

# AI推論処理のための動的近似計算手法の検討

佐藤 日向<sup>1,a)</sup> 杉田 脩<sup>1</sup> 入江 英嗣<sup>1</sup>

**概要:** 大規模深層学習モデルの普及に伴い、推論処理における計算資源および電力消費の増大が課題となっている。本研究では、実行時の状況に応じて近似度を切り替える動的近似計算手法を、深層学習推論へ適用することを検討する。提案手法は Stochastic Iterative Approximation (SIA) に基づき、GPU 上で実行される行列積カーネルに適用される。行列積演算をループとして捉え、ブロック単位で通常ルーチンと近似ルーチンを確率的に切り替えることで、GPU の並列性を維持したまま近似度を動的に制御する。近似ルーチンでは、Attention 機構における QKV 演算を恒等写像として近似する。評価では、シミュレーションにより QKV 計算カーネルの命令数およびサイクル数を測定し、実機評価により推論時間と推論精度を測定した。その結果、高近似領域では推論時間短縮と精度低下のトレードオフが明確に現れた。一方、低近似領域では、全層近似においても比較的小さな精度変化で推論時間を最大 5.7%程度短縮した。以上より、提案手法は GPU 上の QKV 計算に対して動的近似制御を導入し、精度と性能のトレードオフを段階的に制御できることを示した。

## Study of Dynamic Approximate Computing Methods for AI Inference

HINATA SATO<sup>1,a)</sup> SHU SUGITA<sup>1</sup> HIDETSUGU IRIE<sup>1</sup>

### 1. はじめに

近年、言語・視覚・音声を含む大規模な生成 AI / 深層学習モデルが幅広い応用で利用されるとともに、モデル規模も拡大し続けている。しかし、学習・推論の処理は計算資源・電力を大量に消費するため、いかに高速かつ省電力に行うかが課題となっている。特に推論は、サービス提供期間を通じて継続的に実行されるため、リクエスト数の増加に伴って消費電力が累積しやすい。このため、推論時のレイテンシと電力消費は品質と運用コストに直結するため、推論処理の高速化・省電力化が重要な課題となっている [1]。

この課題に対処するアプローチの一つに Approximate Computing がある。Approximate Computing とは、計算精度を落として、処理の効率化や低消費電力化を図る手法である。深層学習をはじめとする多くの応用タスクは、厳密な数値精度を必ずしも必要としない。特に深層学習や大

規模言語モデルの推論においては、すべての計算が高精度である必要はなく、タスクや層ごとに精度要求が異なることが知られている [2], [3]。そのため、近似演算の一部を近似的に処理する仕組みを導入することで、処理速度や消費電力の改善が期待できる。量子化やブルーニングは、推論時の計算量・エネルギー消費の削減に有効なモデル最適化手法として広く検討されており、AI 推論の持続可能性と実用性を高める有望なアプローチと位置付けられている [4]。

近似計算手法においては、一般的に近似度は実行前に適切な度合いに定められるが、適切な計算精度は入力データや出力の使われ方によって大きく変化する。この特徴に着目して、我々は先行研究において動的な近似計算手法を提案している [5]。ここでいう「動的」とは、静的に近似の方法や程度を固定するのではなく、実行時の入力や処理状況に応じて近似の度合いを柔軟に切り替えることを意味する。その利点は、タスクの要求精度やシステムのリソース状況に合わせて適切な精度と効率トレードオフを実現できる点にある。動的近似計算の一つとして、ループ構造に対して適用する Stochastic Iterative Approximation (SIA)

<sup>1</sup> 東京大学

The University of Tokyo

<sup>a)</sup> sato-hinata931@g.ecc.u-tokyo.ac.jp

が提案されている [6]. SIA は、ユーザが制御する近似積極度に応じた確率で、通常ルーチンと近似ルーチンをイテレーションごとに切り替えて実行する. これによって、汎用的かつ動的な近似計算を実現することができる.

本研究では、GPU 上で推論演算を実行するカーネルを対象とし、Transformer モデル [7] の推論処理に対する動的近似計算の適用手法を提案し、その効果を明らかにする. Transformer の推論は行列積や要素ごとの演算など、高いデータ並列性を有する計算が支配的であり、これらは GPU の多数の演算器による並列処理と親和性が高い. 一方で、推論時に近似積極度を動的に調整する場合、分岐やスケジューリングの偏りによって GPU の並列性（スループット）を損なう可能性がある. そこで本研究では、GPU の並列実行形態を維持したまま近似度を制御できる方式として、ブロック単位で確率的に演算精度を可変化する動的近似計算を設計する.

具体的には、GPU 上で動く行列積演算カーネルを一つのループとして考え、ブロック（複数ワープ）ごとに通常ルーチンか近似ルーチンかを選択するように実装した. この設計により、ブロック内同期を伴う実装と整合した形で近似を切り替えられるため、分岐発散による効率低下を抑えつつ GPU 並列性を保ちやすい. 近似ルーチンでは、入力から *Query*, *Key*, *Value* を生成するための線形変換を、恒等写像として近似する処理を行う.

