課題研究報告書

Token 合併と後処理量子化に基づく Vision Transformer 推論高速化の実装と検証

FENG MINGYU

主指導教員 井口 寧

北陸先端科学技術大学院大学
先端科学技術研究科
（情報科学）

2026 年 3 月

Abstract

近年，人工知能技術の急速な発展に伴い，深層ニューラルネットワークはコンピュータビジョン，自然言語処理，音声認識などの幅広い分野で実用化が進んでいる．深いネットワーク構造や大規模なパラメータ数により，従来の手動設計による特徴量では捉えられなかった高次かつ抽象的な表現を学習できる点が，高い性能を実現している要因である．一方で，このような高性能モデルは計算量およびメモリ消費が大きく，学習時には高性能・高消費電力な計算資源を必要とするだけでなく，推論時においても高い計算負荷，GPU メモリ使用量の増大，および推論遅延を引き起こす．これらの要因は，エッジデバイスやリアルタイム処理，計算資源が制約された環境への展開を困難にしている．

この課題に対し，近年では深層ニューラルネットワークの実運用性を高めるため，モデル圧縮および推論高速化に関する研究が盛んに行われている．モデルプルーニングでは，重みやニューロンを単位とした細粒度プルーニングから，チャネル，層，ブロック単位で構造を簡略化する構造化プルーニングまで，さまざまな手法が提案されており，パラメータ数削減や実行効率向上が報告されている．また，モデル量子化（Quantization）では，重みや活性値を浮動小数点数から低ビット幅の整数表現（例：INT8）へ変換することで，計算量およびメモリ帯域の削減を実現し，GPU Tensor Core などの低精度演算資源を有効活用できる点が注目されている．

しかしながら，これらの手法には依然としていくつかの課題が残されている．高い精度を維持するため，多くのプルーニング手法や量子化対応学習（QAT: Quantization-Aware Training）はファインチューニング（fine-tuning）を前提としており，追加の学習データ，計算時間，計算資源を必要とする．そのため，学習済みモデルのみが利用可能な場合や，迅速なモデル展開が求められる実運用環境においては，これらの学習依存手法が導入上の負担となる．さらに，アルゴリズム上の計算削減効果が，必ずしも実際のハードウェア推論における高速化に直結しない点も課題である．これは，カーネル実装の差異，メモリアクセスのボトルネック，カーネル融合の有無，および推論フレームワークにおける低精度演算対応状況などに起因する．特に Vision Transformer（ViT）では，Self-Attention の計算量が長いトークン系列の二乗に比例するため，入力トークン数の増加が計算量とメモリ帯域の双方を急激に増大させ，高解像度入力や長い系列処理において顕著な性能ボトルネックとなる．したがって，精度を維持しつつトークン数および低精度計算負荷を削減し，それらを実際の推論エンジンおよび GPU ハードウェア上で有効に機能させることが，重要な研究課題である．

以上の背景を踏まえ，本研究では学習済み Vision Transformer を対象とし，再学習を必要としない推論高速化手法の検討を行う．具体的には，モデルレベルでの計算削減手法として Token Merging を用い，推論過程において類似トークンを動的に統合することで長い系列を削減し，Self-Attention 計算量の低減を図る．さらに，学習後量子化（PTQ: Post-Training Quantization）を適用し，モデルを FP32

から INT8 などの低精度表現へ変換することで，演算量およびメモリ使用量の削減を行う．

そのため本論文では，これらの手法を統合した推論パイプラインを構築し，Vision Transformer の代表的モデルである ViT-B/16 を用いて評価を行った．その結果，分類精度の低下を 1.2% 以内に抑えたまま，FP32 推論と比較して 3.09〜3.55 倍の推論高速化を実現した．また，NVIDIA が提供する Transformer 向け汎用 INT8 量子化手法と比較しても，同等の精度条件下において最大 12.8% 高速な推論性能を達成した．

本研究は，Token Merging と PTQ を統合することで，精度低下を最小限に抑えつつ推論計算量を大幅に削減し，GPU 実機推論における有効性を実証することを目的とする．また，モデル圧縮および低精度推論において生じるボトルネックを整理し，実装レベルでの検討を通じて，Vision Transformer（Vision Encoder）の推論最適化に関する実践的かつ再現性のある知見を示す．

Abstract

In recent years, with the rapid advancement of artificial intelligence technologies, deep neural networks have been increasingly deployed in a wide range of fields, including computer vision, natural language processing, and speech recognition. Owing to their deep network architectures and large numbers of parameters, these models are able to learn high-level and abstract representations that could not be captured by conventional manually designed features, which is a key factor behind their high performance. On the other hand, such high-performance models require large amounts of computation and memory. They not only demand high-performance and high-power-consumption computational resources during training, but also incur substantial computational load, increased GPU memory usage, and inference latency during deployment. These factors make it difficult to apply such models to edge devices, real-time processing, and environments with limited computational resources.

To address these challenges, extensive research has been conducted on model compression and inference acceleration to improve the practical usability of deep neural networks. In model pruning, a variety of methods have been proposed, ranging from fine-grained pruning at the level of individual weights or neurons to structured pruning that simplifies network architectures at the channel, layer, or block level, achieving reductions in parameter count and improvements in execution efficiency. In addition, model quantization converts weights and activations from floating-point representations to low-bit-width integer formats (e.g., INT8), thereby reducing computational cost and memory bandwidth requirements, and enabling effective utilization of low-precision computing resources such as GPU Tensor Cores.

However, several challenges still remain with these approaches. To maintain high accuracy, many pruning methods and quantization-aware training (QAT) techniques assume fine-tuning, which requires additional training data, computational time, and computational resources. As a result, in practical deployment scenarios where only pretrained models are available or rapid model deployment is required, such training-dependent methods impose a significant burden. Furthermore, reductions in computational complexity at the algorithmic level do not necessarily translate directly into actual inference speedups on real hardware. This is due to differences in kernel implementations, memory access bottlenecks, the presence or absence of kernel fusion, and the degree of support for low-precision computation in inference frameworks. In particular, for Vision Transformers (ViTs), the computational cost of self-attention scales quadratically with the length of the token sequence. As a result, an increase in the number of input tokens leads to a rapid growth in both computational cost and memory bandwidth requirements,

becoming a significant performance bottleneck for high-resolution inputs and long-sequence processing. Therefore, reducing the number of tokens and the load of low-precision computation while maintaining accuracy, and ensuring that these reductions function effectively on actual inference engines and GPU hardware, is an important research challenge.

Based on the above background, this study investigates retraining-free inference acceleration methods for pretrained Vision Transformers. Specifically, Token Merging is employed as a model-level computation reduction technique, in which similar tokens are dynamically merged during the inference process to shorten long token sequences and reduce the computational cost of self-attention. In addition, post-training quantization (PTQ) is applied to convert the model from FP32 to low-precision representations such as INT8, thereby reducing computational cost and memory usage.

Accordingly, this thesis constructs an inference pipeline that integrates these techniques and evaluates its performance using ViT-B/16, a representative Vision Transformer model. Experimental results show that, while keeping the classification accuracy degradation within 1.2%, the proposed approach achieves a 3.09–3.55 $\times$ inference speedup compared with FP32 inference. Furthermore, when compared with NVIDIA's general-purpose INT8 quantization method for Transformer models, the proposed approach achieves up to 12.8% faster inference performance under equivalent accuracy conditions.

This study aims to demonstrate the effectiveness of integrating Token Merging and PTQ in significantly reducing inference computational cost while minimizing accuracy degradation through GPU-based inference experiments. In addition, this work organizes the bottlenecks that arise in model compression and low-precision inference, and through implementation-level investigations, presents practical and reproducible insights into inference optimization for Vision Transformers (Vision Encoders).

目 次

図 目 次

表 目 次

第1章　緒言

1.1　研究背景

近年，計算ハードウェアおよびソフトウェア技術の発展に伴い，画像認識モデルの実運用が現実的な課題として注目されている．コンピュータサイエンス分野，とりわけコンピュータビジョン分野において深層ニューラルネットワークを用いた手法が飛躍的な発展を遂げている．畳み込みニューラルネットワーク（Convolutional Neural Network：CNN）は，局所特徴の抽出能力および位置変換に対する頑健性に優れ，長年にわたり画像認識分野の中核的手法として用いられてきた [1, 2, 3]．多様な視覚タスクに対応するため，ResNet や EfficientNet をはじめとするさまざまな CNN アーキテクチャが提案され，高い性能を達成している [4]．代表例として，VGG や Inception 系列も大規模画像認識を牽引した [5, 6]．一方で，CNN は畳み込み演算の局所受容野に基づく構造上，画像全体にわたる長距離依存関係のモデリングには一定の制約があり，複雑な文脈理解を要するタスクにおいては限界が指摘されてきた．この課題に対し，系列変換では RNN に基づく Seq2Seq が提案され，さらに attention により対応付けが改善された [7, 8]． その後，Transformer モデルは自然言語処理分野において Self-Attention 機構を導入することで，系列データ中の長距離依存関係を効果的に捉える手法として注目を集めた [9]．特に 2020 年に提案された Vision Transformer（ViT）は，画像をパッチ列として扱うことで Transformer を視覚タスクに適用し，画像分類をはじめとする複数のベンチマークにおいて CNN を上回る性能を示した [10]．その後，ViT 系モデルは領域分割や物体検出など，幅広いタスクへと応用が拡大している．本論文では，このようなビジョンタスクに適用される Transformer 系モデルを総称して ViT と呼ぶ．

しかしながら，ViT は高い認識性能を実現する一方で，長いトークン系列に依存した Self-Attention 計算に起因する計算量およびメモリ使用量の増大という課題を抱えている．特に推論時においては，計算負荷の増加，GPU メモリ消費の増大，および推論レイテンシの悪化が問題となり，大規模モデルを実運用環境へ展開する上での制約要因となっている．これらの問題はエッジデバイスに限らず，近年主流となっている GPU を用いた大規模推論環境においても，スループットや資源利用効率の観点から重要な課題である．

このような背景のもと，ViT の推論計算量を削減するためのモデル圧縮および推論高速化手法が盛んに研究されている．特に，Self-Attention の計算量がトークン系列長の二乗に比例することから，トークン数そのものを削減するトークンレ

ベルの最適化手法が注目されている．これらには，重要度の低いトークンを削除する Token Pruning や Token Dropping [17] に加え，トークンを削除せずに結合することで系列長を短縮する Token Merging といった手法が含まれる [19].

一方，モデル構造を大きく変更せずに推論計算を高速化する手法として，モデル量子化も重要な研究分野である．モデル量子化は，重みや活性値を浮動小数点表現から低ビット幅の整数表現へ変換することで，計算量およびメモリ帯域を削減し，GPU Tensor Core などの低精度演算ユニットを有効活用できる点に特徴がある [23]．これにより，精度低下を抑えつつ実機推論における高速化が期待されている．

1.2 目的

本研究の目的は，学習済み Vision Transformer（ViT）を対象として，既存の Token 合併および学習後量子化（PTQ: Post-Training Quantization）を組み合わせ，再学習を必要とせずに実機 GPU 上の推論（特に INT8 推論）として成立させるための統合設計法と実装手順を確立し，推論スループットの向上を実現することである．近年，ViT は高精度な視覚表現を実現する一方で，トークン数に比例して計算量およびメモリ負荷が増大するという課題を有しており，高解像度入力や実運用環境への展開において，推論効率の改善が強く求められている．

本研究では，この課題に対して，モデル構造に基づく計算削減手法と数値表現の低精度化による計算効率向上手法を統合することで，実機上で有効に機能する推論高速化を目指す．具体的には，(i) 推論過程において類似トークンを動的に統合し，Self-Attention の計算量を削減する Token Merging と (ii) 学習後量子化を適用し，FP32 演算を INT8 などの低精度演算へ変換する手法を組み合わせる．これにより，精度低下を最小限に抑えつつ，推論計算量およびメモリ使用量の削減を図る．

さらに本研究では，アルゴリズムレベルでの計算削減効果が必ずしも実際の GPU 推論性能向上に直結しない点に着目し，推論エンジンおよび実装上の制約を考慮した評価を行う．具体的には，ONNX および TensorRT を用いたデプロイ手順を含め，低精度演算対応状況やカーネル選択（フォールバック）等が推論性能に与える影響を整理する．以上を通じて，再現性の高い実機評価に基づき，Vision Transformer の推論最適化に関する実践的な設計指針を示すことを目的とする．

1.3 本論文の構成

本論文は全 5 章から構成される． 第 1 章では，研究背景と目的を述べ，本研究の位置づけを明確にする． 第 2 章では，Vision Transformer の基礎的構造および計算特性を整理した上で，トークン削減手法および量子化手法に関する関連研究

を概観し，本研究で扱う課題を整理する． 第 3 章では，Token Merging と学習後量子化を統合した提案手法の全体構成を示し，推論時における設計方針および実装上の工夫について述べる． 第 4 章では，実験環境，評価指標，および GPU 実機推論を含む評価パイプラインを示すとともに，比較条件を定義する．さらに，実機環境における推論速度および精度の評価結果を示し，量子化カーネルの挙動やフォールバックを含めた考察を行う． 第 5 章では，本研究の概要と得られた成果をまとめるとともに，今後の課題および発展可能性について述べる．

第2章　事前知識と関連研究

2.1　Transformerの基礎

Transformer は，系列データの処理を目的として提案された深層学習モデルであり，2017 年に Vaswani らによって自然言語処理分野において発表された [9]．従来の系列モデリングでは，回帰型ニューラルネットワーク（RNN）や畳み込みニューラルネットワーク（CNN）が主に用いられてきたが，Transformer はこれらの構造に依存せず，自己注意機構（Self-Attention）を中核とする点に特徴がある．本研究で対象とする ViT は，この Transformer の Encoder ブロックを画像トークン列に適用した変種であるため，以降では推論高速化の観点で重要となる構成要素（Embedding，Positional Encoding，Self-Attention，FFN，残差・正規化）を整理する．

2.1.1　Transformer の概要と設計動機

背景：RNN/CNN の限界と Self-Attention の利点　RNN は時系列データを逐次的に処理することで依存関係を表現できる一方，長い系列に対しては勾配消失や勾配爆発が生じやすく，また計算の逐次性により並列化が困難であるという問題を抱えている．長短期記憶ネットワーク（LSTM）などの改良手法により一定の改善は得られたものの，計算効率やメモリ消費の観点では依然として制約が残る [11]．一方，CNN は局所受容野に基づく特徴抽出に優れており高い並列性を有するが，系列全体にわたる長距離依存関係を直接的に捉えることは難しい．

