqrt{N}} \tag{9}$$

例として、第 13 レイヤ目にて式 9 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 2 に示した。

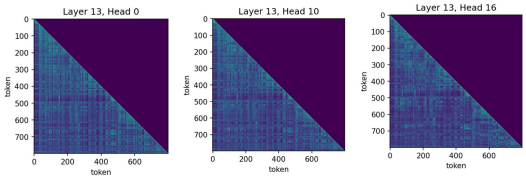

図 2 Llama3 の 13 レイヤー目の Attention 行列 (第 0,10,16 のヘッドのみ)

これらの Attention 行列は、目視ではあるが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 9 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。なお Llama3 は MHA にグループ化クエリー注意処理 (Grouped Query Attention, GQA) [1] を利用しており 1 つの query に対して複数の key-value が作用するが、GQA の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。

3.2 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 3 に表した。ここで LongBench [3] を入力とした。

また先行研究 [6] 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。

[*1] https://ai.meta.com/blog/meta-Llama-3/

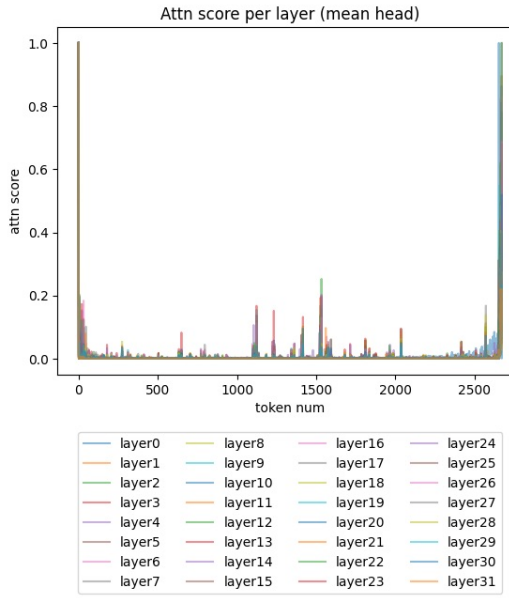

図 3　各レイヤの平均アテンションスコアの分布。初期トークンに Attention が集中しているのは、これらのトークンが Attention sink [20] として動作しているためと考えられる。

図 3 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 [16]。

次に各レイヤ内の平均 Attention スコアの時間軸方向の類似性を調べるために、Prefill 時において Attention 行列の各行内でスコアが高い順にソートした場合の上位 99% のトークンの分布を調査した。ここで例として、レイヤ 15 にて上位 99% トークンを黄色でハイライトした結果を図 4 に示した。

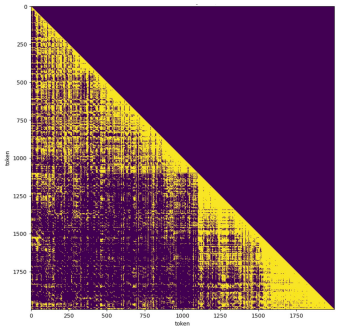

図 4　ベンチマーク実行時のレイヤ 15 での各トークン出力時における、Attention スコアの上位 99% トークンを黄色でハイライトした結果

結果として、Attention スコアのトークン方向の分布については時刻毎に大きく変化し、常に特定のトークンのみがスコアが大きいといった一貫した類似性は確認できなかった。これは出力毎において生成に重要であるトークンは変化することを意味しており、Prefill 時において直前の Attention スコアを利用してキャッシュするトークンを決定する手法 [16] では回答に必要なトークン全てを選択できないことを示唆していると考えられる。

4　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 5 に示した。

4.1　Share-Attention

図 2 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であると仮定し、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。

どのヘッドを共有するかを決定するため、事前にオフラインで共有できるヘッドを決定する。具体的な流れは以下のようである。

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 9 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。

Share-Attention を利用する LLM は、以上の流れで作成した json を基に推論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。

4.2　Select-Attention

前章にて、ヘッド平均 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、重要トークンが概ね決定される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 [16] ではトークンを選択するために 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、k は固定された値であるため、回答に必要なトークンが多いタスクを解く場合や長文が与えられた場合は重要トークンの取りこぼし