評価は、シミュレーション評価と実機評価の双方で行った. シミュレーションでは QKV 計算カーネルを対象に命令数・サイクル数・占有率を測定し、近似制御が GPU 内部指標に与える影響を分析した. 実機評価では推論処理全体を対象に推論時間と推論精度を測定し、近似適用層数および近似粒度の違いに対する性能・精度トレードオフを評価した. その結果、高近似領域では推論時間短縮と精度低下のトレードオフが明確に現れた一方、低近似領域では比較的小さな精度変化で一定の推論時間短縮が得られた. 以上より、提案手法は GPU 上の QKV 計算に対して動的近似制御を導入し、深層学習推論における精度と性能のトレードオフを段階的に調整できることを示した.

## 2. 関連研究

### 2.1 Transformer モデルの概要

Transformer [7] は、Self-Attention を中核とし、系列データの依存関係を高い並列性の下で扱う深層学習アーキテクチャである. Self-Attention では、入力から生成した *Query* (*Q*) と *Key* (*K*) の類似度に基づいて重みを計算し、その重みで *Value* (*V*) を加重和することで文脈を反映した表現を得る. 実際には、*Q*, *K*, *V* は入力表現 *X* に対する線形変換として生成され、この QKV 射影は Self-Attention の前段で実行される主要な計算要素である. さらに Transformer は、入力系列を変換する Encoder と出力系列を生成する

図 1 ソフトウェアの SIA 実装

```
1 for( int i = 0; i < N; i++ ) {  
2     if( approx_level < rand() % 16 ){  
3         // regular routine  
4     } else {  
5         // approximate routine  
6     }  
7 }
```

Decoder からなる構成を持つ. また、複数の Attention を並列に実行して異なる表現空間を扱う Multi-Head Attention、層をまたいで情報を直接伝える Residual Connection、および各層の出力分布を安定化する Layer Normalization を組み合わせることで、系列長に依存しない効率的な並列計算を可能にしている.

### 2.2 深層学習における近似計算

深層学習推論の近似計算は、大きく「演算精度を下げる手法」と「計算そのものを省略する手法」に分けられる. 前者には量子化 [8], [9] があり、後者には層・チャンネル・経路を入力条件に応じて省く早期終了やスキップ実行がある [10], [11], [12], [13]. 本研究では SIA による制御流分岐の適用しやすさにより、後者のアプローチから、特に Transformer の Attention 前段で行われる QKV 生成計算の省略可能性に注目する.

前節で述べた QKV 射影は行列積演算として実装されるため、計算量の主要因の一つとなる. この冗長性に着目した先行研究として、Karbevski らは、理論解析（一定の仮定の下）により *Query* 側重みの冗長性を示し、実験でも  $W_Q$  を恒等写像に置換した設定でベースライン同等の性能を報告している [14]. また、Kowsher らは、*Q*, *K*, *V* で単一の重み行列を共有する Self-Attention を提案し、Self-Attention ブロックのパラメータを大幅に削減できることを示している [15].

### 2.3 SIA（動的近似計算手法）

SIA [6] は、近似の積極度を動的かつ迅速に制御する手法である. プログラム実行中にユーザ側からの入力や状態に応じて近似の積極度を調整することができる. 図 1 にソフトウェアにおける SIA 実装の疑似コードを示す. ユーザは通常ルーチンと近似ルーチンを記述する. 近似の積極度 *approx\_level* が乱数の閾値以上である場合、近似ルーチンが実行される. 図 1 のようにソフトウェアのみで実装すると、乱数生成や近似積極度の変数管理のためのオーバーヘッドが生じる. これに対し、SIA の CPU 環境向け実装では、専用命令や分岐制御機構を導入するハードウェア実装により、このオーバーヘッドを低減することができる.

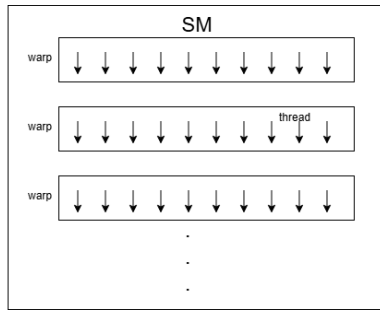


図 2 SM と warp の関係

## 2.4 GPU

### 2.4.1 GPU アーキテクチャ

GPU (Graphics Processing Unit) は、同種演算を多数データへ同時適用する並列処理に適したプロセッサであり、行列積や畳み込みが支配的な AI 推論と親和性が高い。NVIDIA 製 GPU 内部の演算資源は SM (Streaming Multiprocessor) 単位に分割され、各 SM が多数スレッドを保持しながら命令を実行する。また、各 SM は独立した実行資源として動作し、割り当てられたブロック群を他の SM とは独立に進行させる。ここでブロックは、複数のスレッドからなる CUDA の実行単位であり、内部では複数の warp に分かれてスケジューラされる。同一ブロック内のスレッドは共有メモリを利用でき、`__syncthreads()` による同期も可能である。CUDA では、プログラムはスレッド単位で記述されるが、実際の命令実行およびスケジューリングの基本単位は 32 スレッドからなる warp である。図 2 は、SM が複数の warp を保持し、各 warp が 32 スレッドから構成されるという実行単位の関係的模式的に示したものである。図中の下向き矢印は、各 warp を構成するスレッドを表している。