これらの課題に対し，Transformer は自己注意機構を用いることで，系列中の任意の二つの位置間に直接的な依存関係を構築できる．自己注意では，入力系列中の各要素が他のすべての要素を参照しながら表現を更新するため，長距離依存を効率的に捉えることが可能である．さらに，各トークンに対する計算は行列演算として並列に実行できるため，計算効率の面でも従来手法に比べて優れている．

全体構造（Encoder/Decoder）　Transformer は，N 層の Encoder と N 層の Decoder から構成され，各層は（i）Multi-Head Attention と（ii）Position-wise Feed-Forward Network（FFN）を基本要素として積層される．各サブレイヤには残差接続と Layer Normalization が適用され，学習の安定性と深い層構造の実現

を支える．なお，本研究で扱う ViT は Decoder を用いず，Encoder 部を画像トークン列に対して適用する点が特徴である．

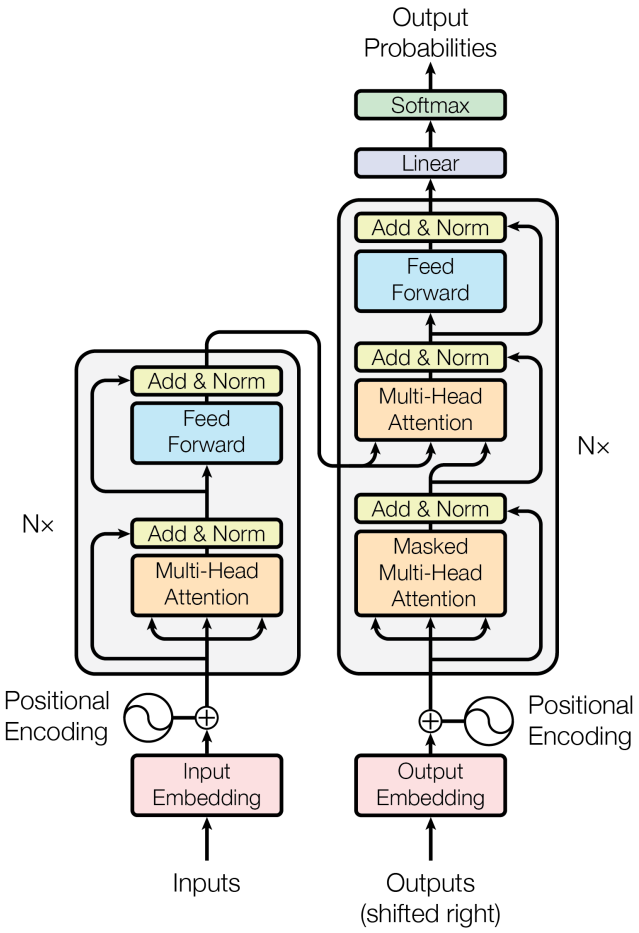

図 2.1: Transformer の全体構造（Encoder/Decoder）[9] より引用

入力埋め込み（Token Embedding）　入力系列 $\{x_1, \ldots, x_L\}$ は埋め込み層により d_{model} 次元へ写像され，

$$\mathbf{e}_i = \text{Embed}(x_i) \in R^{d_{\text{model}}} \tag{2.1}$$

として表現される．以降，系列長を L とし，$\mathbf{E} \in R^{L \times d_{\text{model}}}$ を埋め込み行列とする．

位置埋め込み（Positional Encoding）　Transformer は再帰構造を持たないため，トークンの順序情報を明示的に付与する必要がある．代表的な方法として，固

定のサイン・コサイン位置埋め込みが用いられる．位置 pos と次元 index i に対して次のように定義され，入力は $\mathbf{X} = \mathbf{E} + \mathbf{PE}$ として Attention に入力される：

$$\mathrm{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{\mathrm{model}}}}\right), \tag{2.2}$$

$$\mathrm{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{\mathrm{model}}}}\right). \tag{2.3}$$

Scaled Dot-Product Self-Attention　Self-Attention では，入力 $\mathbf{X}$ から線形変換により Query/Key/Value を生成する：

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V, \tag{2.4}$$

ここで $\mathbf{W}^Q, \mathbf{W}^K \in R^{d_{\mathrm{model}} \times d_k}$，$\mathbf{W}^V \in R^{d_{\mathrm{model}} \times d_v}$ とする．Scaled Dot-Product Attention は次式で与えられる：

$$\mathrm{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}. \tag{2.5}$$

係数 $1/\sqrt{d_k}$ は内積値のスケールを調整し，softmax の入力が過度に大きくなることを防いで数値的安定性を高める．

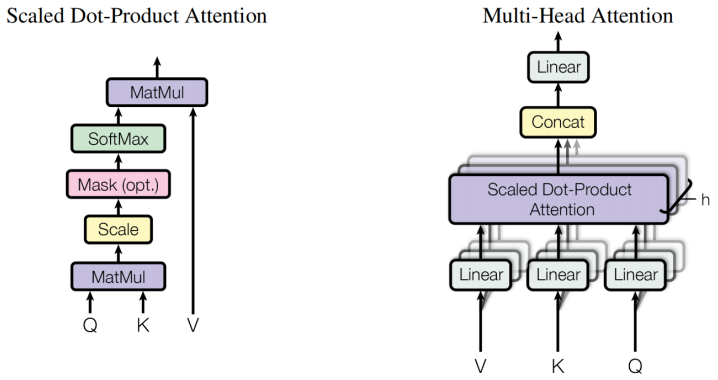

図 2.2: Scaled Dot-Product Attention（左）と Multi-Head Attention（右）[9] より引用

Multi-Head Self-Attention（MHSA）　単一の Attention では表現が制約されるため，Transformer は h 個のヘッドで並列に Attention を計算し，結合する：

$$\mathrm{head}_j = \mathrm{Attn}(\mathbf{X}\mathbf{W}_j^Q,\ \mathbf{X}\mathbf{W}_j^K,\ \mathbf{X}\mathbf{W}_j^V), \tag{2.6}$$

$$\mathrm{MHSA}(\mathbf{X}) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)\mathbf{W}^O, \tag{2.7}$$

ここで $\mathbf{W}^O \in R^{hd_v \times d_{\mathrm{model}}}$ である．各ヘッドが異なる表現部分空間に注意を向けることで，系列中の多様な関係性を同時に捉え，表現力が向上する．

Position-wise Feed-Forward Network（FFN）　Attention の出力に続く Position-wise FFN は，各トークンに独立に適用される二層の全結合ネットワークであり，非線形変換を通じて表現を拡張する役割を担う：

$$\mathrm{FFN}(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2, \tag{2.8}$$

ここで σ は活性化関数（原論文では ReLU，ViT 系では GELU が多い）である．実装構成によっては，推論時の FLOPs の大部分を FFN（MLP）が占める場合もある．

残差接続と Layer Normalization　各サブレイヤは残差接続と正規化により安定化される：

$$\mathbf{y} = \mathrm{LayerNorm}(\mathbf{x} + \mathrm{Sublayer}(\mathbf{x})). \tag{2.9}$$

ViT では Pre-LN 構成が一般的であり，量子化・推論最適化時には LayerNorm や softmax などの数値特性が性能・精度に影響し得る点が重要である．

計算量特性と推論ボトルネック　計算量の観点では，Self-Attention は $\mathbf{Q}\mathbf{K}^\top$ により $L \times L$ の相関行列を扱うため，入力系列長 L に対して計算量およびメモリ使用量が概ね $O(L^2 d)$ で増加する．一方，FFN は概ね $O(Ld^2)$ で増加する．したがってトークン系列長 L の削減と低精度演算の活用は，推論高速化における主要な方向性となる．

表 2.1: Transformer における代表的な計算量

モジュール	計算量	系列長依存
Self-Attention	$O(L^2 d)$	二次
FFN（MLP）	$O(Ld^2)$	一次

2.2 Vision Transformer（ViT）の基礎

Transformer が自然言語処理分野で成功を収めたことを背景に，同様の自己注意機構をコンピュータビジョンへ適用する試みが進み，Dosovitskiy らは画像をトークン列として扱う Vision Transformer（ViT）を提案した [10]．ViT は画像分類において高い性能を示し，以降の多くの視覚モデルにおける基本構成（Vision Encoder）として広く参照されている．本節では，標準 ViT の入力設計と Encoder 構造を整理し，推論高速化の観点から重要となる計算特性を明確にする．

2.2.1 標準 ViT の全体像

標準 ViT は，入力画像を固定サイズのパッチに分割し，各パッチを埋め込み（Patch Embedding）としてトークン列へ変換した上で，Transformer Encoder に入力する構成をとる．最終的に，系列先頭に付与した `[CLS]` トークンの表現を用いて分類を行う．

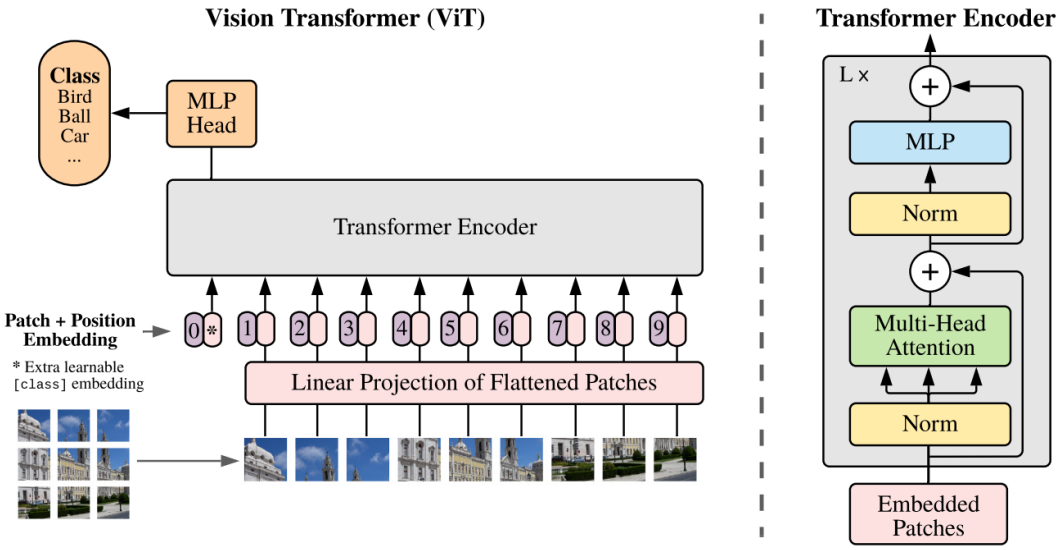

図 2.3: 標準 ViT の構造（Patch Embedding → Transformer Encoder → MLP Head）[10] より引用

2.2.2 ViT の入力設計（Patch / CLS / Positional Embedding）

Patch Embedding：画像をトークン列へ変換　入力画像を $X \in R^{H\times W\times C}$ とし，パッチサイズを $P\times P$ とする．画像を非重複パッチへ分割するとパッチ数（トークン数）は

$$N = \frac{HW}{P^2} \tag{2.10}$$

となる．各パッチをベクトル化して R^{P^2C} に展開し，線形射影により埋め込み次元 D へ写像することで，パッチトークン列 $X_p \in R^{N\times D}$ を得る：

$$X_p = \mathrm{Proj}(\mathrm{Flatten}(\mathrm{Patch}(X))) \in R^{N\times D}. \tag{2.11}$$

このとき，トークン数が推論計算量を支配する主要因となり，後続の自己注意計算においてに対する二次依存が現れる点が重要である．

[CLS] トークンと位置埋め込み　ViT では系列先頭に分類用の `[CLS]` トークン $x_{\mathrm{cls}} \in R^D$ を付与し，`[CLS]` と N 個のパッチトークンを連結した系列に対して，学習可能な 1 次元の位置埋め込み $E_{\mathrm{pos}} \in R^{(N+1)\times D}$ を加算する：

$$Z_0 = \left[x_{\mathrm{cls}};\, X_p\right] + E_{\mathrm{pos}}, \qquad Z_0 \in R^{(N+1)\times D}. \tag{2.12}$$

この設計により，Transformer Encoder は画像の空間情報をトークン列として処理できる．なお，ViT では 2 次元位置埋め込みの導入よりも，単純な 1 次元学習可能埋め込みが採用されている．

2.2.3 Encoder 構造と特性（計算量・限界・変体）

Transformer Encoder による特徴抽出　ViT の Encoder は L 個のブロックからなり，各ブロックは Multi-Head Self-Attention（MHSA）と MLP（FFN）を含む．Pre-LayerNorm 構成（正規化 $\rightarrow$ サブレイヤ $\rightarrow$ 残差）で表すと，$l = 1, \ldots, L$ に対して以下のように記述できる：

$$Z'_l = \mathrm{MHSA}(\mathrm{LN}(Z_{l-1})) + Z_{l-1}, \tag{2.13}$$

$$Z_l = \mathrm{MLP}(\mathrm{LN}(Z'_l)) + Z'_l. \tag{2.14}$$

最終層出力 Z_L から `[CLS]` トークンに対応する表現を取り出し，分類ヘッド（MLP Head）に入力することでクラス確率を得る．

ViT の特性と限界　ViT は大規模事前学習を通じて高い性能を示す一方で，(i) 学習に大規模データを要しやすい点，(ii) 固定パッチ分割により局所構造の表現が弱くなり得る点，(iii) Self-Attention の $O(N^2)$ 計算により推論コストが増大しやすい点，といった課題が指摘されている [10]．

代表的な ViT 変体　DeiT は学習設計の工夫により効率的な学習を目指す [12]．T2T-ViT は Token-to-Token 機構で局所性を強化する [13]．Swin Transformer は Window Attention と階層構造で計算効率と表現力の両立を図る [14]．LV-ViT は token 活用や学習戦略の工夫により性能向上を狙う [15]．

表 2.2: 代表的な ViT 変体

モデル	主な狙い
DeiT	学習効率の改善（学習設計・蒸留等）
T2T-ViT	トークン化の改良による局所性強化
Swin	Window Attention + 階層化による効率化
LV-ViT	学習戦略・token 活用の改良

2.3　モデルプルーニング（従来の計算削減）

モデルプルーニング（model pruning）は，学習済みモデルから不要なパラメータや構造要素を削減し，計算量（FLOPs）・メモリ使用量を削減する代表的なモデル圧縮手法である．Vision Transformer（ViT）に対しても，(i) 埋め込み次元や MLP 中間次元の削減，(ii) Multi-Head Self-Attention（MHSA）の head 削減，(iii) Block（層）単位の削減など，モデル構造側を縮小して推論を軽量化する研究が報告されている．本節では，ViT におけるモデルプルーニングを，非構造化（重みスパース化）と構造化（Head ／チャネル／ Block）の観点から整理し，次節で扱うトークン削減（系列長 L の短縮）との違いを明確にする．

