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

Sarang Pinakin Kadakia[1], Vishwajeet Kulkarni[2]

[1]New York University
[2] New York University
sk11634@nyu.edu, vk2630@nyu.edu

Abstract

Large Language Models (LLMs) like Mistral-7B demonstrate powerful generative and comprehension capabilities, but their high inference cost poses a major barrier to deployment in real-time and resource-constrained environments. Our project aims to make LLMs faster and more memory-efficient at inference time using two key techniques: low-bit quantization and Scaled Dot-Product Attention (SDPA)**.** We implement five Mistral-7B variants, applying 4-bit and 8-bit quantization via BitsAndBytes, and attention optimization via PyTorch's native SDPA. Our evaluation pipeline includes CUDA profiling and perplexity benchmarking. So far, inference has been tested on three variants (Original, 4-bit, and 4-bit + SDPA). The next phase involves full training and model selection.

Introduction

LLMs are transforming how machines understand and generate language, but their large size often leads to high latency and memory consumption. For organizations and researchers working on limited hardware, these limitations can restrict practical usage. Our goal is to enhance inference-time performance of Mistral-7B using hardware-aware optimization techniques:

- **Quantization**: Reducing model precision to 8-bit and 4-bit to shrink memory and improve throughput.
- **SDPA (Scaled Dot-Product Attention)**: Replacing standard attention mechanisms with PyTorch's optimized implementation to accelerate inference.

We target NVIDIA T4 GPUs, making our techniques accessible for academic and startup deployment. Evaluation focuses on latency, perplexity, and CUDA kernel profiling.

Methodology

1.1. Model Architecture

We use Mistral-7B Instruct, a decoder-only transformer, as our base model. We created five variants:

- **Original Mistral-7B** – full-precision, baseline reference
- **8-bit Quantized** – reduced weight size
- **4-bit Quantized** – aggressive compression with nf4 quantization
- **8-bit Quantized + SDPA** – fast attention plus quantization
- **4-bit Quantized + SDPA** – highest compression with fused attention

Quantization is done using HuggingFace's BitsAndBytesConfig, and SDPA is applied using PyTorch's native backend (FlashAttention is skipped due to T4 incompatibility)

1.2. Data Preparation

To evaluate inference quality, we curated a dataset from *Treasure Island*. This allows us to assess generative consistency and fluency under various compression levels.

- Dataset: System + user prompts crafted from the text
- Used to evaluate perplexity and benchmark inference speed
- Tokenized using HuggingFace tokenizer

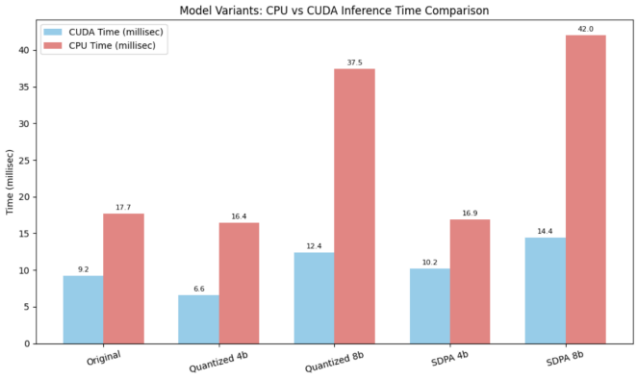
1.3. Training Procedure

No training or fine-tuning has been done yet. All evaluations to date have been conducted in zero-shot inference mode using:

- *torch_dtype=torch.bfloat16* for memory efficiency
- Consistent prompts reused across all model runs
- HuggingFace's *evaluate* with the *perplexity* metric

Evaluation

5-Way Model Comparison (Completed):



From this analysis, we shortlisted the top 3 performing variants → Original**,** 4-bit Quantized**,** and 4-bit + SDPA

Focused Evaluation of Top 3 (Completed):

Variant	Total Inference Time	CUDA Time per Op	Notes
Original	~17 ms	~0.0075 ms	Good baseline, but slowest
4-bit Quantized	**~14 ms**	**~0.0049 ms**	**Fastest & most efficient**
4-bit + SDPA	~15 ms	~0.0052 ms	Slightly slower than 4-bit alo

The 4-bit Quantized model was identified as the most optimal in terms of latency and kernel efficiency.

1.4. Training Optimizations

So far, no training has been applied. However, the pipeline includes:

- BitsAndBytes quantization with double quant + nf4
- Torch profiler for CUDA-level analysis
- SDPA fused attention via PyTorch backend
- Prompt evaluation in zero-shot mode using *evaluate* API

Project Milestones

2.1. Completed Milestones

- Implemented and evaluated 5 Mistral-7B variants
- Measured perplexity and CPU vs CUDA inference time across all variants
- Shortlisted top 3 models based on initial results
- Performed total latency and CUDA kernel efficiency analysis
- Identified 4-bit Quantized model as the best performing

2.2. Upcoming Milestones

- Fine-tune the 4-bit Quantized model on a new dataset
- Re-evaluate perplexity and latency after training
- Optionally compare it to pre-fine-tuned versions
- Document final results and publish GitHub repository

Bottlenecks and Challenges

- **T4 GPU Limitations**: FlashAttention not supported, fallback to PyTorch SDPA
- **Quantization Trade-offs**: Need to balance size reduction with output quality
- **CUDA Profiling**: Large logs required filtering to identify relevant kernel ops
- **Evaluation Consistency**: Prompts, token limits, and batch sizes needed standardization

Detailed Work Contributions

- **Sarang Kadakia:** Implemented all model variants, integrated BitsAndBytes quantization and SDPA, and performed CUDA profiling and kernel analysis.
- **Vishwajeet Kulkarni:** Handled prompt structuring, ran perplexity and latency evaluations, and led documentation, result visualization, and reporting.

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
[4] HuggingFace Perplexity Metric:
https://huggingface.co/spaces/evaluate-metric/perplexity