性能を左右する要因は、実行形態とメモリ階層である。実行は 32 スレッドからなる warp を基本単位とする SIMT 方式であり、同一 warp 内で分岐経路が分かると分岐発散が生じて効率が低下する [16]。メモリはレジスタ・共有メモリ・キャッシュ・グローバルメモリからなる階層構造を持ち、特に高レイテンシなグローバルメモリアクセスを抑え、データ再利用性を高めることが重要である [17]。

### 2.4.2 CUDA の実行モデル

本研究の実装に使用する CUDA について説明する。CUDA では、CPU (ホスト) が GPU (デバイス) 上のカーネルを起動して計算を行う。一般的な処理は、`cudaMalloc` によるメモリ確保、`cudaMemcpy` によるデータ転送、カーネル実行、結果の転送で構成される [17]。

カーネルはスレッド集合として起動され、スレッドはブロックへ、ブロックはグリッドへ配置される。各スレッドは `threadIdx` と `blockIdx` などから一意なインデックスを計算して担当要素を処理する。1 次元グリッドでは、

$$\text{index} = \text{blockDim.x} \cdot \text{blockIdx.x} + \text{threadIdx.x} \quad (1)$$

である。たとえばベクトル加算

$$C[i] = A[i] + B[i] \quad (2)$$

では、各スレッドが  $i = \text{index}$  を担当して  $C[\text{index}]$  を計算することで、要素単位の並列化を実現できる。また、ブロック内では共有メモリと `__syncthreads()` による同期が利用できる一方、ブロック間は単一カーネル内で同期できないため、独立実行を前提に設計される [16]。

## 3. 提案手法

### 3.1 概要

本研究では、Transformer の Attention ブロックにおける QKV 生成計算を対象として、SIA に基づく動的近似制御を GPU カーネルへ適用する。ただし、SIA は CPU 環境における繰り返し構造の切替えを前提とした手法であり、GPU へ単純に適用できるわけではない。GPU では、通常ルーチンと近似ルーチンを細粒度に切り替えると、分岐発散や同期との不整合によって並列実行効率が低下しやすいためである。そこで本研究では、QKV 生成の線形写像を近似対象とし、実行時に通常ルーチンと近似ルーチンを確率的に切り替える構成を採る。その際、切替えの制御度をブロック単位とすることで、同期単位と制御単位を整合させ、GPU の並列性を大きく損なわずに近似積極度を反映できるようにする。近似ルーチンでは、Q, K, V を生成する線形変換を恒等写像として近似する。

### 3.2 近似ルーチン

GPU カーネル関数の内部プログラムに対して SIA を適用させた。SIA は本来 CPU 実装を主対象として提案された手法であるが、本研究ではこれを GPU 上で動作するカーネル関数へ適用する。図 3 に SIA 適用前の QKV 計算カーネルの疑似コードを示す。SIA 適用前では、カーネル内で学習済み重みを用いた通常の行列積演算のみを実行する。これに対し、図 4 に示す提案手法では、ブロック単位で乱数と近似積極度を比較し、通常ルーチンと近似ルーチンを切り替える処理を追加する。通常ルーチンでの行列積の演算には  $16 \times 16$  のタイル化を施した。タイル化とは行列を小さなブロックに分割して計算する手法のことで、GPU ではタイルを共有メモリに載せて同じデータをブロック内のスレッドで再利用することで、グローバルメモリアクセス回数を減らし計算効率を上げることができる。このタイルサイズは、使用したベンチマークおよびモデル設定に対して共有メモリ利用と並列実行性の両立を図りやすいことから採用したものである。ただし、適切なタイルサイズは実装方針、対象 GPU の資源制約、行列サイズ、および使用するモデル構成に依存して変化し得る。

`approx_calculate_QKV`関数はGPU上で実行されるカーネル関数である。通常では、以下の式のように学習した重み  $W$  とバイアス  $b$  から  $Q, K, V$  は求められる。

$$Q = XW_Q + b_Q, K = XW_K + b_K, V = XW_V + b_V \quad (3)$$

この近似カーネル関数では、近似の切り替えをブロック単位で行う。複数ブロックを1つの制御単位として扱い、その単位ごとに通常計算と近似計算を切り替える。近似処理は、2.2 節の冗長性議論を踏まえつつ、 $Q, K, V$  すべてを入力の変等写像で置換する手法を取った。このとき、近似ルーチンは式 (4) のように表される。

$$Q = X, K = X, V = X \quad (4)$$

実行時には、ブロック単位（または2ブロック単位）で制御IDを割り当て、そのIDごとに乱数  $r \in \{0, \dots, 15\}$  を生成する。近似積極度を  $ap \in \{0, \dots, 15\}$  とすると、 $ap < r$  なら通常ルーチン、それ以外なら近似ルーチンを実行する。このため、 $ap$  が大きいほど近似ルーチンが選択される確率が高くなる。図4の `block_group_id` はこの制御単位を表し、`Rand4bitPerGroup` は各制御単位に対して独立に乱数を与える。

