Scaling Smarter: Scaled Dot-Product Attention and Quantized Inference for LLMs

Sarang Pinakin Kadakia, Vishwajeet Kulkarni
sk11634@nyu.edu, vk2630@nyu.edu

Abstract

Large Language Models (LLMs) like Mistral-7B offer exceptional language generation capabilities but face deployment challenges in latency-critical and resource-constrained environments. Our upcoming work focuses on optimizing LLM inference using two key techniques: low-bit quantization (via BitsAndBytes) and Scaled Dot-Product Attention (SDPA) via PyTorch's native backend. We plan to develop and evaluate five Mistral-7B variants combining these techniques and use metrics like perplexity, CUDA time profiling, and inference latency to identify the most efficient variant. Once identified, we will fine-tune the best-performing model on a domain-specific dataset to validate efficiency and quality trade-offs post-training.

Keywords: *Mistral-7B, Quantization, 4-bit/8-bit Precision, SDPA, PyTorch, BitsAndBytes, Inference Optimization, Perplexity Evaluation, CUDA Profiling*

Introduction

Large language models are transformative but often infeasible to deploy in real-time settings due to their high memory and compute requirements. Our objective is to make models like Mistral-7B more accessible and scalable by optimizing their inference path without sacrificing output quality.

We aim to:

- Reduce memory and latency overhead via 4-bit and 8-bit quantization
- Enhance attention compute efficiency through PyTorch's Scaled Dot-Product Attention (SDPA)
- Evaluate and compare optimized model variants under constrained GPU environments (e.g., NVIDIA T4)
- Fine-tune the best variant for domain adaptation and performance analysis

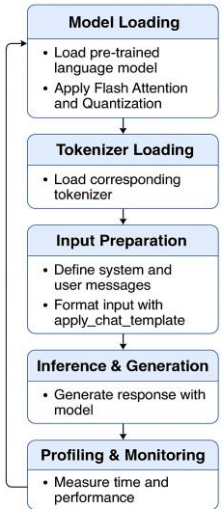
Methodology

1.1. Model Architecture

We will optimize the Mistral-7B architecture using Hugging Face Transformers, creating five model variants:

- **Original Mistral-7B** – Full-precision baseline
- **8-bit Quantized** – Moderate compression
- **4-bit Quantized** – Aggressive memory reduction using nf4
- **8-bit Quantized + SDPA** – Balanced speed and efficiency
- **4-bit Quantized + SDPA** – Max compression and optimized attention

Quantization will use BitsAndBytes. SDPA, fully compatible with T4 GPUs, will be applied using PyTorch's native backend.



The model flow (above diagram) includes:

- Loading the quantized model
- Loading the tokenizer
- Preparing chat-style inputs
- Running inference
- Profiling inference time and CUDA performance

1.2. Data Preparation

For both evaluation and fine-tuning, we will use the TREASURE ISLAND Tiny dataset, a compact literary corpus ideal for studying narrative generation and stylistic coherence. Our motivation for choosing this dataset is threefold:

- **Manageable size** – suitable for experimentation on limited hardware (e.g., T4 GPUs).
- **Rich language** – allows us to test fluency, style, and narrative structure in generation tasks.
- **Open licensing** – freely available for modification and reproduction via Project Gutenberg.

1.3. Training and Optimization Techniques

Quantization (4-bit and 8-bit): Weights will be compressed using BitsAndBytes with double quantization and nf4 format, targeting faster inference and reduced GPU memory use.

Scaled Dot-Product Attention (SDPA): To optimize attention layers, we will use PyTorch's SDPA, which fuses key attention operations into a faster and more memory-efficient computation. Unlike FlashAttention, SDPA works on T4 GPUs and integrates seamlessly with both full-precision and quantized models. We expect significant latency improvements from SDPA-enhanced variants.

Parameter-Efficient Fine-Tuning: After identifying the best variant, we will fine-tune it using PEFT strategies via Hugging Face's *Trainer*. Only a small set of parameters—like final layers or adapters—will be updated. This approach preserves speed and memory savings while improving domain adaptation on the Treasure Island Tiny dataset.

Resource Utilization: Initial prototyping will be done on T4 GPUs (16GB). For fine-tuning and in-depth CUDA profiling, we'll move to NYU's A100 HPC cluster. We'll use *torch.profiler* to analyze kernel performance, latency, and memory usage across all model variants.

Evaluation

2.1. Quantitative Analysis

- **Perplexity:** Computed using Hugging Face's *evaluate* module
- **Inference Latency:** Measured across 5 variants to identify the fastest configuration
- **Memory Usage:** Compared across precision levels
- **CUDA Time Profiling:** Visualized using PyTorch Profiler and variant-wise runtime graphs

2.2. Qualitative Analysis

- **Output Coherence:** To complement the quantitative metrics, we will manually review outputs from each model variant for linguistic quality. This includes evaluating the fluency, grammar, sentence structure, and contextual fit of generated text. We are particularly interested in whether aggressive compression (like 4-bit quantization) or the use of SDPA introduces noticeable artifacts such as stilted phrasing, repetition, or semantic drift. These observations will help us determine if the latency benefits of optimization come at the cost of readability and naturalness.

- **Fine-Tuning Adaptation:** Once the most efficient variant is fine-tuned on the Treasure Island Tiny dataset, we will assess how well the model adapts to domain-specific writing style and content. Outputs will be compared before and after fine-tuning to evaluate improvements in vocabulary usage, narrative tone, and stylistic consistency. This qualitative check is critical to ensuring that parameter-efficient tuning can enhance performance in targeted tasks without requiring full retraining or additional compute.

Conclusion

This project aims to build a practical framework for deploying large language models like Mistral-7B on constrained hardware without compromising generation quality. Through a combination of 4-bit and 8-bit quantization and attention-level optimization using PyTorch's SDPA, we expect to achieve significant gains in inference speed and memory efficiency. Our evaluation strategy—combining perplexity, latency profiling, and manual review—will help us select the best-performing model variant. By applying supervised fine-tuning only to essential parameters, we aim to further tailor the model to a domain-specific dataset like Treasure Island Tiny, demonstrating that LLMs can be both compressed and adapted efficiently. If successful, this framework could extend to real-world use cases in chatbots, summarization tools, and embedded NLP systems, especially in low-resource environments. Ultimately, this work is a step toward democratizing LLM deployment by making powerful models faster, smaller, and more accessible without sacrificing functionality.

References

Dataset:
[1] Treasure Island:
https://huggingface.co/datasets/treasure_hunt

Code:
[1] Mistral-7B Instruct – HuggingFace:
https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.1
[2] BitsAndBytes (bnb) – Github:
https://github.com/TimDettmers/bitsandbytes
[3] PyTorch SDPA Documentation:
https://pytorch.org/docs/stable/generated/torch.nn.functional.scaled_dot_product_attention.html