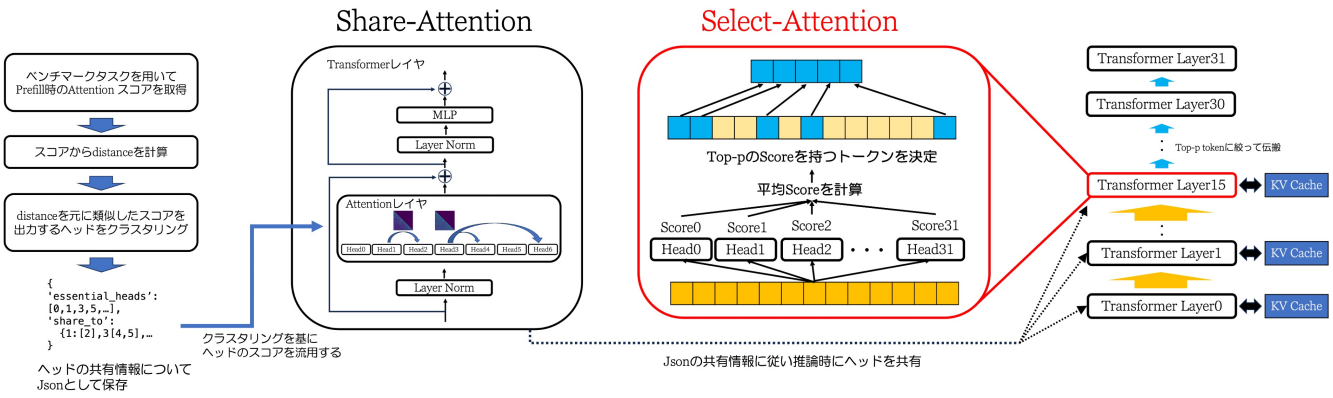

図 5　提案手法 (Share-Attention と Select-Attention) の動作概要

が発生する可能性がある。そのため本研究ではトークン選択手法 Top-p 手法を採用した。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。Top-p 手法を利用することで、タスクに応じて動的に選択するトークン数を変更することができると考えられる。

また事前実験より、図 4 で示したように Prefill 時に重要トークンを決定することは難しいことが確認できた。そこで本手法では各生成時毎にフィルターレイヤにて重要トークンを決定して後半レイヤに伝搬させることを考える。

後半レイヤに伝搬される重要トークンは時刻により変化するため、後半レイヤの Transformer ブロックからの出力は毎時刻変化する。KV キャッシュは以前の出力が変化しないことを前提としているため、後半レイヤでは利用することが難しい。そこで本手法では前半レイヤのみ KV キャッシュを持ち、後半レイヤはキャシュを持たない構成をとる。これはキャッシュの大きさが半分となるため VRAM 消費量が小さくなる利点がある。また Prefill 段階で重要トークンを決定して保存する Gemfilter [16] や H2O [21], SnapKV [12] とは異なりトークン生成毎に重要トークンを決定するため、Prefill 時でのトークンの取りこぼしが発生せず回答の精度を維持できると考えられる。

一方で、p の値を大きくすると後半レイヤに伝搬されるトークン数が増加するため回答速度の低下が懸念される。これは後半レイヤは KV キャッシュを持たないため、伝搬されたトークンに対して毎回 Key, Value の作成および Attention 処理を行う必要があるためである。この懸念については次章において p の値を変化させて比較実験を行った。

5　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。

[*2] https://ai.meta.com/blog/meta-Llama-3-1/

ヘッドのクラスタリングで用いる th の値を大きくすれば共有できるヘッドの数は多くなるが、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 ベンチマーク [8] [7] 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。その結果を図 6 に示した。

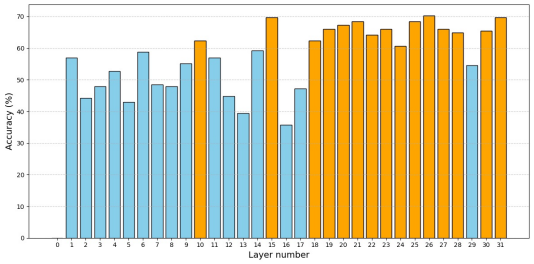

図 6　th=0.9 とするレイヤを変えてベンチマークを行ったときの回答精度。基準モデルでは回答精度は 69% である。オレンジ色のグラフは精度が 60% を超えていることを表す。

結果より、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 から 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。