例えば、 $ap = 0$  では近似ルーチンはほとんど選択されず、 $ap = 15$  ではほぼすべての制御単位で近似ルーチンが実行される。本研究では、この強い近似設定を採ることで、近似適用率の増加に対する性能向上と精度低下の関係を明確に観測することを狙う。

### 3.3 近似粒度

本研究では、 $Q, K, V$  を求める演算に対して、1ブロックあたり256スレッドのカーネルを用いた。このとき1ブロックは32スレッドからなる8個のwarpに相当し、SMの同時常駐可能スレッド数を1024とすれば、理想的には1SMあたり4ブロックを同時常駐できる。

近似粒度の設計では、タイル化行列積における `_syncthreads()` によるブロック内同期との整合を重視した。制御粒度をwarp単位まで細かくすると、近似ルーチンを実行するwarpは計算量が減っても、同一ブロック内の通常ルーチンを実行するwarpが同期点に到達するまで待機する必要がある。そのため、近似warpの演算削減がブロック全体の効率化に結び付きにくい。これに対し、ブロック単位で切り替えることで、同期単位と制御単位を一致させ、GPU並列性を大きく損なわずに近似制御を導入できる。

本研究ではこの得失を踏まえ、制御単位に含めるブロック数を1(1ブロック単位)と2(2ブロック単位)に設定した2条件を比較した。これは、実装可能な最小単位である1ブロック単位制御と、その制御単位を2ブロックへ拡張した場合を比較することで、近似制御粒度を粗くしたとき

図3 SIA適用前の行列積演算処理の疑似コード

```
1 extern "C" __global__ void calculate_QKV(input,
2     output) {
3     int bid = blockIdx.x + gridDim.x * blockIdx.y;
4     int tid = threadIdx.x;
5     int idx = bid * blockDim.x + tid;
6     if (idx < total_threads) {
7         // regular routine
8     }
9 }
```

図4 SIA適用後の行列積演算処理の疑似コード

```
1 extern "C" __global__ void approx_calculate_QKV (
2     input, output) {
3     // int block_group_blocks = 1 or 2
4     int bid = blockIdx.x + gridDim.x * blockIdx.y;
5     int block_group_id = bid / block_group_blocks;
6     rnd4 = Rand4bitPerGroup(block_group_id)
7     if (approx_level < rnd4) {
8         // regular routine
9     } else {
10        // approx routine
11        output = input;
12    }
13 }
```

の影響を確認するためである。ここで、占有率はSM上に同時常駐して実行可能なスレッド群の充足度を表す指標であり、GPU資源利用の程度を評価するために用いる。これにより、近似制御粒度の違いが命令数、サイクル数、占有率、および推論精度に与える影響を評価する。

## 4. 評価

### 4.1 概要

本章では、提案手法を性能と精度の両面から評価する。評価はシミュレーション評価と実機評価の2種類で行う。シミュレーションでは、命令数、サイクル数、GPUの占有率を測定する。実機評価では、実行時間と推論精度を測定する。近似積極度を0から15まで、プログラム実行ごとに変化させ、演算削減における性能向上と精度低下のトレードオフを明らかにする。

評価実装の範囲としては、シミュレーション評価ではQKV計算カーネルのみをC++で記述して詳細シミュレーションし、性能指標を取得する。一方、実機評価では推論処理全体をC++で実装し、近似適用層数の異なる複数条件を比較する。なお、本研究の評価では、プログラム実行中に近似積極度を変えるのではなく、各近似積極度ごとに独

表 1 基準となる GPU 設定

Streaming Multiprocessor	30 SMs, 1350 MHz
Warp Width	32
Warps/SM	32
Warp Scheduler/SM	4

立に実行し、その結果を比較することで近似度選択による特性の違いを評価した。

ここで、動的近似計算に対応するアーキテクチャでは、乱数に基づく確率的制御が専用機構として高速に実行されることも想定される。しかし、本研究の評価ではそのようなハードウェア支援は仮定せず、乱数生成と分岐をソフトウェア実装した上で測定を行った。したがって、シミュレーション評価で得られる命令数およびサイクル数には、近似ルーチン自体の効果に加えて、乱数生成および分岐制御のオーバーヘッドも含まれている。

#### 4.2 評価環境

性能評価には GPGPU-Sim 4.0.0 [18] を用いた。GPU は RTX 2060 をモデルとし、基本的な設定は表 1 に示す。GPGPU-Sim は、CUDA プログラムを対象として GPU 上の実行をサイクルレベルでモデル化するシミュレータであり、実機を用いずに命令数や実行サイクル数などの指標を取得することが可能である。本研究では、提案手法が対象とする「近似適用カーネル」に対して、近似積極度の変化が命令数・サイクル数に与える影響を観測する目的で GPGPU-Sim を使用した。

