Optimizing Large Language Models for CPUs

Eyan P. Noronha
Department of Computer Science
Stanford University
eyannoronha@stanford.edu

1 Introduction

Large Language Models (LLMs) have revolutionized the field of natural language processing, demonstrating remarkable capabilities in tasks such as text generation, translation, and question answering. However, the deployment of these models has been hindered by the substantial computational resources required, often necessitating specialized hardware like GPUs. While GPUs excel in handling the parallel computations inherent in LLMs, they can be expensive and may not be readily available in all environments. This has led to a growing interest in optimizing LLMs for deployment on CPUs, which are more widely accessible and often more cost-effective[He et al., 2024a].

2 Problem Statement and Approach

This project will explore various optimization techniques to improve the efficiency of serving LLMs on CPUs. These techniques include:

Quantization: Quantization involves reducing the precision of weights and activations, which decreases memory consumption and accelerates inference. I will explore different quantization schemes, including 8-bit, 6-bit, 5-bit, 4-bit, 3-bit and 2-bit quantization. For 8-bit and 4-bit quantization, I will use the `bitsandbytes` package, which built into the Hugging Face ecosystem. Additionally, I also explore k-means quantization using llama.cpp (https://github.com/ggerganov/llama.cpp), which cluster weight values around centroids, enabling efficient parameter compression while maintaining model accuracy.

Pruning: Pruning involves removing unnecessary connections or neurons from a neural network. This technique can significantly reduce the model size and improve inference speed. I will experiment with different pruning algorithms, such as magnitude pruning and structured pruning using `torch.pruning` and huggingface `optimum` library, and evaluate their impact on model accuracy [Ansel et al., 2024].

Speculative decoding: Speculative decoding is a technique that involves predicting future tokens during inference. This technique can help to reduce the latency of decoding by avoiding unnecessary computation. I plan to develop a speculative decoding scheme specifically tailored for CPU-based inference.

Another important tool for optimizing LLMs on CPUs is the GGUF format (Generalized GGML Universal Format), which has been developed specifically to support LLMs optimized for CPU inference. The GGUF format allows for aggressive quantization, including 6-bit, 5-bit, 3-bit, and 2-bit quantization, which is essential for fitting large models into limited CPU memory without significant accuracy loss. GGUF also supports model sparsity and multi-threaded inference, making it highly efficient for CPU-based deployment. By enabling compact and efficient parameter storage, GGUF allows larger models to be loaded on standard CPUs, significantly reducing memory usage and enhancing throughput. Relevant literature for the project includes Shen et al. [2023], He et al. [2024a,b], Huang et al. [2024], Yue et al. [2024].

Preprint. Under review.

3 Models, Baselines, and Evaluation Setup

The optimization techniques described above will be implemented and evaluated on a variety of large language models, including GPT2-small, GPT2-medium, GPT2-large, GPT2-xl, Llama-1B, and Llama-8B. For initial benchmarking, we set up latency baselines for the GPT models and the Llama-1B model with output generation lengths of 20, 50, 100, 150, and 200 tokens. For the Llama-8B model, latency was measured for maximum output lengths of 13, 15, 17, 20, and 25 tokens, as generating more than 25 tokens required over 20 minutes of processing time on our hardware.

To ensure consistency in latency measurements, a fixed input prompt was used across all models, which were run on a MacBook Air with an M2 chip and configured to use the CPU only. Figure 1 shows the latency results for each model, arranged as follows: GPT2-small, GPT2-medium, GPT2-large, GPT2-xl, Llama-1B, and Llama-8B from left to right and top to bottom.

