C-016

TinyLLaMA の推論における TensorRT-LLM を用いた高速化

Acceleration Using TensorRT-LLM for Inference of TinyLLaMA

岡本 航太朗 †　　吉田 明正 †
Kotaro Okamoto　　Akimasa Yoshida

1　はじめに

近年, 大規模言語モデルの実用化が進み, 生成 AI や自然言語処理タスクにおいて高精度な応答が得られるようになってきている．こうしたモデルの多くは数十億から数千億の莫大なパラメータを有することから, 推論処理には多大な計算資源と時間を要する．そのため, リアルタイム応答や大量のクエリ処理を求められる場面においては, 推論速度の高速化が重要な課題となる．このような背景のもと, 高速な推論を実現するために NVIDIA が提供する TensorRT-LLM[1] が注目されている．TensorRT-LLM は, 推論エンジンの最適化や量子化を通じて, GPU における処理効率を大幅に向上させることを目的としている．本稿では, PyTorch と TensorRT-LLM を用いて TinyLlama モデルによる推論処理を行い, スループットや実行時間の観点から両者の性能を比較・評価する．

2　大規模言語モデルの推論処理

LLM は, 推論処理における計算負荷が非常に大きい．特に, Attention 機構における計算コスト, 各トークン生成の逐次性, メモリアクセスの非効率といった要因が, 処理時間やスループットの向上を阻害するボトルネックとなっている．本章では，まず PyTorch による推論の特徴について述べた後，TensorRT-LLM による最適化について概説する．

2.1　PyTorch を用いた推論の特徴

PyTorch は，研究開発を目的とした柔軟性の高い深層学習フレームワークであり，LLM（大規模言語モデル）の実装・訓練・推論において広く利用されている．特に動的計算グラフ（Dynamic Computational Graph）を特徴とし，直感的かつ柔軟なネットワーク定義が可能であるため，試行錯誤やデバッグがしやすく，多くの研究者や開発者に支持されている．一方，推論処理においては，入力に対して 1 トークンずつ逐次的に出力を生成するデコード処理が主となる．この過程では，各トークン生成のたびにモデル全体を通して計算が行われる必要があり，特に Attention 機構では前の全トークンとの相関計算を再帰的に実施するため，計算量が大きくなる．PyTorch は開発面では優れるものの，推論時の実行効率という点では必ずしも最適ではなく，専用の最適化フレームワークとの性能差が生じる要因となっている

2.2　TensorRT-LLM を用いた推論の特徴

TensorRT は，NVIDIA が開発した GPU 推論最適化フレームワークであり，ニューラルネットワークを実行効率の高い形式に変換・最適化して高速な推論を実現する．従来は画像分類や物体検出といった CV（Computer Vision）タスクにおける使用が主であったが，近年は LLM 推論にも対応が拡張されており，その代表例が TensorRT-LLM である．

TensorRT-LLM は，LLM 特有の計算ボトルネックを解消するために設計されており，複数の最適化技術を組み合わせている．具体的には, モデルを静的な最適化グラフへ変換し，演算の合成や不要な処理の除去を行い, CUDA カーネルレベルで実行できる専用エンジンを構築する．Attention 処理の高速化として FlashAttention 等のアルゴリズムを採用している．また、FP16 や FP8 といった低精度演算に最適化されており，最新の GPU アーキテクチャで Tensor Core を効率的に活用できる．

3　TensorRT-LLM を用いた推論処理

本章では，TensorRT-LLM による大規模言語モデルの高速推論を実現するための手順について述べる．まず, Hugging Face などで提供される PyTorch 実装のモデルを TensorRT-LLM で使用可能な形式に変換し，エンジンファイルを構築する過程について述べる．次に，そのエンジンファイルを用いて Docker コンテナ上で推論を実行する環境の構築について述べる．

3.1　モデル変換

TensorRT-LLM は，PyTorch などで事前学習された大規模言語モデルを最適化された形式に変換し，CUDA レベルで高速な推論を可能にするフレームワークである．推論を実行するには，まずモデルを TensorRT-LLM が要求する形式に変換し，エンジンファイルを構築する必要がある．具体的には，提供されているスクリプト convertcheckpoint.py を用い，HuggingFace 形式のモデルの重みや設定ファイルを TensorRT-LLM が内部的に処理できる形式に変換した．変換後は，TensorRT-LLM に付属する builder.py を実行し，FP16 精度を指定してエンジンをビルドした．ビルド時には，最大入力長，最大出力長，バッチサイズなどを明示的に設定することで，モデルの構造と使用目的に応じた最適化が施される．このエンジン構築ステップは一度実行すれば，以降の推論では最適化されたファイルを再利用できる．これにより，PyTorch 推論で必要とされる実行時の計算グラフ構築や動的メモリ割り当てのオーバーヘッドを回避し, Attention 計算やトークン生成処理におけるボトルネックを大幅に軽減することが可能となる．

3.2　Docker 上における推論環境

TensorRT-LLM による推論処理は，NVIDIA が提供する公式 Docker イメージを用いることで，環境構築の手間を省き，依存関係の整った状態で再現性の高い実験を行うことができる．本研究では，CUDA12.8 および TensorRT-LLM 0.20.0rc2 に対応した Docker イメージを使用し，TinyLlama のエンジンをビルド・実行した．推論は Docker コンテナ内で実行され，構築済みの TensorRT エンジンファイルを読み込むことでモデルを

†明治大学総合数理学部ネットワークデザイン学科
Department of Network Design, School of Interdisciplinary Mathematical Sciences, Meiji University

初期化している．エンジンの読み込みには，TensorRT-LLM に付属する Python API の LLM クラスを使用し，モデルディレクトリを指定することで推論環境を簡潔に構築できる．

4 NVIDIA RTX A5500 搭載 Xeon サーバにおける性能評価

本性能評価では，TensorRT-LLM と PyTorch による推論性能を評価するため，NVIDIA RTX A5500 GPU を搭載したサーバ上で多肢選択問題を用いた実験を行い，処理時間・スループット・正解率の観点から比較を行う．

4.1 性能評価環境

本性能評価には，表 1 に示す NVIDIA RTX A5500 搭載 Xeon サーバを使用した．また, 表 2 に評価に用いたクエリの形式を示す．

表 1　実行環境.