シミュレーション評価では、推論処理全体を対象とするのではなく、近似を適用した  $Q, K, V$  計算のための行列積カーネル部分のみを C++ で記述して GPGPU-Sim 上で実行した。

実機評価は Google Colaboratory 上で行い、GPU として NVIDIA Tesla T4 を使用した。実機評価では、推論処理のすべての層を C++ 実装で GPU 実行し、QKV 近似を導入する層数が異なる 4 条件、すなわち (i) 第 1 層のみ、(ii) 第 1・2 層、(iii) 第 1・2・3 層、(iv) 全層を比較した。実機評価における推論時間は、CPU 側オーバーヘッドの影響を避けるため、CUDA Event API (`cudaEventRecord`, `cudaEventSynchronize`) を用いて計測した。開始イベントは最初の Embedding 処理直前、終了イベントは最終の分類層の実行完了直後に記録し、その経過時間をモデル全体の推論実行時間とした。

#### 4.3 データセット

学習と評価には Hugging Face の `dair-ai/emotion` データセット [19] を用いた。本データセットは学習データ 16000 個、検証データ 2000 個、テストデータ 2000 個の計 20000 個から構成される。

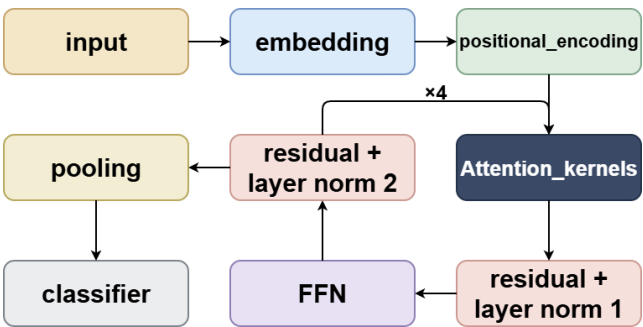


図 5 使用モデルの推論処理フロー

表 2 使用モデルの設定

項目	値
学習時バッチサイズ	32
推論時バッチサイズ	8
学習率	$1 \times 10^{-4}$
エポック数	10
最大トークン長	128
埋め込み次元 $d_{\text{model}}$	256
ヘッド数 $h$	8
エンコーダブロックの層数 $N$	4
FFN 中間次元 $d_{\text{ff}}$	1024
出力クラス数	6
最適化手法	AdamW

シミュレーション評価では、各近似積極度・各近似粒度の組合せごとに 10 回のシミュレーションを実行した。推論時バッチサイズは 8 であるため、各条件で合計 80 データを処理した。

実機評価では、テストデータ 2000 個を用いて評価した。

#### 4.4 モデル

本研究では、テキスト分類タスクに対して、Transformer の Encoder のみからなる分類モデルを使用した。図 5 に、使用モデルの推論処理フローを示す。入力文は文章長  $seq.len = 128$  でトークン化され、各トークンは次元  $d_{\text{model}} = 256$  の埋め込みベクトルへ変換された後、位置情報 (Positional Encoding) を付与されて Encoder へ入力される。Encoder ブロックは  $N = 4$  層で、各層は QKV 計算、Multi-head Attention、Residual Connection、Layer Normalization、FFN から構成される。最終的に系列全体をプーリングして 1 つのベクトルを得て、線形分類器により 6 クラスへ分類した。

加えて、モデルの学習に使用した主要なハイパーパラメータを表 2 に示す。学習は Python で行い、推論は C++ を用いた。学習させたパラメータを各層と各演算ブロックに分けて Numpy の .npz 形式として保存し、そのファイルを C++ (cuda) で記述されたプログラムで `cnpy` [20] を用いて読み込むことで Python で学習させたモデルを C++ で動かした。

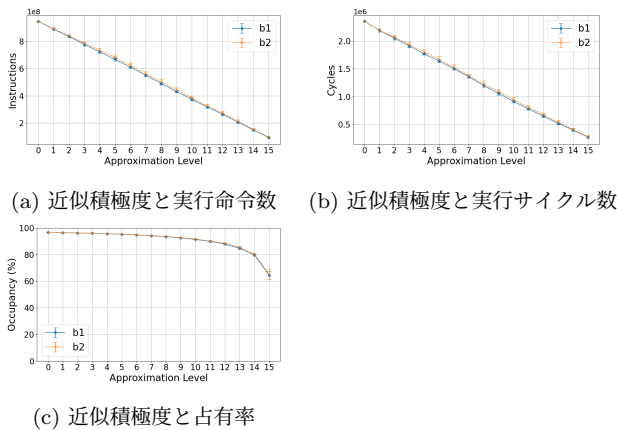


図 6 近似積極度と性能指標の関係 (QKV 計算 1 層)

## 4.5 性能評価

### 4.5.1 近似積極度と実行命令数

まず、近似積極度と実行命令数の関係について述べた。図 6(a) は、Encoder の QKV 計算 1 層に対する実行命令数を近似積極度ごとに示したものである。ここで、凡例の b1 は 1 ブロック単位の制御 (8warp), b2 は 2 ブロック単位の制御 (16warp) を表した。