図 6 と上記考察を踏まえ、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

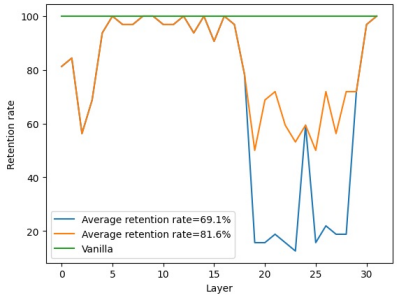

図 7　平均ヘッド維持率が異なる場合の各レイヤにおけるヘッド維持率。前半と中〜後半レイヤについてヘッド維持率が小さい (共有しているヘッド数が大きい) ことがわかる。

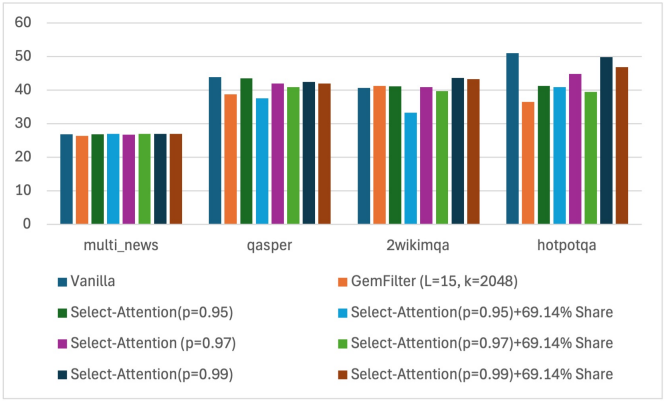

図 8　Vanilla(基準モデル), Gemfilter, Select-Attention 及び Share-Attention を併用した場合の Longbench のスコア比較

表 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

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 [16] での結果も記載した。

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 8 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、Share-Attention を用いる際に CUDA に最適化され長文入力に対応した Attention 計算が可能な Flash_Attention [5] に Share-Attention 機能を追加できておらず、現状はヘッド毎に Attention 計算を行う必要があり Share-Attention 機能を十分に活かす実装になっていないため、計測を行っていない。

Share-Attention 手法により Attention 処理の計算量は削減されるため 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。

最後に、Select-Attention のみを用いて、p を 0.99 から 0.91 まで段階的に現象させて検証を行った。また Prefill 時に Select 利用の有無にてそれぞれスコアを比較した。ここで両手法とも生成時には Select-Attention を利用している。結果を表 3 に示した。

結果より、Prefill・生成時どちらでも Select を用いた場合だと p が小さい場合に QA タスク (2wikiqa, hotpotqa, multifieldqa_en, triviaqa) においてスコアが低下したのに対し、Prefill 時に Select を使わない場合だと p を小さくしてもスコアがほとんど低下しないことが確認できた。これは QA タスクは回答トークン数が少なく、Prefill 時に出力したトークンが生成時のヒントとなり適切な回答が可能となるためと考えられる。

また初期状態での Prefill・生成速度を 1 とした場合の、各 p における Select-Attention の速度比較を行った結果を図 9 に示した。

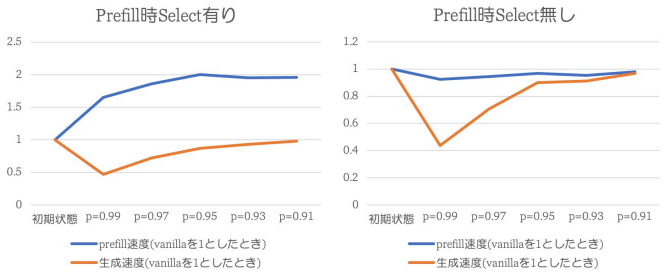

図 9　Select-Attention での p/prefill 時の Select 有無を変更した場合の Prefill・生成速度の変化

図より、Prefill 時にも Select-Attention を利用した場合は、初期状態と比べて約 1.6 から 2 倍の Prefill 時間の向上が確認できた。これは後半レイヤにおいて処理されるトークン数が入力トークンと比較して数%程度になり、計算時間が短縮されたためと考えられる。

同様の理由により生成時においても p が小さい場合は計算量が初期状態に近づくため、Prefill 時の Select-Attention 利用の有無にかかわらず、p が小さい場合は初期状態と比較して生成速度の