非構造化モデルプルーニング　非構造化モデルプルーニングは，重み行列 W の要素単位で重要度に基づき除去し，スパース性（sparsity）を導入する．典型的には magnitude pruning のように，閾値 τ によりマスク m を決定する：

$$m_i = \mathbf{1}\left(|W_i| \geq \tau\right), \quad W' = W \odot m, \tag{2.15}$$

ここで $m_i \in \{0, 1\}$ は各重みの保持／削除を表す．ViT では，主に線形層（Q, K, V 投影や MLP）の重みに対してスパース性を導入する形で適用されることが多い．評価上は，パラメータ数や理論的な乗算回数を削減できる一方で，GPU 上で実効速度を得るにはスパースカーネルやデータレイアウト最適化が必要となり，密行列実装に比べて実装依存性が高い点が課題となる．また，高い削減率を目指すほど精度劣化が顕在化しやすく，**再学習（fine-tuning）** を伴うケースが多い．

構造化モデルプルーニング　構造化モデルプルーニングは，まとまった構造単位（Head，特徴次元，Block）を削減することで，**密行列演算のまま** FLOPs を確実

に削減できる利点がある．ViT の代表例としては，MHSA における head pruning（head 単位の削減）や，MLP 中間次元（チャネル）の削減，あるいは Block（層）の削除が挙げられる．一般に，グループ単位（例：ある head に対応する重み群，ある中間次元に対応する重み群）で正則化を課すことで，重要度の低いグループを削除へ誘導する：

$$\min_{W} \ \mathcal{L}(W) + \lambda \sum_{g \in \mathcal{G}} \|W_g\|_2. \tag{2.16}$$

評価上は，FLOPs 削減がそのまま推論時間短縮につながりやすい一方で，モデル構造（次元や head 数，層数）が変化するため，事前学習済み重みの再配置や再学習が必要になりやすく，デプロイ（ONNX/TensorRT）時には shape 互換性・実装整合性の管理が課題となる．

ViT における代表的研究　ViT を対象としたモデルプルーニングの代表例として，内部次元（Attention/MLP 次元）を重要度に基づいて削減する方向性が報告されている（例：Vision Transformer Pruning: VTP）[16]．これらは，モデル構造側を縮小することで FLOPs を削減し，密演算のまま推論コスト低減を狙う点に位置づけられる．

モデルプルーニングの位置づけと限界　以上より，モデルプルーニングは「モデル構造そのもの」を削減することで計算を減らす一方，高い削減率では精度劣化を補うための fine-tuning が必要となることが多く，また GPU 上の実効速度は（非構造化では特に）実装・カーネルに依存する．これに対し，本研究が扱うトークン削減は，モデルの重み形状を大きく変えずに系列長 L を短縮することで Self-Attention の計算を直接軽量化できる点が特徴である．ただし，入力依存で系列長 L' が変化する点は，推論エンジンの静的最適化や量子化校正と衝突し得るため，次節以降で整理する Token Pruning / Merging 手法と同様に，実運用（デプロイ）観点での整合性が重要となる．

2.4　トークン削減手法の関連研究

Vision Transformer（ViT）における推論計算の主要因の一つは，Self-Attention がトークン系列長 L に対して二次（$O(L^2)$）に増大する点にある．そのため，推論時にトークン数そのものを削減し，Attention 計算およびメモリアクセスを軽量化する研究が盛んに行われている．
前節で述べたモデルプルーニングは，重みや構造を削減することで計算量を下げるのに対し，本節で扱うトークン削減は，入力系列長 L を直接短縮することで，Self-Attention の二次計算（$O(L^2)$）を抑制する点に特徴がある．本節では，代表的な削除系と結合系の手法を整理する．

トークン削減手法は大きく，(i) 重要度の低いトークンを除去して以降の演算を省略する手法（Pruning / Dropping）と (ii) トークンを結合して系列長を短縮しつつ情報を保持しようとする手法（Merging）に分類できる．前者は削減効果が大きい一方で不可逆な情報損失を伴いやすく，後者は精度を維持しやすい反面，結合規則や類似度計算に基づく追加処理が必要となる場合がある．また，多くの手法は入力ごとにトークン数が動的に変化するため，静的形状を前提とする推論エンジン最適化との整合性が課題として残る．

2.4.1 Token Pruning / Token Dropping（動的トークン削減）

Token Pruning / Dropping は，推論の途中で重要度の低いトークンを除去し，以降の計算を省略することで高速化を狙う手法群である．一般に，各トークン x_i に対して重要度スコア s_i を推定し，上位 k 個のトークン集合 $\mathcal{S}_k$ のみを保持する：

$$\mathcal{S}_k = \mathrm{TopK}(\{s_i\}_{i=1}^{L}, k), \quad X' = \{x_i \mid i \in \mathcal{S}_k\}, \quad L' = k. \tag{2.17}$$

これにより Self-Attention の計算は概ね $O(L^2)$ から $O(L'^2)$ へ削減できる一方，除去されたトークンは復元できず（不可逆），重要情報の欠落による精度劣化を招く可能性がある．また，入力ごとに L' が動的に変化するため，静的形状を前提とする推論エンジン最適化や量子化校正（calibration）との整合性が課題となる．

DynamicViT[17] では，推論中に段階的にトークンを間引くことで ViT の計算量を削減する手法を提案している．これは，各 Transformer ブロックに軽量な Importance Predictor を挿入し，トークン重要度を推定して上位のみを保持することで，後段の Self-Attention / MLP の入力系列長を短縮する設計である．評価では，精度低下を抑えつつ計算量を削減できることが報告されている．利点は，トークン数そのものを直接減らせるため理論上の高速化余地が大きい点である．一方，(i) 重要度推定誤差により早期に重要トークンを除去すると回復できない点，(ii) Predictor の追加により実装が複雑化する点，(iii) L' が入力依存で動的に変化する点が実運用上の課題である．

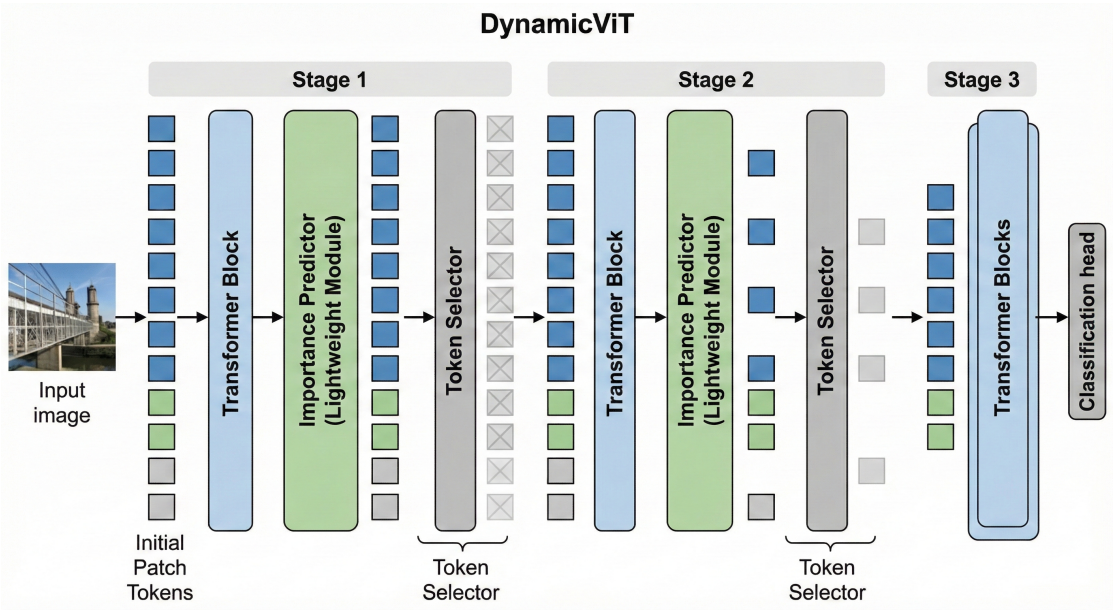

図 2.4: DynamicViT における段階的トークン削減の概念図

図 2.4 に示すように，Stage ごとにトークン数を減らす設計は，後段ほど計算量を大きく抑えられる反面，誤った削除が精度に直結するという性質を持つ．

EViT[18] では，Self-Attention により得られる注意重みを重要度指標として利用し，追加の予測モジュールを導入せずにトークン選択を行う手法を提案している．具体的には，CLS トークンから各画像トークンへの注意重みを用いて重要度を評価し，重要なトークンのみを保持して後段へ伝搬させる．評価では，追加モジュールを抑えつつトークン数削減による推論軽量化が可能であることが報告されている．利点は，Attention 情報を直接使うため構成が比較的単純で，Pruning のための外付け Predictor を不要とする点である．一方で，(i) 注意重みの推定は入力依存で変動しやすく安定性に影響し得る点，(ii) トークン除去が不可逆である点，(iii) L' が動的に変化する点は共通の課題として残る．

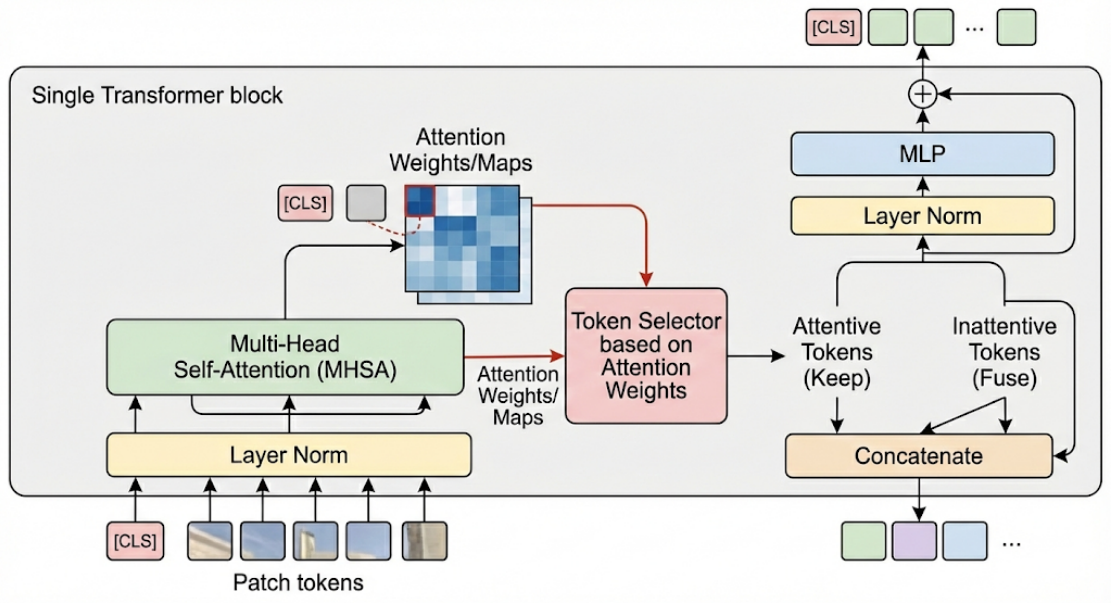

図 2.5: EViT における Attention に基づくトークン選択

Pruning/Dropping 系の位置づけ　以上の Pruning/Dropping 系は，トークン数を直接削減できるため計算量削減効果が大きい一方で，不可逆な情報損失と動的系列長 L' を伴う点が本質的な制約である．特に，本研究が扱う学習後量子化（PTQ）では校正時に統計を収集する必要があり，入力ごとに L' が変化すると校正の前提（固定形状）と衝突する可能性がある．このため，動的トークン削減を実運用（推論エンジン + PTQ）へ統合するには，校正と推論の整合性を保つ設計が重要となる．

2.4.2　Token Merging（削除ではなく結合）

Token Merging は，重要度の低いトークンを単純に削除するのではなく，類似するトークン同士を結合（merging）して系列長を短縮するアプローチである．文献 [19]（ToMe）では，各 Transformer ブロックにおいてトークン間の類似度に基づいてペアを形成し，段階的にトークンを統合することで計算量削減を行う．結合を用いることで情報を完全に破棄せず，統合表現として保持できるため，Dropping に比べて精度劣化を抑えやすい点が狙いである．

ToMe の利点は，既存の ViT 構造を大きく変更せずに適用可能であり，学習済みモデルに対して追加学習を行わずに推論コストを削減できる点である．一方で，(i) 推論中に系列長 L' が変化する点は Pruning 系と共通であり，(ii) 類似度計算やマージ規則に依存して追加処理が必要となる場合がある．特に，PTQ の校正は固定形状を仮定して統計を収集することが多いため，動的な L' を持つ ToMe を

統合する際には校正・推論の整合性を保つ設計が重要となる．この点が，本研究における ToMe と PTQ の統合課題の中核である．

表 2.3: ViT における計算削減手法の整理

カテゴリ	アイデア	利点	主な課題	再学習
モデルプルーニング	重み/構造（次元・Head・Block）を削減	構造化なら密演算で高速化しやすい	fine-tuning 依存／疎は実効速度が実装依存	要
Token Pruning / Dropping	重要度で Token を除去し $L\downarrow$	削減効果が大（$O(L^2)\downarrow$）	不可逆／動的 L' ／ PTQ 校正・静的最適化と衝突	要
Token Merging（ToMe）	類似 Token を結合し $L\downarrow$（情報保持）	精度低下が小／追加学習なし	動的 L' ／結合実装／校正・デプロイ整合	不要

2.5 量子化の関連研究

近年の GPU では Tensor Core をはじめとする低精度演算ユニットが広く普及しており，INT8 や FP16 などの低精度演算を活用した推論高速化が実運用上の重要課題となっている．モデル量子化は，重みおよび活性値を高精度な浮動小数点表現から低ビット幅の整数表現へ変換することで，計算量およびメモリ帯域の削減を実現する代表的手法である．

2.5.1 QAT（量子化対応学習）

量子化対応学習（Quantization-Aware Training; QAT）は，学習過程において量子化誤差を明示的に考慮しながら最適化を行う手法である．一般に，QAT では順伝播時に擬似量子化（Fake Quantization）を挿入し，逆伝播では Straight-Through Estimator（STE）を用いて勾配を近似的に伝搬させる [20, 21].