図 6(a) より、近似積極度が高くなるほど実行命令数は減少した。また、命令数は段階的に低下しており、近似積極度に応じて通常計算と近似計算を切り替える SIA 制御が機能していることが確認できた。さらに、b1 と b2 の差は全体として小さく、制御粒度の違いによる命令数への影響は限定的であった。

### 4.5.2 近似積極度と実行サイクル数

次に、近似積極度と実行サイクル数の関係について述べた。図 6(b) は、Encoder の QKV 計算 1 層に対する実行サイクル数を近似積極度ごとに示したものである。

図 6(b) より、近似積極度が高くなるほど実行サイクル数は減少した。また、サイクル数は段階的に低下しており、近似積極度に応じて通常計算と近似計算を切り替える SIA 制御がサイクル数にも反映されていることが分かった。実行命令数と同様に、b1 と b2 の差は全体としてごく僅か、制御粒度の違いによるサイクル数への影響は小さかった。

### 4.5.3 近似積極度と占有率

次に、近似積極度と占有率の関係について述べた。図 6(c) は、Encoder の QKV 計算 1 層に対する占有率を近似積極度ごとに示したものである。

図 6(c) より、低～中程度の近似積極度では、近似積極度の増加に伴って占有率は緩やかに低下した。一方で、近似積極度が 14 および 15 の高近似領域では、占有率が大きく低下する傾向が確認できた。

## 4.6 精度評価

次に、近似積極度と推論精度の関係について述べた。本節で示す推論精度は、実機評価で取得した結果である。近

表 3 近似積極度 0 と近似積極度 15 の比較  
(精度低下量と推論時間削減率)

近似適用層	b1 精度低下 [p.p.]	b2 精度低下 [p.p.]	b1 時間削減率 [%]	b2 時間削減率 [%]
第 1 層	3.55	4.00	3.59	2.18
第 1・2 層	13.35	16.20	5.83	5.12
第 1・2・3 層	29.15	30.85	9.98	10.45
全層 (0-3)	39.25	38.85	15.79	16.85

表 4 近似積極度 0 と近似積極度 2 の比較 (精度変化量と推論時間削減率)

近似適用層	b1 精度変化量 [p.p.]	b2 精度変化量 [p.p.]	b1 時間削減率 [%]	b2 時間削減率 [%]
第 1 層のみ	0.65	1.10	3.58	0.69
第 1・2 層	-0.30	2.05	2.85	1.78
第 1・2・3 層	0.00	4.30	2.10	3.78
全層 (0-3)	0.70	6.70	4.95	5.67

似積極度ごとの推論精度は、テストデータに対して予測ラベルが正解ラベルと一致した割合として定義した。

図 7 より、全体として b1 と b2 の差は小さく、推論時間に関しては両者に大きな差は見られなかった。一方、推論精度は多くの条件で b1 が b2 をわずかに上回る傾向が確認できた。

また、近似適用層数を第 1 層のみから全層へ増やすほど、近似積極度の増加に対する推論精度の低下幅と推論時間の短縮幅はともに大きくなった。ただし、近似積極度が中程度の領域では、推論時間が単調に減少せず、局所的に増加する点が複数条件で確認された。

表 3 に、近似積極度 0 と近似積極度 15 を比較した場合の精度低下量と推論時間削減率を示した。近似適用層数が増えるほど、推論時間削減率は高くなる一方で、精度低下量も増加した。例えば、全層近似では b1 で 39.25 ポイント (b2 で 38.85 ポイント) の精度低下と引き換えに、推論時間は 15.79% (b2 で 16.85%) 短縮された。一方、第 1 層のみに近似を適用した場合、精度低下は 3.55 ポイント (b2 で 4.00 ポイント) に抑えられたが、推論時間削減率は 3.59% (b2 で 2.18%) にとどまった。

表 4 に、近似積極度 0 と近似積極度 2 を比較した場合の精度変化量と推論時間削減率を示す。全層近似では、近似積極度 2 にすることで推論時間は b1 で 4.95%, b2 で 5.67% 短縮された一方、精度低下はそれぞれ 0.70 ポイント、6.70 ポイントであった。また、第 1 層のみに近似を適用した場合、推論時間削減率は b1 で 3.58%, b2 で 0.69% であり、精度低下はそれぞれ 0.65 ポイント、1.10 ポイントにとどまった。さらに、第 1・2 層および第 1・2・3 層では、b1 において近似積極度 2 の精度が近似積極度 0 と同等である結果が見られた。

## 4.7 考察

性能評価の結果より、近似積極度の増加に伴って実行命令数および実行サイクル数はいずれも段階的に減少し、SIA による確率的切替えが GPU カーネルの計算量削減として有効に機能することを確認した。また、占有率につい