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%

表 3　Select-Attention のみを用いた Longbench の各種スコア。p の値と Prefill 時の Select-Attention の利用の有無を変更してスコアを比較している。

	multi_news		qasper		2wikimqa		hotpotqa		multifieldqa_en		triviaqa	
Method \Prefill 時の Select の有無	有	無	有	無	有	無	有	無	有	無	有	無
初期状態	26.79		43.88		41.29		51.16		50.83		88.89	
p=0.99	26.9	26.77	42.48	42.97	43.65	42.8	49.78	50.62	49.73	49.61	86.43	86.67
p=0.97	26.73	26.79	41.94	43.52	40.9	41.25	44.88	50.87	48.26	46.28	78.82	88.06
p=0.95	26.81	26.86	43.51	43.88	41.18	39.52	41.27	50.76	46.24	50.83	72.38	89.34
p=0.93	26.65	26.54	40.77	44.06	34.83	40.18	41.03	49.75	45.09	50.87	65.18	86.65
p=0.91	26.59	26.82	39.86	40.79	34.2	41.59	37.53	51.8	40.42	50.47	51.17	86.37

低下はほとんど確認されなかった。

以上の結果より、Prefill 処理では p を大きく、生成時には p を小さくすることで Prefill 速度の高速化及び生成速度の維持が期待できると考えられる。

6　終わりに

本稿では、Transformer-デコーダを利用した大規模言語モデルの動作高速化およびメモリ使用率の効率化に向けた手法として、同一レイヤ内における Attention スコアの類似性に着目して Share-Attention を, レイヤ間における Attention スコアの類似性に着目して Select-Attention を提案し検証を行った。

結果として、Share-Attention では distance を用いた類似度計算によってクラスタリングを行いヘッドを共有することにより、全体において 2〜3 割程度、一部の層では 8 割程度 Attention 処理を減らした場合でも LLM が適切に動作することが確認できた。

また Select-Attention では、Llama3, 3.1 (8B, Instruct モデル) の第 15 層を Select-Attention レイヤとして用いることで後半レイヤの KV キャッシュを不必要とし全体のキャッシュメモリ量を半減させながら、後半レイヤに伝搬するトークン数を 9 割程度削減しても要約タスクにおいては回答精度の減衰はほとんど無く、QA タスクにおいても 1,2 割程度の減衰であることが確認できた。また生成時の速度低下も限定的であることが確認できた。

さらに Prefill 時には通常処理を、生成時にのみ Select-Attention を用いることにより、QA タスクにおいても精度が低下しないことが確認できた。

この応用として、Prefill 時に p を値を大きく、生成時に p の値を小さくすることにより、KV キャシュ消費量半減かつ Preill 速度を 1.6 倍程度に高速化しながら、生成速度・精度を維持できることが確認できた。

今後の展望として、CUDA での並列処理に特化した手法である Flash_Attention に Share-Attention に対応した機能を追加することでさらなる効率化および長文タスクでのベンチマークが可能になると思われる。

また Select-Attention において Prefill 時・生成時だけでなく、入力長やタスク内容によって p の値を動的に変化させることで、より柔軟に Prefill・生成速度を最適化できると考えられる。

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] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, Singapore, December 2023. Association for Computational Linguistics.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[3] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, August 2024.

Association for Computational Linguistics.
[4] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.
[5] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.
[6] 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. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21158–21166, Miami, Florida, USA, November 2024. Association for Computational Linguistics.
[7] Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
[8] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.
[10] Huiqiang Jiang, Yucheng LI, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 52481–52515. Curran Associates, Inc., 2024.
[11] Jingyao Li, Han Shi, Sitong Wu, Chuanyang Zheng, Zhenguo Li, Xin Jiang, Hong Xu, and Jiaya Jia. QuickLLaMA: Query-aware inference acceleration for large language models. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert, editors, *Proceedings of the 31st International Conference on Computational Linguistics*, pages 508–528, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics.
[12] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM Knows What You are Looking for Before Generation, June 2024. arXiv:2404.14469.
[13] Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. RetrievalAttention: Accelerating Long-Context LLM Inference via Vector Retrieval, September 2024. arXiv:2409.10516.
[14] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: a tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.
[15] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training.
[16] 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.
[17] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.
[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
[19] Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. InfLLM: Training-Free Long-Context Extrapolation for LLMs with an Efficient Context Memory, May 2024. arXiv:2402.04617.
[20] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks, April 2024. arXiv:2309.17453.
[21] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2o: heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.