代表的な一様量子化では，実数値 x を以下のように整数値へ写像する：

$$\hat{x} = \mathrm{clip}\Big(\Big\lfloor \frac{x}{s} \Big\rceil + z,\ q_{\min},\ q_{\max}\Big), \tag{2.18}$$

ここで s はスケール，z はゼロ点，$[q_{\min}, q_{\max}]$ は量子化可能範囲を表す．QAT ではこの量子化演算を順伝播に組み込みつつ，逆伝播時には

$$\frac{\partial \hat{x}}{\partial x} \approx 1 \tag{2.19}$$

と近似することで学習を継続する．

QAT は量子化後の推論精度を高く維持できる点で優れているが，再学習が必須であり，大規模モデルや学習資源が制限された環境では実運用上のコストが大きいという課題がある．

2.5.2 PTQ（学習後量子化）

学習後量子化（Post-Training Quantization; PTQ）は，学習済みモデルに対して追加学習を行わず，少量の校正データを用いて量子化パラメータを推定する手法である．代表的な PTQ 手法として，重みおよび活性値の分布統計量（最小値・最大値やヒストグラム）に基づいてスケールとゼロ点を決定する方法が知られている [22, 23]．

典型的な対称量子化では，スケール s を

$$s = \frac{\max(|x|)}{2^{b-1} - 1} \tag{2.20}$$

として定義し，b ビット整数へ写像する．PTQ は再学習を必要としないため導入が容易であり，迅速なモデルデプロイが求められる実運用環境に適している．一方で，活性値分布が非対称または裾の重い場合，量子化誤差が大きくなり精度劣化を招きやすいという欠点がある．

特に Transformer 系モデルでは，GELU や Softmax などの非線形演算により活性分布が歪みやすく，CNN に比べて PTQ が困難であることが指摘されている [24]．

2.5.3 PTQ4ViT（Twin-Uniform Quantization）

PTQ4ViT は，Vision Transformer における活性分布の特性に着目した PTQ 手法である [25]．ViT では Self-Attention や GELU により，多数の小さな値と少数の大きな外れ値（outlier）が混在する非一様な分布が頻繁に観測される．このような分布に対して単一スケールの一様量子化を適用すると，外れ値に引きずられて量子化分解能が低下する問題が生じる．

PTQ4ViT では活性分布を二つの領域に分割し，それぞれに異なるスケールを割り当てる Twin-Uniform Quantization を導入する．すなわち，密集領域と疎領域に対して

$$x \to \begin{cases} Q_1(x; s_1, z_1), & x \in \mathcal{R}_{\text{dense}} \\ Q_2(x; s_2, z_2), & x \in \mathcal{R}_{\text{sparse}} \end{cases} \tag{2.21}$$

のように異なる量子化写像を適用することで，重要な値域における表現精度を維持する．

この手法により，追加学習を行わずに ViT に対する INT8 量子化の精度劣化を大幅に抑制できることが報告されている．本研究では，この PTQ4ViT を Token Merging と組み合わせた場合の実機推論性能への影響を検証対象とする．

2.6 研究目的（問題定義）

本研究の目的は，再学習を必要とせず，学習済み Vision Transformer（ViT）に対して同等精度帯を維持したまま，実機 GPU 上での推論速度向上とモデル軽量化を両立する統合的な推論高速化手法とデプロイ手順を確立することである．本研究では推論最適化のみを対象とし，量子化対応学習（QAT）や蒸留等の追加学習は前提としない．評価対象は ViT-B/16 および ImageNet-1K とし，Top-1 精度と実機推論における throughput/latency を評価指標として採用する．実験環境・測定手順・比較条件の詳細は第 4 章で述べる．

本研究で扱う ToMe は推論中にトークン系列長を動的に変化させ得るため，静的形状を仮定しがちな量子化校正および推論エンジン最適化と設計上の不整合が生じる可能性がある．さらに量子化は理論上の演算削減だけでは性能向上に直結せず，Q/DQ ノード配置，フォールバック（FP 実行），reformat/cast といった実装・ランタイム要因により実機性能が左右される．したがって本研究では，理論計算量の低減ではなく，実機 GPU 上での実効スループット改善を最終目標として問題を定義する．

以上を踏まえ，本研究では，(1) ToMe と PTQ を再学習なしで統合し，実機 GPU 推論として安定に動作するパイプラインを構成できるか，(2) 同等精度帯において ToMe 単独／ PTQ 単独／統合の各条件でスループットをどの程度改善できるか，(3) 実機性能を支配する要因（Q/DQ 配置，フォールバック，タイル整合性等）は何であり，どの条件で低精度演算の有効性が最大化されるか，という三点を研究課題（RQ）として設定する．

本研究の貢献は，ToMe と PTQ4ViT を統合する際に生じる形状・校正・実行の衝突点を体系的に整理し，解決指針を提示する点にある．加えて，ONNX（Q/DQ，動的形状）から TensorRT（INT8）へのデプロイ手順を確立し，実機 GPU 上での評価パイプラインを構築する．さらに，ViT-B/16 を対象に精度・速度・モデルサイズの観点から統合手法を定量評価し，性能差が生じる要因を分析することで，統合による速度向上の成立条件を明確化する．

2.7 ToMe と PTQ の統合における課題

ToMe は推論中にトークン系列長を変化させる手法である一方，PTQ は統計収集（校正）と最適化の観点から静的なテンソル形状を前提とすることが多い．こ

の両者を同時に適用する場合，単純な組み合わせでは解決できない課題が顕在化する．

第一に，ToMe により系列長 L' が入力ごとに変動すると校正時の統計収集（observer/metric）が不安定化し，校正停止や再現性低下を引き起こし得る．第二に，Q/DQ 配置が疎である場合には INT8 伝播が途切れ，reformat/cast の挿入や FP へのフォールバックを誘発し得る．特に動的形状 ONNX では最適化上の制約も加わるため，Q/DQ の配置方針が性能に直結する．第三に，L' の変動により GEMM 形状がタイル境界（例：32/64）から外れると最適カーネルが選択されず性能が低下し得る．また Softmax や LayerNorm 等の FP 実行が残る場合，INT8 化による効果が頭打ちになる可能性がある．

以上の課題を踏まえ，本研究では実機 GPU 推論として機能する統合パイプラインを構成し，性能向上の成立条件を明確化する．

2.8 まとめ

本章では，本研究に必要な事前知識と関連研究を整理した．まず Transformer の基本構成（Embedding，Self-Attention，FFN，残差・正規化）を確認し，Self-Attention が系列長に対して二次（$O(L^2)$）に計算・メモリコストを増大させることが，推論高速化における主要なボトルネックであることを示した．続いて ViT の入力設計（Patch/CLS/Positional Embedding）と Encoder 構造を整理し，パッチ数 N がそのままトークン系列長として Attention 計算を支配する点を明確化した．

次に，計算削減の関連研究として，モデル構造側を縮小するモデルプルーニングと系列長 L を直接短縮するトークン削減手法を対比した．モデルプルーニングは FLOPs を確実に削減できる一方で，精度維持のために再学習が必要になりやすく，また実効速度が実装やカーネルに依存するという実運用上の制約を持つ．これに対しトークン削減は，Self-Attention の二次計算を直接抑制できるが，Pruning/Dropping では不可逆な情報損失が生じやすく，さらに多くの手法で系列長 L' が入力依存で動的に変化する．Token Merging（ToMe）は，トークンを結合して情報を保持しつつ L を短縮でき，追加学習なしで適用可能という点で，本研究の前提に適合することを整理した．

さらに量子化について，QAT と PTQ の位置づけを概観し，本研究では再学習を前提としないため PTQ を採用することを明確にした．一方で Transformer/ViT では活性分布の歪みや outlier により PTQ が難しい場合があり，PTQ4ViT は Twin-Uniform Quantization によりこの問題を緩和する代表的手法であることを述べた．

以上の調査より，ToMe と PTQ4ViT の併用は「再学習なしで推論を高速化する」という観点で有望である一方，(i) 動的系列長 L' と校正統計収集の不整合，(ii) Q/DQ 配置や動的形状制約に伴うフォールバック・reformat/cast，(iii) タイル整

合性や FP 演算残存による実効性能の頭打ち，といった統合時の課題が顕在化することを確認した．次章では，これらの課題を踏まえて提案手法の全体構成を示し，統合を成立させる設計方針と実装上の工夫，ならびに評価の再現可能な手順を述べる．

第3章　提案手法

第 2 章で整理した統合上の課題であり，本章では解決策と実装を示す．まずは Token Merging（ToMe）と学習後量子化（PTQ4ViT）を統合した提案手法の全体像を示し，推論時の設計方針と実装上の工夫について述べる．本章で提案するのは，ToMe や PTQ4ViT 自体の新規アルゴリズムではなく,「動的トークン削減（ToMe）と PTQ（PTQ4ViT）を再学習なしで実機 GPU 推論に成立させる統合設計法」である．本手法は,（i）校正と推論のモード分離,（ii）Q/DQ 配置による INT8 伝播の確保,（iii）ONNX（動的形状）→ TensorRT（INT8）デプロイ手順, の 3 点を中核として構成される．本研究の狙いは，再学習を行わずに同等精度帯を維持しつつ，実機 GPU 上での推論スループット向上とモデル軽量化を両立することである．また、精度–速度トレードオフを明確にする。特に，(i) ToMe による動的トークン長と PTQ 校正の静的形状要求の衝突，(ii) Q/DQ 配置と実機カーネル選択（Tensor Core / フォールバック），(iii) ONNX（動的形状）→ TensorRT（INT8）デプロイの 3 点を中心に整理する．

3.1　全体像

提案手法の全体パイプラインを図 3.1 に示す．入力画像に対して PyTorch 上で ViT に ToMe を挿入し，PTQ4ViT により校正（calibration）を行う．校正後，Q/DQ ノードを含む ONNX を出力し，TensorRT により INT8 エンジンを生成して実機推論を行う．本パイプラインは「再学習不要」「動的トークン長」「実機デプロイ前提」の 3 点を中心に設計した．

ToMe によるトークン削減は計算量（FLOPs）とメモリアクセスの双方を低減し得る一方，量子化による実機性能は Q/DQ 配置やフォールバック（FP 演算への戻り）に強く依存する．そこで本研究では，(i) 校正時の形状不整合を避ける校正時 r=0 制御，(ii) GEMM 前後に Q/DQ を近接配置するデータ経路短縮，(iii) Tensor Core のタイル親和性を考慮した r 設計，を導入し，実機で効果が顕在化する統合設計を行う．

表 3.1: 本章で用いる主な記号と定義

記号	定義
L	ToMe 適用前のトークン列長（例：ViT-B/16, 224⇒ $L = 197$）
L'	ToMe 適用後のトークン列長（画像内容と r により変動）
r / $r^\star$	各ブロックでマージするペア数（推論時の設定値）
d	埋め込み次元（例：ViT-B/16 では $d = 768$）
$\mathcal{D}_{cal}$	校正（calibration）に用いるデータ集合
qparams	量子化パラメータ集合（scale, zero-point, threshold 等）
Q/DQ	Quantize/DeQuantize ノード（INT8 と FP の境界）
fallback	INT8 カーネルが選択できず FP16/FP32 実行へ戻る現象

Algorithm 1 End-to-End pipeline for ToMe + PTQ4ViT deployment

procedure DeployQuantizedToMe(model, $\mathcal{D}_{cal}$, $r^\star$)
  model ← InsertToMe(model) ▷▷ ToMe patch (CLS protected), see Alg. 2
  SetMode(model, `calib`)
  model ← ApplyIntegrationFix(model) ▷▷ set r=0 and override attention, see Alg. 5
  qparams ← PTQ4ViT_Calibrate(model, $\mathcal{D}_{cal}$) ▷▷ calibration loop, see Alg. 3 (uses Alg. 4)
  SetMode(model, `infer`)
  SetMergeRate(model, $r^\star$)
  engine ← ExportAndBuild(model, qparams) ▷▷ ONNX(Q/DQ, dynamic) + TensorRT build, see Alg. 6
  metrics ← Benchmark(engine)
  return engine, metrics
end procedure

Alg. 1 は，ToMe と PTQ4ViT を統合した推論パイプラインを，校正段階（calibration）と推論段階（inference）に分離して記述している点が特徴である．本アルゴリズムでは，量子化校正における形状安定性（静的形状）と，推論時における動的トークン削減（動的 L'）の両立を目的として，マージ率 r を段階的に切り替える設計を採用している．

具体的には，校正段階では，まず ToMe をモデルに挿入した上で（Alg. 2），形状安定化のために r=0 を設定し，さらに attention forward を上書きすることで，監視ノードや metric が参照する中間テンソルを確実に生成する（Alg. 5）．その後，固定された形状のもとで PTQ4ViT の校正ループを実行し，量子化パラメータ $qparams$ を推定する（Alg. 3）．PTQ4ViT 内では外れ値を考慮した Twin-Uniform を用いて，スケール候補の探索を行う（Alg. 4）．

一方，推論段階では，校正で得た $qparams$ を固定したまま，マージ率を $r^\star$ に設定して ToMe を有効化し，入力ごとに動的なトークン削減（$L \to L'$）を許容す

る．このとき，ONNX へのエクスポートでは Q/DQ ノードと動的次元（dynamic axes）を明示的に含め，TensorRT 上で INT8 エンジンを生成可能とする（Alg. 6）．

このように Alg. 1 は,「校正時の安定性」と「推論時の動的最適化」を処理段階ごとに分離することで，ToMe と PTQ4ViT の統合を実機推論において成立させる実装指針を与えている．

アルゴリズム間の呼応関係（コードの対応） 上位手順は Alg. 1 であり，図 3.1 の各処理ブロックに [Alg.k] として対応付けている．校正フェーズでは，形状安定化のため r=0 を設定し，監視に必要な中間テンソル生成を保証するため attention override を行う（Alg. 5）．その上で，qparams の推定は PTQ4ViT 校正ループ（Alg. 3）により実施し，外れ値を扱う概念手続きとして Twin-Uniform（Alg. 4）を参照する．推論フェーズでは，校正で得た $qparams$ を固定したまま，$r^\star$ を設定して ToMe を有効化し，各 Transformer block の MHSA 直後で ToMeMerge（Alg. 2）が実行されることで，$L \to L'$ の動的系列長が実現される（図 3.2）．

最後に，ONNX への Q/DQ 反映と動的次元指定，および TensorRT INT8 build は Alg. 6 に従う．

表 3.2: Algorithm 1–6 の位置づけ

アルゴリズム	役割	呼び出し関係
Alg. 1	提案手法の End-to-End 手順（校正→推論→デプロイ→評価）	章全体の上位手順
Alg. 2	ToMe のマージ処理（MHSA 直後, CLS 保護）	推論フェーズで各 block に適用
Alg. 3	PTQ4ViT 校正ループ（qparams 推定）	Alg. 1 の校正フェーズから呼び出し
Alg. 4	Twin-Uniform の手続き（outlier-aware）	Alg. 3 内部で使用
Alg. 5	統合の実装手順（校正時 r=0 + override）	Alg. 1 の校正設計を具体化した実装版
Alg. 6	ONNX（Q/DQ, dynamic）出力と TensorRT INT8 build	Alg. 1 のデプロイ工程として呼び出し