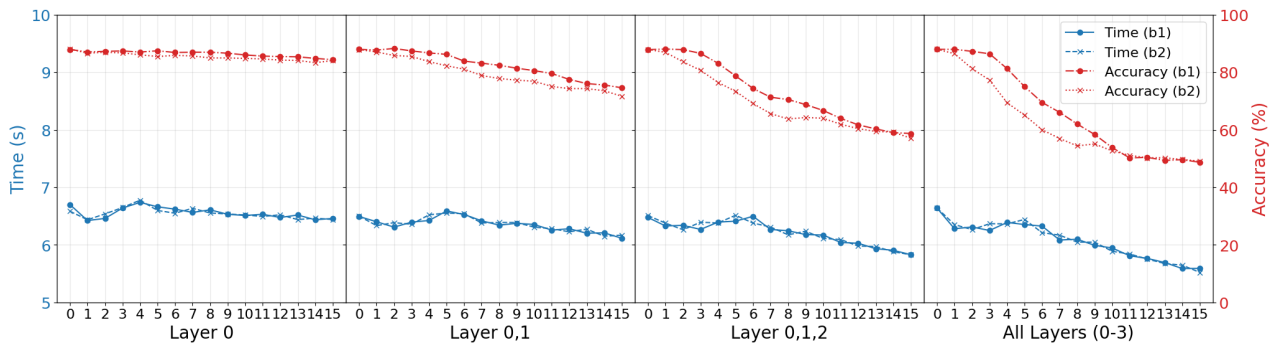


図 7 近似適用層数ごとの実行時間と推論精度の関係

ては、近似積極度を中程度まで高めた範囲では低下は緩やかであり、並列実行性は大きく損なわれない。一方で、高近似領域（近似積極度 14, 15）では占有率の低下が顕著となるため、性能向上効果と実行資源利用率の両面を考慮した近似積極度の設定が必要である。

精度評価では、低近似領域では比較的小さい精度変化で実行時間短縮が得られた。例えば全層近似の近似積極度 0 → 2 では、推論時間は b1 で 4.95%, b2 で 5.67% 短縮された一方、精度低下はそれぞれ 0.70 ポイント、6.70 ポイントであった。これに対し近似積極度 0 → 15 では、推論時間は b1 で 15.79%, b2 で 16.85% 短縮されたが、精度低下は 39.25 ポイント、38.85 ポイントに達した。これらの結果は、低近似領域では精度を大きく損なわずに計算量削減の効果を得られる一方、高近似領域では性能向上と引き換えに精度劣化が急速に顕在化することを示している。また、多くの条件で b1 の方が b2 よりも高い推論精度を示したことから、1 ブロック単位制御の方が近似の影響をより細かく分散でき、2 ブロック単位制御に比べて表現劣化を局所化しやすかったと考える。一方で、中近似領域では推論時間が単調に減少せず、条件によっては逆に増加する場合も見られた。これは、近似適用率が中程度の条件では、通常ルーチンと近似ルーチンの混在によりブロック間の実行時間のばらつきが増大し、乱数生成・分岐制御のオーバーヘッドも相まって、演算削減効果が実行時間短縮に十分結び付かなかったためと考えられる。

以上より、本研究の結果からは、低近似領域では比較的小さい精度変化で時間短縮が得られ、高近似領域では性能向上と精度低下のトレードオフが明確に現れることが示された。一方、中近似領域における挙動については、条件によるばらつきの要因を十分に特定できておらず、現時点で一意的な運用指針を結論づけることは難しい。したがって、今後は低近似領域と高近似領域の使い分けを含む制御方法を検討するとともに、中近似領域の挙動解析および制御法の改良を進める必要がある。

## 5. おわりに

本論文では、AI 推論処理、特に Transformer 推論処理に対する動的近似計算手法を検討した。具体的には、SIA に基づく確率的切替えを GPU カーネルへ導入し、QKV 計算を対象として、近似積極度に応じて通常ルーチンと近似ルーチンをブロック単位で切り替える方式を実装した。さらに、1 ブロック単位および 2 ブロック単位の制御を比較し、近似制御粒度の違いが性能および精度に与える影響を評価した。

評価の結果、近似積極度の増加に伴って命令数およびサイクル数は減少し、実行時間短縮の傾向が確認された。一方で、近似積極度や近似適用層数を大きくするほど推論精度は低下し、特に高近似領域ではその低下が顕著となった。これらの結果から、提案手法は推論性能と推論精度のトレードオフを動的に調整する手段となることを示した。

今後の展望としては、QKV 計算以外の演算ブロックに対しても近似を適用し、各アルゴリズムに応じた詳細実装を行った場合の計算量および推論精度を評価することで、提案方式の効果をより詳細に解析する。また、恒等写像以外の近似手法を導入し、特に行列内部の数値特性に着目した、より自然で情報量が失われにくい近似方式を検討する。加えて、これらの解析結果をフィードバックし、ブロック単位の SIA をハードウェアで支援可能な動的近似 GPU アーキテクチャの開発を進める。これらを通じて、動的近似計算の実用性と適用範囲の拡大を目指す。

## 謝辞

本研究の一部は共同研究（富士通株式会社）「データ転送性能ボトルネックを解消する新アーキテクチャ SPU に向けたソフトウェアならびにその周辺技術の研究」による。

## 参考文献

- [1] Xu, J., Zhou, W., Fu, Z., Zhou, H. and Li, L.: A Survey on Green Deep Learning (2021).
- [2] Wang, K., Liu, Z., Lin, Y., Lin, J. and Han, S.: HAQ: Hardware-Aware Automated Quantization with Mixed