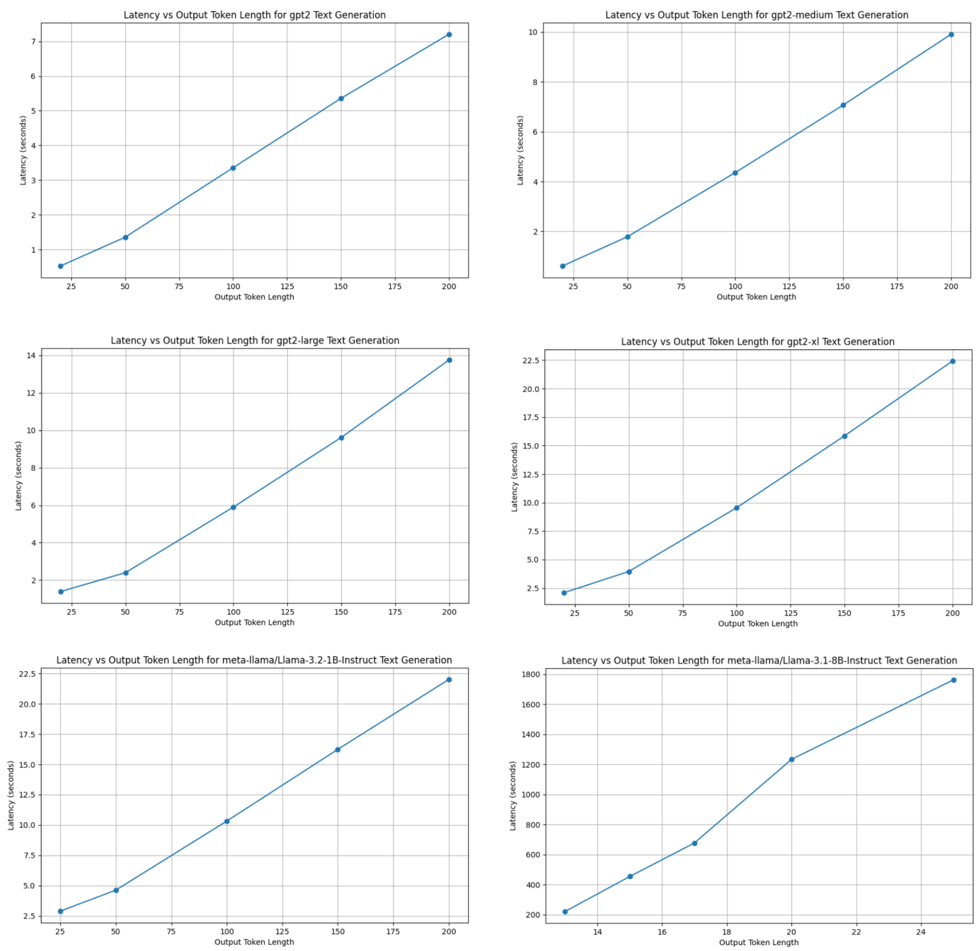

Figure 1: Latency Baselines for GPT2 small, medium, large, xl and Llama 1B and 8B.

In subsequent phases, I will evaluate these optimization techniques on a wider range of CPUs available on Google Cloud, including 10 different Intel, AMD, and Nvidia CPUs, to assess performance across various hardware setups. If time permits, I also plan to analyze the impact of quantization and pruning on specific layers of each model. This analysis will help determine how these optimizations affect model quality, latency, and throughput at different layers within the neural network, providing deeper insights into the best strategies for efficient LLM inference on CPUs.

References

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024. doi: 10.1145/3620665.3640366. URL `https://pytorch.org/assets/pytorch2-2.pdf`.

Pujiang He, Shan Zhou, Wenhuan Huang, Changqing Li, Duyi Wang, Bin Guo, Chen Meng, Sheng Gui, Weifei Yu, and Yi Xie. Inference performance optimization for large language models on cpus. *arXiv preprint arXiv:2407.07304*, 2024a.

Pujiang He, Shan Zhou, Changqing Li, Wenhuan Huang, Weifei Yu, Duyi Wang, Chen Meng, and Sheng Gui. Distributed inference performance optimization for llms on cpus. *arXiv preprint arXiv:2407.00029*, 2024b.

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Xianglong Liu, Luca Benini, Michele Magno, and Xiaojuan Qi. Slim-llm: Salience-driven mixed-precision quantization for large language models. *arXiv preprint arXiv:2405.14917*, 2024.

Haihao Shen, Hanwen Chang, Bo Dong, Yu Luo, and Hengyu Meng. Efficient llm inference on cpus. *arXiv preprint arXiv:2311.00502*, 2023.

Yuxuan Yue, Zhihang Yuan, Haojie Duanmu, Sifan Zhou, Jianlong Wu, and Liqiang Nie. Wkvquant: Quantizing weight and key/value cache for large language models gains more. *arXiv preprint arXiv:2402.12065*, 2024.