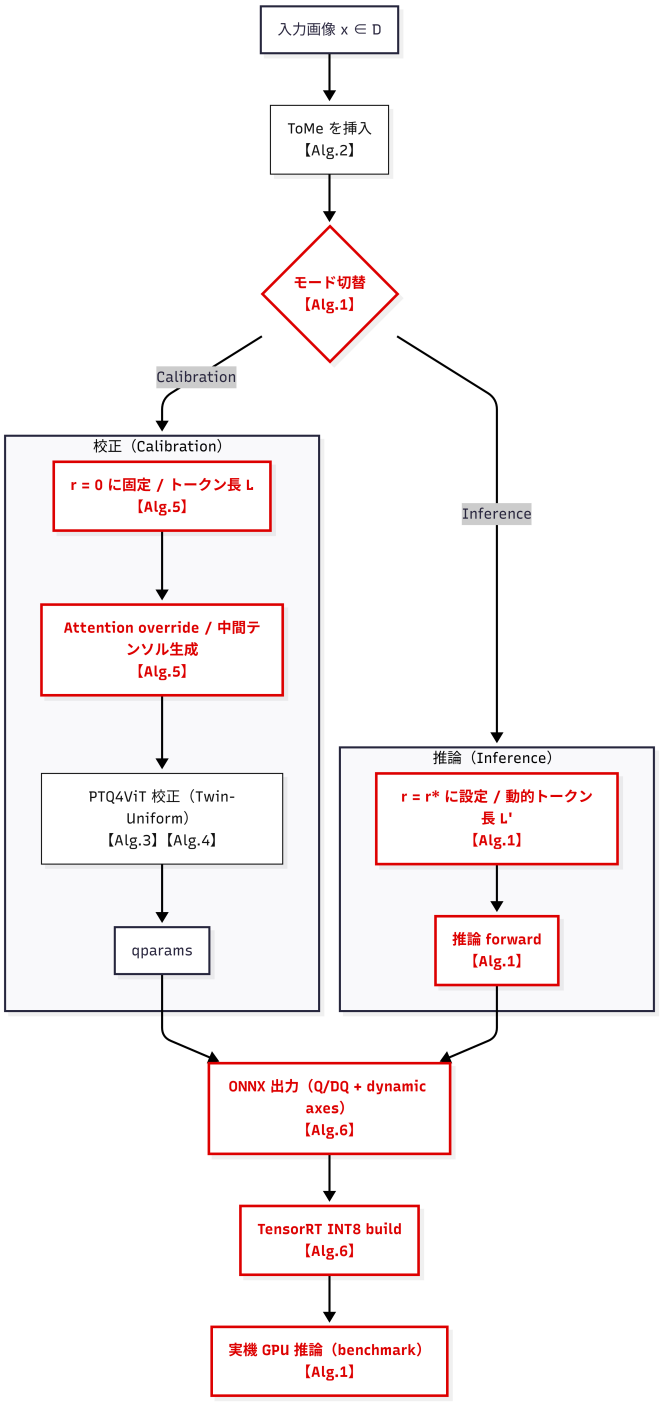

図 3.1: 提案手法の全体パイプライン（フローチャート）．PyTorch（ToMe 挿入）→ PTQ4ViT 校正（Twin-Uniform）→ ONNX（Q/DQ，動的形状）→ TensorRT（INT8）→ 実機推論

3.2 実装と貢献

本研究では既存 OSS（ToMe 実装，PTQ4ViT 実装，ONNX/TensorRT ツール群）を基盤として用いつつ，**ToMe と PTQ4ViT を動的形状・実機 GPU 推論まで一貫して成立させるための統合設計と実装**を筆者が担当した．利用した OSS と改変・追加箇所を表 3.3 に整理する．

表 3.3: 使用 OSS と本人貢献の切り分け

項目	利用した OSS（既存）	本人貢献（改変・追加）
ToMe 適用	ToMe の timm patch / matching 実装	CLS 保護の確認，r 制御（校正 r=0 ／推論 $r^\star$），挙動可視化・計測コード
PTQ4ViT 校正	PTQ4ViT 校正ループ（Twin-Uniform，metric，observer）	ToMe 挿入モデルでも校正が破綻しない Attention override（中間テンソル確実化），校正手順の整理
ONNX 出力	PyTorch → ONNX export 機構	Q/DQ 配置方針（GEMM 直前後へ集約），動的 token 軸（dynamic axes）の定義・検証
TensorRT 推論	TensorRT builder / trtexec 等	dynamic profile 設計（min/opt/max），fallback/reformat の検出・解析，実機ベンチ手順の整備
再現・実験管理	既存ユーティリティ（logging / dataset loader 等）	校正データ抽出，設定固定，ログ出力，結果集計

3.3 ToMe の設計と挿入位置

3.3.1 各ブロックでの処理（MHSA 直後のマージ）

ViT の各 Transformer ブロックにおいて，ToMe は MHSA の出力直後に挿入する．入力トークン列長を L，マージ後の列長を L' とすると，ToMe は類似度に基づきトークンを結合し，$L \to L'$ を実現する．分類タスクを想定し，CLS トークンは常に保護する（マージ対象から除外する）．

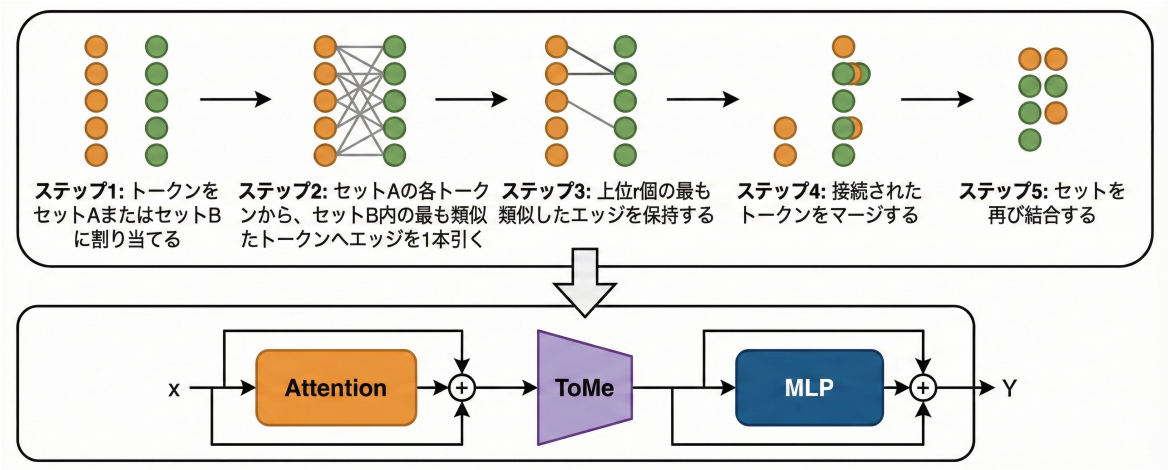

図 3.2: ToMe の挿入位置

各ブロックの MHSA 直後に ToMe を挿入し，CLS を保護しながら類似トークンをマージして $L \to L'$ とする．

3.3.2 ToMe アルゴリズム：Matching と Merge

ToMe の核は「類似トークンのマッチング」と「マージ規則」である．原版の代表的実装では，トークン集合を二分（bipartite）し，類似度が最大となるペアを選択した上で，上位 r ペアを重複無しにマージする．Alg. 2 に擬似コードを示す．**アルゴリズムの意図と位置付け．** Alg. 2 は，ToMe における代表的な「bipartite matching に基づくトークン結合」を推論時に適用する基本形を示している．本アルゴリズムは，各 Transformer ブロック内で Self-Attention により更新されたトークン表現を入力とし，意味的に近いトークン同士を選択的に統合することで，系列長を $L \to L'$ に短縮することを目的とする．

重要な点として，本アルゴリズムはトークンを削除（drop）するのではなく，情報を保持したまま結合（merge）する点に特徴がある．このため，DynamicViT や EViT に代表される Token Pruning 系手法と比較して，不可逆な情報消失を抑えやすく，精度低下が緩やかであることが報告されている．

Algorithm 2 ToMe (original-style): bipartite matching and merge (CLS-protected)

procedure ToMeMerge(X, r)
  CLS $\leftarrow X[0]$, Patches $\leftarrow X[1:]$
  $(A, B) \leftarrow$ Bipartition(Patches) ▷ ▷ e.g., even/odd split
  $S \leftarrow$ SimilarityMatrix(A, B) ▷ ▷ cosine/dot
  for $i \leftarrow 1$ to $|A|$ **do**
    $j^\star \leftarrow \arg\max_j S[i, j]$
    pairs.append$(i, j^\star, S[i, j^\star])$
  end for
  pairs $\leftarrow$ SortByScore(pairs) ▷ ▷ descending
  selected $\leftarrow$ GreedySelectNonConflicting(pairs, r)
  for all $(i, j, \_) \in$ selected **do**
    $A[i] \leftarrow$ MergeRule$(A[i], B[j])$ ▷ ▷ weighted avg etc.
    mark $B[j]$ as removed
  end for
  Patches' $\leftarrow$ Compact(A, B)
  $X' \leftarrow$ Concat(CLS, Patches')
  return X'
end procedure

計算量．類似度計算は一般に $O(L^2 d)$ を要し，類似度行列を素朴に保持するとメモリは $O(L^2)$ となる．ただし実装上は，トークン集合の分割（bipartition）や近似マッチングを用いることで定数項を削減できるほか，層ごとにマージ量を制御するスケジュール r_ℓ を導入することで計算負荷を調整できる．また，分類タスクの安定性の観点から，CLS トークンをマージ対象から除外（保護）することが重要である．

実装上の注意点．実際の推論実装においては，類似度行列 S を明示的に全保持することは避け，行単位での最大値探索や逐次的なペア選択を用いることで，メモリ使用量を抑制することが多い．また，マージ後の系列長 L' が層ごとに変動するため，後段の演算（特に量子化 GEMM）との整合性を考慮したマージ量 r の設計が重要となる．

これらの点を踏まえ，本研究では Alg. 2 をそのまま適用するのではなく，量子化校正および実機推論と整合する形で拡張・制御する．その具体的な設計方針については，次節以降で詳述する．

3.3.3 マージ量 r の設計と CLS 保護

マージ量 r は精度と速度のトレードオフを支配する最重要ハイパーパラメータである．一般に r を増加させるとスループットは向上する一方，情報損失により精度は低下する．さらに実機 GPU（特に Tensor Core）では，GEMM の行列形

状がタイル（例：32/64）と整合しない場合，カーネル選択や内部パディングにより効率が低下する可能性がある．したがって本研究では，精度だけでなく **タイル親和性（tile-alignment）**も考慮して r を設計する．タイル親和性が 32/64 などのタイル境界から外れると，GEMM 内部でパディングや別カーネルが選択される場合があり，INT8 の理論性能がそのまま出ないことがある．

具体例：r=8 におけるトークン削減の推移（ViT-B/16）． 図 3.3 に，筆者が実装した ToMe（bipartite soft matching）に基づき，ViT-B/16 に対して推論時のトークンマージ挙動を可視化した例を示す．ToMe は各層の Attention 出力後に類似トークンを統合するため，トークン列長は層を通じて段階的に減少し，例えば $197 \to 189 \to 181 \to \cdots$ と推移して，最終的に 12 層目では 101 トークンまで削減される．トークン総数 L が小さくなるほど，Attention の計算コスト（概ね $O(L^2)$）および MLP の演算コストが低下するため，本例では 1 枚あたりの計算量（FLOPs）が 17.56 GFLOPs から 12.68 GFLOPs へ低下することが確認できる．なお，本図は挙動理解のための可視化例であり，精度・速度の定量評価は第 4 章で実測により議論する．

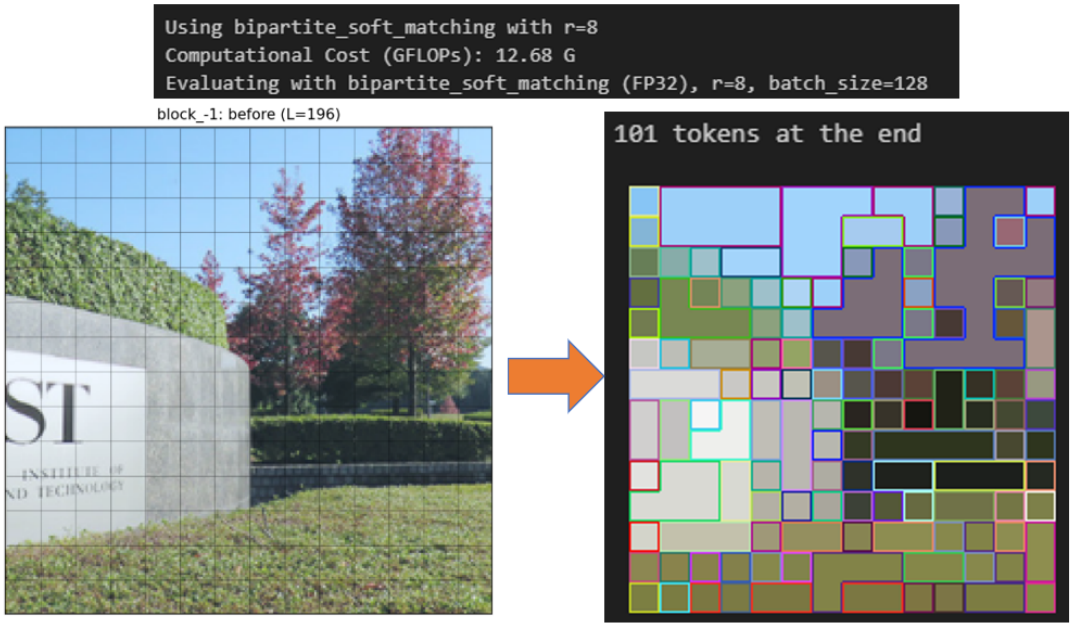

図 3.3: ToMe の可視化例（ViT-B/16, r=8）

3.4 PTQ4ViT の設計

3.4.1 校正データと監視ノード

PTQ4ViT は推論時の統計量を用いて量子化スケールを決定する学習後量子化（PTQ）である．本研究では，Transformer ブロック内の主要な線形演算に対して統計を収集する．具体的には，