- Precision, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8612–8620 (online), DOI: 10.1109/CVPR.2019.00881 (2019).
- [3] Shen, S., Dong, Z., Ye, J., Ma, L., Yao, Z., Ghohami, A., Mahoney, M. W. and Keutzer, K.: Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT, *arXiv preprint*, (online), available from <https://arxiv.org/abs/1909.05840> (2019).
- [4] Tabbakh, A., Al Amin, L., Islam, M., Mahmud, G. M. I., Chowdhury, I. K. and Mukta, M. S. H.: Towards sustainable AI: a comprehensive framework for Green AI, *Discover Sustainability*, Vol. 5, p. 408 (online), DOI: 10.1007/s43621-024-00641-4 (2024). Article number 408.
- [5] 富田和孝, 中村朋生, 小泉 透, 出川祐也, 入江英嗣, 坂井修一: 近似の積極性を動的制御可能なアーキテクチャのためのコンパイラフレームワーク, *情報処理学会論文誌*, Vol. 63, No. 4, pp. 1019–1028 (オンライン), DOI: 10.20729/00217606 (2022).
- [6] Nakamura, T., Tomida, K., Kouno, S., Irie, H. and Sakai, S.: Stochastic Iterative Approximation: Software/hardware techniques for adjusting aggressiveness of approximation, *Proceedings of the 2021 IEEE 39th International Conference on Computer Design (ICCD)*, IEEE, pp. 74–82 (online), DOI: 10.1109/ICCD53106.2021.00023 (2021).
- [7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I.: Attention Is All You Need, *arXiv preprint*, (online), available from <https://arxiv.org/abs/1706.03762> (2017).
- [8] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H. and Kalenichenko, D.: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2704–2713 (online), DOI: 10.1109/CVPR.2018.00286 (2018).
- [9] Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J. and Han, S.: SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models, *Proceedings of the 40th International Conference on Machine Learning* (Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S. and Scarlett, J., eds.), Proceedings of Machine Learning Research, Vol. 202, PMLR, pp. 38087–38099 (online), available from <https://proceedings.mlr.press/v202/xiao23c.html> (2023).
- [10] Wang, X., Yu, F., Dou, Z.-Y., Darrell, T. and Gonzalez, J. E.: SkipNet: Learning Dynamic Routing in Convolutional Networks, *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 409–424 (online), DOI: 10.1007/978-3-030-01261-8\_25 (2018).
- [11] Kutukcu, B., Baidya, S. and Dey, S.: SLEXNet: Adaptive Inference Using Slimmable Early Exit Neural Networks, *ACM Transactions on Embedded Computing Systems*, Vol. 23, No. 6, p. 104 (online), DOI: 10.1145/3689632 (2024). Article 104, 29 pages.
- [12] Tang, S., Wang, Y., Kong, Z., Zhang, T., Li, Y., Ding, C., Wang, Y., Liang, Y. and Xu, D.: You Need Multiple Exiting: Dynamic Early Exiting for Accelerating Unified Vision Language Model, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10781–10791 (online), DOI: 10.1109/CVPR52729.2023.01038 (2023).
- [13] P, H. R., Srivastava, V., Chaurasia, K., Pacheco, R. G. and Couto, R. S.: Early-Exit Deep Neural Network—A Comprehensive Survey, *ACM Computing Surveys*, Vol. 57, No. 3, pp. 75:1–75:37 (online), DOI: 10.1145/3698767 (2024). Article 75.
- [14] Karbevski, M. and Mijoski, A.: Key and Value Weights Are Probably All You Need: On the Necessity of the Query, Key, Value weight Triplet in Decoder-Only Transformers (2025).
- [15] Kowsher, M., Prottasha, N. J., Yu, C.-N., Garibay, O. and Yousefi, N.: Does Self-Attention Need Separate Weights in Transformers?, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, Albuquerque, New Mexico, Association for Computational Linguistics, pp. 535–543 (online), DOI: 10.18653/v1/2025.naacl-industry.44 (2025).
- [16] Corporation, N.: *CUDA C++ Best Practices Guide* (2025). Release 13.1. Accessed: 2026-01-30.
- [17] Corporation, N.: *CUDA C++ Programming Guide* (2025). Release 13.1. Accessed: 2026-01-30.
- [18] Khairy, M., Shen, Z., Aamodt, T. M. and Rogers, T. G.: Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling, *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, pp. 473–486 (online), DOI: 10.1109/ISCA45697.2020.00047 (2020).
- [19] Saravia, E., Liu, H.-C. T., Huang, Y.-H., Wu, J. and Chen, Y.-S.: CARER: Contextualized Affect Representations for Emotion Recognition, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, Association for Computational Linguistics, pp. 3687–3697 (online), DOI: 10.18653/v1/D18-1404 (2018).
- [20] Rogers, C. and contributors: *cnpy: Read and write NumPy .npy and .npz files in C/C++*, GitHub repository. Accessed: 2026-02-03.