CPU	Xeon Gold 6326 16 コア
メモリ	256GB
GPU	NVIDIA RTX A5500 × 4
OS	Ubuntu 20.04LTS
CUDA	12.8
TensorRT	10.9.0.34
TensorRT-LLM	0.20.0rc2
PyTorch	2.6.0

表 2　評価に用いたクエリの形式.

ベンチマーク形式	問題数	回答形式
BoolQ(二択) 形式	400 問	True / False
MMLU(四択) 形式	400 問	A / B / C / D

4.2 四択形式と二択形式のクエリを用いた推論性能評価

まず，図 1 に示すように四択形式においては，PyTorch(FP32) における一問当たりの平均推論時間が 0.14[s], FP16 では 0.15[s] なのに対して, TensorRT-LLM は 0.09[s] であり，PyTorch (FP32) に対して 36%, FP16 に対して 40%の処理時間短縮が達成された．さらに，図 2 スループットの平均値においては TensorRT-LLM は 100.35 [token/s] を示し，PyTorch(FP32) の 71.52[token/s]，PyTorch(FP16) の 65.29[token/s] を大きく上回っている．TensorRT-LLM の正解率は 26.50%であり，PyTorch(FP32) の 30.00%, FP16 の 26.00%と大きな差はみられない結果になった。

次に，二択形式では，正解率はすべての手法で同一の 52.25%であり，TensorRT-LLM は図 1 の一問当たり

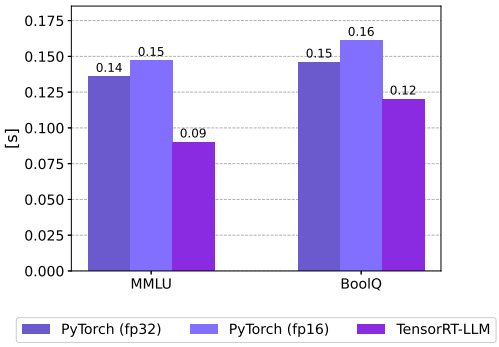

図 1　推論時間の評価.

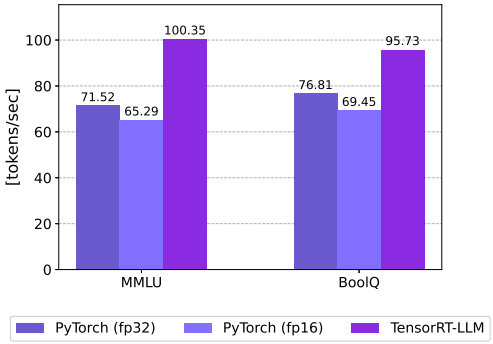

図 2　スループットの評価.

の平均推論時間において 0.12[s] と最短を記録し，PyTorch(FP32) の 0.15[s] および PyTorch(FP16) の 0.16[s] と比較して，25%近い高速化が確認された．また，図 2 のスループットでも 95.73 [token/s] と最高値を示し，PyTorch(FP32) の 76.81 [token/s] に対して 25%の性能向上を実現している．

以上の結果から，TensorRT-LLM は PyTorch と比べて推論において高速性とスループット性能が優位であることが確認できた．リアルタイム性や大量処理が求められる環境において，TensorRT-LLM の適用は非常に有効であると考えられる．

5 おわりに

本稿では，小規模モデルである TinyLlama 1.1B を用いて，RTX A5500 搭載 Xeon サーバ上で PyTorch および TensorRT-LLM による推論処理の性能を比較し，推論時間，スループット，精度の観点から評価を行った．性能評価の結果，TensorRT-LLM は PyTorch と比較して，推論時間が短く，スループットが高いことが明らかとなった．特に，BoolQ のような短文応答タスクにおいては，PyTorch(FP16) と比べて 38%高いスループットを達成し，精度を維持したまま高速化できることが確認された．一方，MMLU 形式の四択問題では，スループットにおいて優位性を示しているが，PyTorch(FP32) に対して精度がやや低下する傾向も見られた．

今後の課題としては，よりパラメータ数の大きなモデルを用いた比較実験の実施，INT8 など他の低精度形式を用いた推論精度および速度の検証を行うことが挙げられる．

参考文献

[1] NVIDIA Welcome to TensorRT-LLM ’s Documentation! *https://nvidia.github.io/TensorRT-LLM*, 2025.

[2] Dataset Card for MMLU, *https://huggingface.co/datasets/cais/mmlu*, 2025.

[3] Peiyuan Zhang, Guangtao Zeng, Wei Lu, et al. TinyLlama: An Open-Source Small Language Model, *https://arxiv.org/abs/2401.02385*, 2023.

[4] Qingyuan Li, Ran Meng, Yiduo Li, et al. ASpeed Odyssey for Deployable Quantization of LLMs, *https://arxiv.org/pdf/2311.09550*, 2023.

[5] Adam Paszke, Sam Gross, Francisco Massa, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library, *https://arxiv.org/pdf/1912.01703*, 2019.