- Attention：qkv 投影，出力投影（proj）
- MLP：$fc1$，$fc2$

の入出力に監視ノードを置き，活性値の分布（レンジ，外れ値など）を収集する．校正は少数のサンプルで行い，再学習や勾配更新は行わない．

3.4.2 PTQ4ViT 校正ループ

PTQ4ViT の代表的な考え方は，単純な min/max に依存せず，外れ値を含む分布に対して Twin-Uniform などの戦略を用いて量子化誤差を抑制する点にある．また，候補スケールの評価に Hessian-guided metric を用いることで，少数の校正画像でも精度低下を抑えられることが報告されている．本研究では，GEMM 中心に INT8 を適用し，Softmax/LN は FP 系に保持する．

Algorithm 3 PTQ4ViT (original-style): calibration with Twin-Uniform and metric-based scale search

procedure PTQ4ViT_Calibrate(model, $\mathcal{D}_{cal}$)
  observers ← AttachObservers(model) ▷ ▷ qkv/proj, fc1/fc2 IO
  for all $x \in \mathcal{D}_{cal}$ **do**
    Forward(model, x)
    UpdateStats(observers) ▷ ▷ range/outliers/hist
  end for
  for all op ∈ QuantizableOps(model) **do** ▷ ▷ Linear/MatMul mainly
    candidates ← GenerateCandidates(op)
    bestScore ← ∞
    for all $\theta \in$ candidates **do**
      ApplyTwinUniform(op, θ) ▷ ▷ outlier-aware
      score ← Metric(op) ▷ ▷ Hessian-guided / recon
      if score < bestScore **then**
        bestScore ← score
        qparam[op] ← θ
      end if
    end for
  end for
  return qparam
end procedure

Alg. 3 は，PTQ4ViT における代表的な校正ループを概念的に示したものであり，学習済みモデルに対して再学習を行うことなく，少量の校正データ $\mathcal{D}_{cal}$ から量子化パラメータを推定する処理を表している．

本アルゴリズムの特徴は，単一の min/max 統計に基づくスケール決定ではなく，複数の候補スケール Θ を評価し，出力誤差や感度指標（Hessian-guided metric など）を用いて最適な量子化パラメータを選択する点にある．これにより，外れ値を含む活性分布に対しても，INT8 量子化時の精度劣化を抑制できる．

統計収集（stats collection）の計算コストは，校正データ数に比例して $O(|\mathcal{D}_{cal}| \cdot \mathrm{Cost}(f))$ となる．また，量子化スケールの探索（scale search）では，N_{ops} 個の量子化対象演算子それぞれについて候補集合 Θ を評価するため，追加コストは $O(N_{\mathrm{ops}} \cdot |\Theta| \cdot \mathrm{Cost}(\mathrm{metric}))$ で表される．一般に INT8 化の効果は GEMM で最も大きく，一方で Softmax や LayerNorm は数値安定性の観点から FP（FP16/FP32）に保持されることが多い．

3.4.3 Twin-Uniform（概念）

Algorithm 4 Twin-Uniform quantization (conceptual)

procedure TwinUniform(a, b, τ)
  $a^{in} \leftarrow \{a : |a| \le \tau\}$, $a^{out} \leftarrow \{a : |a| > \tau\}$ ▷ ▷ split
  $(Q_{in}, Q_{out}) \leftarrow$ FitTwoUniformQuantizers(a^{in}, a^{out}, b)
  $\hat{a} \leftarrow$ PiecewiseQuantize$(a, Q_{in}, Q_{out}, \tau)$
  return $\hat{a}$
end procedure

Alg. 4 は，Twin-Uniform 量子化の基本的な考え方を概念的に示したものである．実際の PTQ4ViT 実装では，inlier / outlier の分割やスケール推定はヒストグラム近似や統計的閾値探索により行われ，本擬似コードはその処理フローを簡略化して表現している．

3.4.4 Twin-Uniform と Q/DQ の配置方針

本研究の実装では，Q/DQ ノードを GEMM 直前後に密に配置し，量子化・逆量子化の間隔を短くする（データ経路短縮）．一方で，Softmax および LayerNorm は FP 系（FP16/FP32）に残す．

なぜ Q/DQ 配置が kernel selection に効くのか． TensorRT では演算子が INT8 カーネル（Tensor Core）を選択するために，**その演算子入力が INT8（または INT8 を維持可能な量子化表現）として伝播すること**が重要になる．Q/DQ が GEMM から離れて配置されると，中間に FP 演算や型変換が挟まり，INT8 の伝播が途切れて FP16/FP32 カーネルへフォールバックしやすい．さらに不要な reformat/cast が挿入されると，理論演算量が減っていても実測性能が悪化する場合がある．したがって本研究では，GEMM の直前で quantize，直後で dequantize という配置を基本とし，INT8 の適用範囲を GEMM 中心に集中させる．

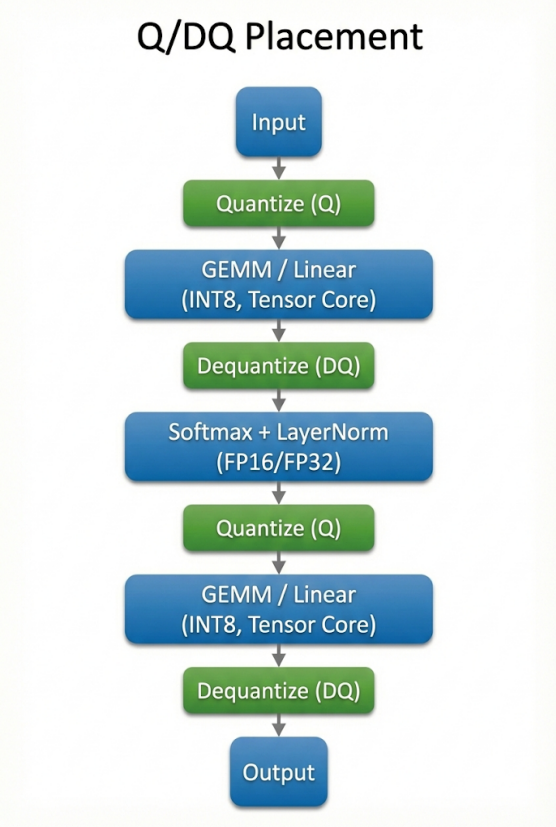

図 3.4: Transformer 層内における Q/DQ 配置

GEMM には INT8 を適用し，Softmax/LN は FP に保持する．図 3.4 に示すように，Q/DQ ノードを GEMM の直前後に近接配置することで，INT8 表現の伝播を維持しやすくなり，TensorRT による INT8 カーネル選択が促進される．

3.5 統合時の問題点と対策

3.5.1 動的トークン数 vs 静的校正の衝突

ToMe は入力画像の内容に応じてトークンを結合するため，推論時のトークン長 L' が画像毎に変動し得る．一方，PTQ の校正は一般に固定形状を仮定して統計を収集するため，動的な L' は統計収集の停止や不整合を引き起こす可能性がある．この衝突は ToMe と PTQ を統合する上での主要な実装的課題である．

3.5.2 対策：校正時 r=0 と Attention override

本研究では，校正時には ToMe のマージ量を r=0 に固定し，トークン長を一定に保つことで校正を安定化させる．また，Attention 計算に対して override を導

入し，(i) softmax を毎回明示的に実行し，(ii) 監視ノードが参照する中間テンソルを必ず生成・保存する，という制約を満たすように実装した．

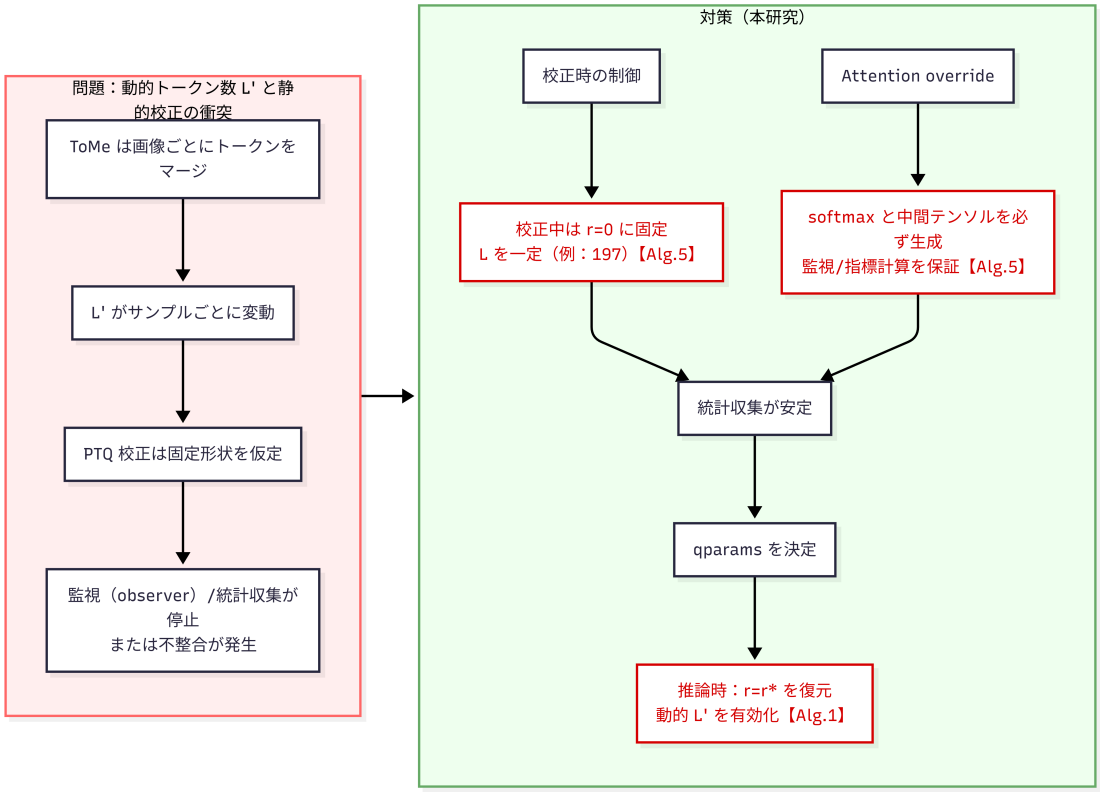

図 3.5: 統合時に発生する問題（動的 L' と静的校正の衝突）

図 3.5 は，赤字（赤枠）は本研究の対策実装，黒字は既存 OSS を表す．ToMe により推論時系列長が入力依存で変化する一方，PTQ 校正では統計収集のために形状の一貫性が必要である，という不整合を示している．本研究では「校正」と「推論」のモードを明確に分離し，校正時は系列長を固定して統計収集を保証することで，両手法の統合を成立させる．

Algorithm 5 Integration fix: calibration-time r=0 + attention override for ToMe-injected model

```
procedure ApplyIntegrationFix(model)
    model.r ← 0
    for all block ∈ model.blocks do
        block.attn.forward ← custom_attn_forward
    end for
    return model
end procedure
```

Alg. 5 は，ToMe 挿入モデルに対して PTQ4ViT の校正を安定に実行するための**校正前処理**（r=0 による形状固定と attention override）をまとめたものである．本アルゴリズムは，動的 L' と静的校正の衝突を回避し，監視ノードや metric が参照する中間テンソルを確実に生成することを目的とする．

なお，推論時のマージ率 $r^\star$ への復帰は，上位手順 Alg. 1 の `SetMergeRate`（推論フェーズ）で行う．

3.6 実装詳細

3.6.1 ONNX 出力（Q/DQ と動的形状）

校正済みモデルを Q/DQ ノードを含む ONNX グラフとして出力する．ToMe により推論時のトークン長が変動し得るため，ONNX では動的次元として L（あるいは L'）を扱えるように設計する．ただし動的形状は最適化探索を難しくするため，r 設計で形状のばらつきを抑えることが重要である．

3.6.2 TensorRT（INT8）による実機デプロイ

TensorRT により INT8 エンジンを生成し，実機 GPU 上で推論を行う．INT8 化の主対象は GEMM であり，Tensor Core（IMMA/WGMMA）を用いた高速カーネルの選択を期待する．一方，Softmax や LayerNorm は FP 系にフォールバックする可能性があるため，フォールバックの発生箇所とその影響を第 5 章で分析する．

Alg. 6 は，校正で得た量子化パラメータを ONNX（Q/DQ）へ反映し，動的トークン次元を含むプロファイル設計の下で TensorRT INT8 エンジンへ変換する一連の処理を示す．ここでの要点は，Q/DQ を GEMM 周辺に集中させて INT8 表現の伝播を維持し，不要な reformat/cast の挿入や FP カーネルへのフォールバックを抑える点にある．

Algorithm 6 ONNX export with Q/DQ (dynamic token axis) and TensorRT INT8 build

```
procedure ExportAndBuild(model, qparams)
                                                    ▷ — Export ONNX with Q/DQ —
    InsertQDQ(model, qparams)
    KeepFP({Softmax, LayerNorm})
    onnx ← ExportONNX(model, dynamic_axes={token_len})
    ValidateQDQ(onnx)
                                                    ▷ — Build TensorRT engine —
    engine ← BuildTensorRT(onnx, profiles={min,opt,max}, INT8=on)
    return engine
end procedure
```

3.6.3 ToMe のプラグイン化

ToMe を推論時に適用するため，本研究では推論エンジン内部で ToMe を実行可能とする設計を採用する．具体的には，TensorRT の動的形状に対応した拡張インタフェースを用いて，ToMe 処理をプラグインとして実装する．これにより，ONNX グラフ外での事前処理を必要とせず，推論グラフ内で動的にトークン数を削減できる．

ToMe を TensorRT 上で実行する際には，以下が設計上の要点となる．第一に，トークン長が入力ごとに変動するため，バッチ内で可変な L' を扱えるようワークスペースを管理する必要がある．第二に，分類精度への影響を抑えるため，CLS トークンはマージ対象から除外し，不変性を保証する．第三に，不規則なトークン削減は後段 GEMM の効率を損ね得るため，r の設定はハードウェア親和性（タイル整合など）を考慮して制約する．

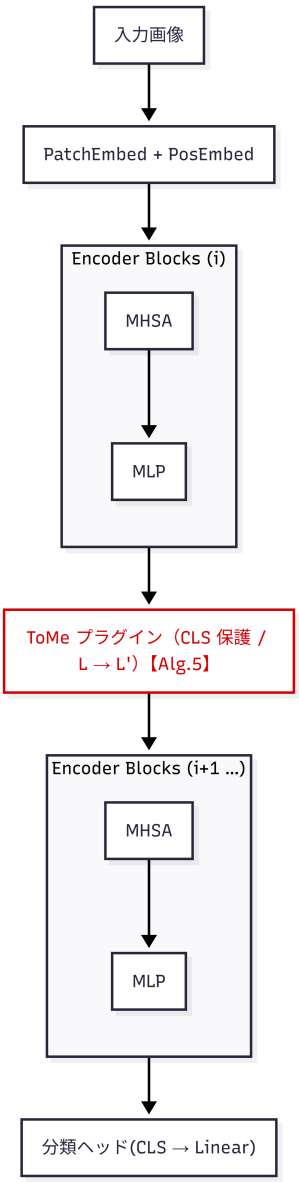

図 3.6: TensorRT の推論グラフ内に ToMe の挿入位置（MHSA 直後）

赤枠は本研究で扱う ToMe 適用点を示し，[Alg.5] は対応する処理を示す．図 3.6 は，Alg. 5 で述べた「推論時に $r^\star$ を有効化する」段階を，TensorRT エンジン内部の処理として実現するイメージを示す．前節までの議論（校正時 r=0 による形状固定，Q/DQ による INT8 化）と矛盾せず，校正は PyTorch 側で静的形状として実施し，推論時のみ ToMe プラグインにより動的な L' を許容する，という役割分担を明確にする．

3.7　本章のまとめ

本章では，ToMe と PTQ4ViT を再学習なしで統合し，実機 GPU 推論（ONNX → TensorRT INT8）まで一貫して成立させる統合設計法を提示した．具体的には，動的トークン長と静的校正の衝突に対し，校正時 r=0 と Attention override による安定化を示した．さらに，Q/DQ を GEMM 直前後に近接配置し，Softmax/LN を FP に保持することで，INT8 伝播と fallback 抑制を両立する実装指針を整理した．次章では，評価指標・比較条件・実験環境を定義し，実機 GPU 推論を含む評価を行う．

第4章　実験評価

本章では，実験環境，評価指標，GPU 実機推論を含む評価パイプラインを示し，比較条件（ベースライン）を明確化する．特に，(i) 時間計測の測定区間（どこからどこまでを測るか），(ii) 複数回試行による誤差範囲（mean±std），(iii) 第三者が同一条件で再現できる実行手順，を明示し，再現性を重視して記述する．

以降，評価指標は throughput（img/s）を主とし，精度は ImageNet-1K の Top-1 accuracy を用いる．また，表中の throughput は原則として Sec. 4.4 の E2E 定義（I/O を含む）に従い，$N = 5$ 回の独立試行の mean±std を示す．

4.1　実験環境

本研究の主要評価は NVIDIA H100 を搭載する GPU サーバ上で実施した．H100 は Tensor Core（IMMA/WGMMA）による低精度演算に強みを持ち，INT8 推論の実機性能を評価する上で適切である．ソフトウェアスタックは CUDA 12 系，相当の TensorRT 10 系，PyTorch および timm を用いた．

表 4.1: 実験環境

項目	内容
GPU	NVIDIA H100（Tensor Core: IMMA/WGMMA）
CPU	AMD EPYC 7313
OS	Ubuntu 22.04.5 LTS
CUDA	12.9
TensorRT	10.13.3.9
PyTorch / timm	2.5.1 / 0.4.12
推論精度	FP32 / INT8（TensorRT）

4.2 モデル・データセット

4.2.1 モデル

評価対象は Vision Transformer ViT-B/16（入力解像度 224）とした．ViT は入力パッチ数に比例して計算量が増加するため，Token Merging による L 削減と量子化の統合評価に適している．

4.2.2 データセット

ImageNet-1K 検証セットを用いて Top-1 精度を評価した．PTQ の校正には ImageNet-1K 訓練セットからサブサンプルした校正用データを用いた．

表 4.2: データセットと用途

データ	用途	備考
ImageNet-1K train	PTQ 校正（calibration）	32 / 128 サンプル
ImageNet-1K val	精度評価（Top-1）	全データで評価

4.3 再現性（Reproducibility）

評価がブラックボックスにならないよう，本研究では (1) 使用環境（表 4.1），(2) 校正データ抽出条件（表 4.2），(3) 測定区間の定義（Sec. 4.4），(4) 測定設定（表 4.3），(5) 実行手順（下記チェックリスト）を明示することで第三者が同一条件で再現できる形で記述する．乱数 seed は固定し，校正サンプル抽出と測定の両方で同じ seed を用いる．

再現チェックリスト

1. 表 4.1 の環境（CUDA/TensorRT/PyTorch/timm など）を固定する．
2. 校正サンプル抽出および測定の乱数 seed を固定する（例：`seed=0`）．
3. ImageNet train から校正データ $\mathcal{D}_{cal}$ を抽出する（$|\mathcal{D}_{cal}| \in \{32, 128\}$）．
4. 校正は形状を安定化させるため $r = 0$ に固定して実施し，qparams を決定する（章 3 参照）．
5. 校正済みモデルから Q/DQ と動的トークン軸を含む ONNX を出力し，TensorRT で INT8 engine を生成する（章 3 参照）．

6. ImageNet val 全体で Top-1 accuracy を評価する（同一前処理）.

7. throughput は Sec. 4.4 の測定区間定義に従い，表 4.3 の条件で $N = 5$ 試行し，mean±std を算出する.

4.4 時間計測の定義

throughput（img/s）の測定区間を明確化するため，本研究では，End-to-End（E2E）throughput を定義する.

End-to-End throughput（E2E） E2E は「入力バッチの取得〜Top-1 出力まで」を測定対象とし，(1) DataLoader/前処理，(2) Host→Device 転送（H2D），(3) 推論（PyTorch forward または TensorRT enqueue），(4) Device→Host 転送（D2H），(5) 後処理（argmax）を含む．本章で報告する throughput はすべてこの E2E 定義に従う．

同期（GPU timing） GPU 計測は非同期実行の影響を受けるため，計測区間の前後で同期を行う．PyTorch では `torch.cuda.synchronize()` を用い，TensorRT では CUDA event または同等の同期を用いて測定区間を揃える.

E2E 評価の妥当性 本研究の throughput 測定は，E2E 定義を採用している．I/O 干渉を排除した合成データテスト（Synthetic Data Test）では，H100 はより高い絶対値を達成可能であるが，本研究の目的は実際のパイプライン処理における提案手法の有効性を検証することにある．実験結果は，同一の I/O ボトルネック条件下においても，本提案手法が顕著な性能向上を実現したことを実証している.

表 4.3: 測定設定

項目	設定
Batch size	128（全条件で固定）
Warm-up	200 iterations
Measured iterations	1000 iterations
Trials	$N = 5$（独立試行，mean±std を報告）
Seed	0（校正サンプル抽出・測定で固定）
Timing scope	E2E と Engine-only を定義（Sec. 4.4）
GPU sync	区間前後で同期（cudaSynchronize 相当）

4.5　ベースラインと評価指標

4.5.1　ベースライン

本研究では以下を比較対象として用いる．FP32 は PyTorch 実行，INT8 は TensorRT エンジンとして実行し，同一バッチサイズ・同一測定方法で throughput を比較する．

- **FP32（PyTorch）**：量子化・ToMe なしの標準推論.
- **PyTorch 既定 PTQ（偽量子）**：FakeQuant による疑似量子化（主に精度挙動確認用）.
- **TensorRT INT8（NVIDIA 既定 PTQ）**：一般的な INT8 化（比較対象）.
- **TensorRT INT8（PTQ4ViT）**：ViT 向け設計の PTQ（本研究で採用）.
- **TensorRT INT8 + ToMe**：上記 INT8 に ToMe を統合した条件（提案）.

4.5.2　評価指標

- **Top-1 accuracy（%）**：ImageNet-1K val における分類精度.
- **Throughput（img/s）**：単位時間あたり処理画像数（大きいほど良い）.
- **Model size（MB）**：モデル／エンジンの保存サイズ.

4.5.3　統合に伴う注意（ToMe + PTQ）

校正（calibration）は forward を複数回実行して統計を収集するため，計算コストは概ね $O(|\mathcal{D}_{cal}| \cdot \mathrm{Cost}(f))$ に比例する．また，実機推論 throughput は，(i) Q/DQ 配置（INT8 伝播とフォールバック）と，(ii) 動的トークン長が TensorRT の profile 選択やカーネル効率に与える影響に依存して変化し得る．したがって，本研究では Q/DQ の配置方針および r の設計を明示し，比較条件間で測定手順を統一して評価した.

4.6　単独手法の評価

4.6.1　ToMe 単独（FP32）の精度–速度トレードオフ

まず，ToMe の単独効果を確認するため，FP32（PyTorch）実行においてマージ量 r を変化させ，Top-1 精度と throughput の関係を測定した（表 4.4）．r の増加に伴い token 数が減少し，MHSA/MLP の計算量が低下するため throughput は増加する．一方で，過度なマージは情報損失を招き，精度低下として現れる．

表 4.4: ToMe 単独適用の結果（ViT-B/16, 224, FP32/PyTorch，mean±std, $N = 5$）

r	Top-1 Acc (%)	Throughput (img/s)	Speed-up
0	84.54	714.09 ± (2.67)	1.00
1	84.46	742.65 ± (2.80)	1.04
2	84.41	771.22 ± (2.89)	1.08
3	84.33	792.64 ± (3.05)	1.11
4	84.23	821.20 ± (3.11)	1.15
5	84.13	856.91 ± (3.25)	1.20
6	84.08	878.33 ± (3.45)	1.23
7	84.00	906.89 ± (3.51)	1.27
8	83.79	949.74 ± (3.65)	1.33

表 4.4 の要点．FP32 条件では ToMe によって最大 1.33× の throughput 向上が得られた．一方で r を大きくするほど精度低下が増えるため，以降の統合実験では「同等精度帯」を満たす r を中心に議論する．

4.6.2　PTQ4ViT：偽量子（FakeQuant）と実機 INT8 の違い

PyTorch の偽量子（FakeQuant）経路は量子化・逆量子化を模擬するが，主要演算（GEMM）が Tensor Core の INT8 カーネルへ置換されない場合が多い．そのため，偽量子の throughput は実機 INT8 の速度を反映せず，主に精度挙動や校正の妥当性確認に用いるべきである．

表 4.5 の要点．偽量子経路は速度面で不利になりやすく，throughput の比較対象としては適切ではない．以降では，TensorRT により実際に INT8 化したエンジンの測定値を主に用いる．

表 4.5: 偽量子（FakeQuant）経路の観測例（$r = 0$，mean±std, $N = 5$）

条件	Throughput (img/s)	相対値
FP32（PyTorch）	714.09 ± (2.67)	1.00
PTQ4ViT（偽量子）	446.82 ± (1.61)	0.626

4.7 デプロイ後（TensorRT INT8）の結果

4.7.1 主要結果の概要（FP32 / NVIDIA PTQ / PTQ4ViT）

次に，TensorRT によりデプロイした INT8 エンジンの基本性能を示す．表 4.6 は FP32 baseline，INT8（NVIDIA 既定 PTQ），INT8（PTQ4ViT）を同一環境で測定した結果である．throughput は Sec. 4.4 の E2E 定義に従い，各値は $N = 5$ 回試行の mean±std を示す．

表 4.6: 主要結果（H100 TensorRT，mean±std, $N = 5$）

手法	Top-1 Acc (%)	Throughput (img/s)	Model Size (MB)
FP32 baseline	84.54	714.09 ± (2.67)	331.00
INT8（NVIDIA PTQ）	83.67	2105.37 ± (11.54)	89.80
INT8（PTQ4ViT）	84.13	2204.87 ± (11.98)	85.33

表 4.6 の要点．TensorRT INT8 により，FP32 に対して約 2.95–3.09× の throughput 向上が得られた．またモデルサイズは FP32 の 331MB から，INT8 では約 85–90MB に削減された．さらに PTQ4ViT は NVIDIA 既定 PTQ と比較して，精度・速度・サイズのいずれにおいても有利な傾向が確認できる．

4.7.2 統合結果：INT8（PTQ4ViT）+ ToMe の r スイープ

PTQ4ViT により TensorRT INT8 エンジンを生成した後，ToMe のマージ量 r を変更して throughput と精度の変化を測定した（表 4.7）．INT8 条件下でも r の増加に伴い throughput は増加するが，精度低下とのトレードオフが存在するため，「同等精度帯」を満たす r の選択が重要となる．

表 4.7 の要点．INT8（PTQ4ViT）を基準（$r = 0$）とすると，ToMe により最大で 1.149× の追加 throughput が得られた．一方で精度低下も増えるため，次節では「NVIDIA 既定 PTQ と同等精度」に揃えた比較を行う．

表 4.7: INT8（PTQ4ViT）+ ToMe：r スイープ結果（H100 TensorRT，mean±std, $N = 5$）

ToMe r	Top-1 Acc (%)	ΔAcc (pt)	Throughput (img/s)	Rel. Speed
0	84.13	0.00	2204.87 ± (11.98)	1.000
1	84.11	-0.02	2214.79 ± (12.20)	1.0045
2	83.97	-0.16	2284.19 ± (12.44)	1.036
3	83.92	-0.21	2308.63 ± (12.71)	1.047
4	83.80	-0.33	2354.10 ± (12.99)	1.068
5	83.67	-0.46	2374.93 ± (13.21)	1.077
6	83.51	-0.62	2431.24 ± (13.46)	1.103
7	83.42	-0.71	2475.33 ± (13.72)	1.122
8	83.34	-0.79	2535.60 ± (13.94)	1.149

4.8 比較（NVIDIA 手法 vs 提案手法）

4.8.1 NVIDIA の汎用推論最適化（FasterTransformer）

本研究では比較対象として，NVIDIA が公開する高速推論ライブラリ FasterTransformer を参照する [26]．FasterTransformer は Transformer 系モデル全般を対象に，演算融合やカーネル最適化，低精度推論（主に GEMM を中心とする INT8 等）を含む汎用的な高速化を提供する．一方で，その設計は特定の ViT 構造（例：トークン系列長の動的変化，CLS 保護，ViT 特有の活性分布）を前提とした特化ではなく，幅広いモデルに適用可能な一般的 PTQ / 実装戦略に基づく．そこで本研究では，この汎用的な NVIDIA 既定 PTQ を実装・測定し，ViT に特化した PTQ4ViT および ToMe 統合の有効性を検証する．

4.8.2 同等精度帯における NVIDIA 既定 PTQ との比較

NVIDIA 既定 PTQ の Top-1（83.67%）と同等精度となる点として，本研究では INT8（PTQ4ViT）+ ToMe において $r = 5$ を採用する．このとき E2E throughput は 2374.93 img/s となり，NVIDIA 既定 PTQ（2105.37 img/s）と比較して、1.13 倍の throughput（12.8%の高速化）を実現した（表 4.8）．

表 4.8 の要点．精度を同一（83.67%）に揃えた条件下でも，提案統合（PTQ4ViT + ToMe）は NVIDIA 既定 PTQ より高 throughput を達成した．これは，(i) ViT 向けに校正・スケール設計を行う PTQ4ViT と，(ii) トークン系列長を削減する ToMe を，実機 INT8 デプロイとして成立させた統合設計が有効であることを示唆する．

表 4.8: 同等精度帯（83.67%）における NVIDIA 既定 PTQ と提案統合（PTQ4ViT + ToMe）の比較（mean±std, $N = 5$）

Method	Top-1 Acc (%)	Throughput (img/s)	Speed Gain
INT8（NVIDIA PTQ）	83.67	2105.37 ± (11.54)	–
INT8（$r = 5$）	83.67	2374.93 ± (13.21)	1.13×（+12.8%）

4.9 ボトルネック分析（実機性能を支配する要因）

統合手法の性能は，単なる理論 FLOPs の削減率だけでは説明できず，実行時のカーネル選択，型変換，および形状依存のハードウェア効率によって大きく左右される．本節では，実機ログおよびアーキテクチャ特性に基づき，性能を制約する要因を分析する．

4.9.1 実機推論における一般的ボトルネック

まず，モデル構造に起因する要因として，Softmax や LayerNorm が挙げられる．これらは数値安定性の観点から FP 実行（FP16/FP32）として残る場合が多く，行列演算が INT8 で高速化されるほど，これら残存 FP 部分が Amdahl の法則における下限（lower bound）として支配的になり得る．

次に，実装上の要因として Q/DQ ノードの配置が挙げられる．配置が疎である，あるいは中間に FP 演算が挟まると，INT8 伝播が途切れて reformat/cast が挿入され，INT8 カーネル選択が阻害される．本研究では GEMM 直前後に Q/DQ を配置する設計（Chapter 3）により，これらのオーバーヘッドを最小化した．

4.9.2 演算精度による加速効率の差異（NVIDIA Hopper アーキテクチャに基づく考察）

実験結果（表 4.4 および表 4.7）において，ToMe ($r = 8$) による throughput 向上率は，FP32 (TF32) 条件下で 1.33 倍であったのに対し，INT8 条件下では 1.15 倍にとどまった．理論上の計算量削減率は同一であるにもかかわらず，高速化率に差異が生じた原因は，NVIDIA H100 (Hopper) アーキテクチャの特性に起因する以下の二点によって説明できる．

- **Tensor Core の命令セット制約と Padding オーバーヘッド：**NVIDIA の技術資料によると，H100 で導入された第 4 世代 Tensor Core は，INT8 演算において Asynchronous Warpgroup MMA (WGMMA) と呼ばれる新しい命令セットを使用することで極めて高い throughput を実現する [27]．しか

し，WGMMA は行列形状に対して厳密なアライメント（例えば $M = 64$ の倍数など）を要求する [28]．ToMe によって生成されるトークン数 L'（例: 101, 147）は不規則であり，このハードウェア要求と整合しない．その結果，コンパイラ（TensorRT）は不足分を埋めるための Padding（ゼロ埋め）を挿入するか，あるいは効率の劣る旧来の命令へフォールバックする必要が生じる．FP32 (TF32) と比較して，INT8 の演算サイクルは極めて短いため，この Padding やカーネル切り替えに伴うオーバーヘッドの比率が相対的に増大し，ToMe の理論的な計算量削減効果を相殺したと考えられる．

- **Arithmetic Intensity の低下と Memory Wall への到達：**H100 の仕様上，INT8 Tensor Core の理論ピーク性能（3,958 TFLOPS）は，TF32（989 TFLOPS）の約 4 倍に達する [29]．一方で，メモリ帯域幅は演算性能の向上ほど劇的には変化しない．これにより，INT8 推論では Operational Intensity (演算強度) が低下し，処理の律速要因が演算（Compute Bound）からメモリアクセス（Memory Bound）へとシフトする．ToMe は主に「演算量」を削減する手法であるため，演算が支配的な FP32 では効果が大きく，メモリ転送や Softmax 等の非 Tensor Core 処理が支配的となる INT8 では，その加速寄与率が低下したと結論付けられる．

4.10 まとめ

本章では，ViT-B/16 を対象に，ToMe，PTQ4ViT，および統合（TensorRT INT8 デプロイ）の効果を Top-1 精度と throughput により評価した．測定の再現性を担保するため，測定区間（E2E / Engine-only）を定義し（Sec. 4.4），統一条件（表 4.3）の下で $N = 5$ 回試行の mean±std を報告した．

FP32 baseline に対し，TensorRT INT8 により大幅な throughput 向上とモデルサイズ削減を確認した．また同等精度帯（83.67%）では，PTQ4ViT + ToMe（$r = 5$）が NVIDIA 既定 PTQ よりも 12.8% 高速であり、約 1.13 倍の性能向上を確認した．

さらに，ボトルネック分析を通じて，ToMe の加速効果が演算精度（FP32/INT8）によって異なる現象を形状アライメント制約とメモリ律速の観点から説明した．これらの知見は，将来的な ViT 推論最適化において，アルゴリズムとハードウェア特性の整合性を考慮する重要性を示唆している．

第5章　おわりに

本章では，第 1 章から第 4 章で論じた手法設計および実機評価の結果を総括し，本研究によって得られた知見と貢献を整理する．また，残された課題と今後の展望について述べる．

5.1　本研究のまとめ

本研究では，再学習を必要とせずに Vision Transformer（ViT）の推論を高速化することを目的とし，動的なトークン削減手法である Token Merging（ToMe）と，学習後量化手法である PTQ4ViT を統合した推論パイプラインを設計・実装した．ToMe は推論時に冗長なトークンを結合することで Self-Attention および MLP の計算量を動的に削減し，PTQ4ViT は INT8 化によって Tensor Core を活用した演算高速化とモデルサイズの圧縮を実現する．

両手法の統合における最大の課題は，ToMe に起因する動的なトークン長と，PTQ 校正が要求する静的なテンソル形状の不整合であった．また，ONNX（Q/DQ，動的形状）から TensorRT（INT8）へのデプロイにおいては，Q/DQ ノードの配置やフォールバックの発生が実効性能を大きく左右する．本研究では，校正フェーズと推論フェーズを明確に分離する運用（校正時 r=0，推論時 r=$r^{\star}$）と，Attention override による中間テンソルの明示的な確保を組み合わせることで，校正の安定性と推論時の動的最適化を両立させる手法を確立した（第 3 章）．

NVIDIA H100 を用いた実機評価（第 4 章）では，FP32 ベースラインに対し，提案する統合パイプラインが大幅なスループット向上とモデルサイズ削減を達成することを確認した．さらに，同等精度帯において ToMe を併用することで，INT8 単独よりもさらに高いスループットが得られる条件が存在することを実証した．

5.2　本研究の貢献と得られた知見

本研究の貢献は，単に二つの既存手法を組み合わせた点にあるのではなく，**動的トークン長を含む *ViT* を，ハードウェアアクセラレーション（*INT8*）が効く状態で実機デプロイ可能な形に落とし込んだ**点にある．実験および分析を通じて，以下の重要な知見が得られた．

(1) 校正と推論の役割分離による統合の成立　PTQ の校正プロセスでは統計収集のために安定した順伝播が必要であるが，ToMe によるトークン長の変動は観測対象テンソルの形状不一致を招き，校正の失敗要因となる．校正時は形状を固定（r=0）して量子化パラメータを確定させ，推論時のみ動的削減（r=$r^\star$）を適用する設計が，両手法を矛盾なく統合するための現実解として有効であることを示した．

(2) 理論計算量と実機性能の乖離要因の特定　ToMe による FLOPs 削減は理論上有効であるが，実機性能は必ずしもそれに比例しない．特に INT8 化によって GEMM が劇的に高速化されると，相対的に Softmax や LayerNorm などの残存 FP 演算，およびデータ再整形（reformat/cast）等のメモリ律速処理がボトルネックとして顕在化する（Amdahl の法則）．したがって，安定した高速化を得るためには，モデルの理論値だけでなく，実機上の INT8 カーネル適用範囲をログレベルで管理・最大化する必要がある．

(3) トークン形状とカーネル効率の相関（Tile Alignment）　ToMe によってトークン系列長が $L \to L'$ へと不規則に変化すると，Tensor Core が要求するタイリング境界（Tile Alignment）から外れ，パディングや非効率なカーネル選択を誘発する場合がある．この形状依存性は，FP32 と INT8 で ToMe の加速効果が異なる現象を説明する上で重要な要因であり，アルゴリズム設計においてハードウェア特性を考慮することの重要性を示唆している．

5.3　今後の課題

今後の課題は，統合手法の高速化を追求するだけでなく，成立条件を一般化し，再現可能な最適化指針を確立することにある．

第一に，ボトルネックの定量的分解である．本研究で示唆された要因を検証するために，**Nsight Systems 等を用いた詳細なプロファイリング**を実施する必要がある．これにより，フォールバック箇所（Softmax/LN 等）や reformat/cast の発生源に加え，不規則なトークン形状に起因する **Padding による演算オーバーヘッド** を特定し，INT8 伝播とカーネル効率を阻害する要因を定量的に切り分けることが求められる．

第二に，ハードウェア親和性を考慮した r の最適化である．ToMe のマージ率 r は精度と速度のトレードオフを決定するだけでなく，生成されるトークン長 L' を通じて Tensor Core の効率にも影響を与える．したがって，単なる一律のマージではなく，タイル整合を考慮した r のスケジューリングや制約付き最適化へと拡張する余地がある．

第三に，多様なアーキテクチャへの適応と汎用性の向上である． 本研究では，CLS トークンを有する標準的な ViT 構造を対象に統合アルゴリズムを検証した．

しかし，Swin Transformer[14] などの階層型アーキテクチャや，Global Average Pooling[30] を採用する変体モデルにおいては，CLS トークンが存在しない，あるいはその役割が限定的である．ToMe は，CLS トークンとのアテンション重みを指標としてトークンの重要性を評価し，重要な情報を保護する設計であるため，これらのアーキテクチャへ直接適用するには限界がある．

したがって，今後は CLS トークンに依存しない汎用的なトークン削減指標の構築が不可欠である．例えば，トークン全体の統計量に基づく寄与度評価や，局所的なウィンドウ内での類似度判定など，階層構造と親和性の高いマージアルゴリズムへの拡張が検討される．さらに，物体検出やセグメンテーションといったタスクでは，画像分類とは異なる演算比率やメモリアクセスパターンを有するため，本手法の統合効果を他タスクへ波及させることは，提案手法の汎用性と実用性を高める上で極めて重要な課題である．

5.4 おわりに

本研究は，ToMe と PTQ4ViT を統合し，ONNX（Q/DQ，動的形状）を経て TensorRT（INT8）へデプロイする一連の手順を確立することで，Vision Transformer の実機推論スループットを効果的に改善できることを示した．本論文で提示したプロファイルに基づくボトルネック分析と，形状依存性を踏まえた設計指針は，今後の深層学習モデルにおける推論最適化研究の基礎となるものである．

参考文献

[1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[4] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 6105–6114, 2019.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[6] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[7] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[12] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning (ICML)*, 2021.

[13] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis E. H. Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *International Conference on Computer Vision (ICCV)*, 2021.

[14] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *International Conference on Computer Vision (ICCV)*, 2021.

[15] Zi-Hang Jiang, Qibin Hou, Li Yuan, Daquan Zhou, Rong Jin, Anlong Wang, and Jiashi Feng. All tokens matter: Token labeling for training better vision transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[16] Mingkai Zhu et al. Vision transformer pruning. *arXiv preprint arXiv:2104.08500*, 2021.

[17] Yongming Rao, Wenliang Zhao, Bo Liu, et al. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[18] Yanjie Liang et al. Evit: Expediting vision transformers via token reorganizations. In *International Conference on Learning Representations (ICLR)*, 2022.

[19] Daniel Bolya, Cheng-Yang Fu, Xiaolong Dai, and Peizhao Zhang. Token merging: Your vit but faster. In *International Conference on Learning Representations (ICLR)*, 2023.

[20] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons. *arXiv preprint arXiv:1308.3432*, 2013.

[21] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[22] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference. *arXiv preprint arXiv:1806.08342*, 2018.

[23] Markus Nagel, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

[24] Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Understanding and overcoming the challenges of efficient transformer quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[25] Zejiangh Yuan, Yuxuan Liu, et al. Ptq4vit: Post-training quantization for vision transformers. In *European Conference on Computer Vision (ECCV)*, 2022.

[26] NVIDIA. FasterTransformer. `https://github.com/NVIDIA/FasterTransformer`. Accessed: 2025-12-20.

[27] NVIDIA. Nvidia hopper architecture whitepaper (gtc 2022). `https://www.advancedclustering.com/wp-content/uploads/2022/03/gtc22-whitepaper-hopper.pdf`, 2022. Accessed: 2026-01-15.

[28] NVIDIA. Cutlass documentation: Cute wgmma tutorial (sm90 / hopper). `https://research.colfax-intl.com/cutlass-tutorial-wgmma-hopper/`, 2024. Accessed: 2026-01-15.

[29] NVIDIA. Nvidia h100 tensor core gpu datasheet. `https://resources.nvidia.com/en-us-gpu-resources/h100-datasheet-24306`, 2022. Accessed: 2026-01-15.

[30] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *International Conference on Learning Representations (ICLR)*, 2014.

研究業績

1. FENG MINGYU，高野 恵輔，井口 寧,「Token 合併と低精度計算に基づく ViT モデル高速化検証」，2025 年度 電気・情報関係学会 北陸支部連合大会，JHES25G1_4，1 page，金沢工業大学，2025 年 9 月 20 日.
   受賞：優秀論文発表賞

謝辞

本研究を進めるにあたり，熱心にご指導いただいた主指導教員の井口 寧教授，ならびに高野 恵輔助教と先輩の村上 舜様に心より感謝申し上げます．また，中間審査にて有益なご助言をいただきました田中 清史教授，青木 利晃教授に感謝いたします．

井口研究室の皆様には，日々の議論や研究生活において多くの刺激と助けをいただきました．皆様のおかげで，充実した時間を過ごすことができました．

JAIST での 2 年間は，私にとって非常に有意義な時間でした．ここでの経験を通じて，「研究とは何か」を知りたいという入学当初の目的を，自分なりに果たすことができました．

最後に，日本での生活と私の挑戦を常に信じ，温かく支え続けてくれた家族に深く感謝いたします．