LLLM: Local LLM
Team - Cyclops

Lakshay Sethi[*] **Onkar Thorat**[*] **Pratik Mandlecha**[*]
{lsethi, othorat, pmandlec}@andrew.cmu.edu

1 Motivation

In our interconnected world, where concerns about data privacy and efficient use of resources are paramount, the need for on-device machine learning solutions has never been greater. Traditional models rely on sending data to external servers, posing privacy risks and straining network resources. Our project addresses this challenge directly by enabling local execution of the CodeLlama model, a fine-tuned 7B-parameter language model, on an 8GB RAM MacBook. By leveraging advanced quantization and compression techniques, our aim is to empower users without compromising data privacy.

Hypothetical Deployable Scenario: Consider an employee in a high-stakes company, deeply engrossed in a confidential coding project that demands precision and secrecy. In the midst of their work, they encounter a complex coding challenge that requires expert guidance. The employee wishes to utilize the advanced capabilities of the CodeLlama model for code completion and debugging. In this scenario, our project becomes invaluable. By enabling the local execution of CodeLlama on the employee's 8GB RAM MacBook, the model provides instant, accurate coding suggestions and aids in debugging, all without the need to send any data outside the company's secure environment.

2 Task definition and problem setup

2.1 Inputs and Outputs, Datasets, and Evaluation

Inputs: The inputs for our project consist of instructions for code completion or debugging, provided in text format.

Outputs: The output generated by the CodeLlama model is the corresponding code for the given instructions, also in text format.

[*]Everyone Contributed Equally – Alphabetical order

Datasets: We will utilize the Mostly Basic Python Programming (MBPP) dataset, containing a diverse range of Python programming tasks. This dataset serves as the foundation for training and evaluation.

Evaluation: The accuracy of the generated code will be assessed by running it on test cases provided by the MBPP dataset.

2.2 Target Hardware Platform

The target hardware platform for this project is an 8GB MacBook. The MacBook's computational power, combined with its portability and ease of use, makes it an ideal choice for our on-device machine learning application. The MacBook's native text input and output modalities align seamlessly with our project requirements, allowing for a smooth user experience.

2.3 Peripherals and Justification

No specific peripherals are required as the MacBook provides an integrated and comprehensive development environment. Its standard keyboard and display fulfill the input and output modalities, ensuring a self-contained setup for our project.

2.4 Input and Output Modalities

The input modality is textual instructions, and the output modality is textual code.

2.5 Existing Tools and Libraries

We will employ standard Python libraries for text processing, ensuring compatibility between the input instructions and the model. Specifically, we will utilize libraries like NLTK (Natural Language Toolkit) for text analysis and manipulation.

2.6 Training Off-Device and Cloud Compute Credits

Our project mainly involves compressing already trained model and using it for inference on a local machine. However, we shall require cloud credits for initial baselines and in case we decide to do model distillation and train a student model.

2.7 Timeline and Milestones

Week 1-2: Set up the development environment, import the CodeLlama model, and establish basic input-output functionalities.

Week 3-4: Develop the code generation module and optimize it for efficiency. Begin initial accuracy evaluations using the MBPP dataset test cases.

Week 5-6: Apply quantization techniques and model compression techniques on the working model.

Stretch Goals:

Week 7-8: Integrate Stanford MOSS for extended evaluation, comparing generated code with expected output. Address any discrepancies and refine the code generation algorithm.

3 Research questions and experimental design

3.1 Research Questions (Hypotheses)

Following are the research questions that are of our interest which we want to address through experimentation in this project.

- Can the CodeLlama model having 7 billion parameters (size of 28GB in fp32) be effectively compressed to fit within the memory constraints of an 8GB shared RAM without a significant loss in performance? Since the model itself is quite bigger than the shared RAM, we want to keep two options open for now as a backup - using 16GB shared RAM and using the disk for sharding.

- What is the relative contribution of inidvidual as well as combined techniques - pruning, quantization, tensor decomposition, and model distillation (if time and compute permits) on tradeoffs that include model parameters, accuracy, FLOPs, energy consumption, and carbon emissions?

- What are the tradeoffs by removing the attention heads of some particular blocks?

- How does the final compressed model's performance compare with original model?

3.2 Experimental Design

3.2.1 Datasets

CodeLlama is trained predominantly on a near-deduplicated dataset of publicly available code which contains many discussions about code and code snippets included in natural language questions or answers. We will employ the Mostly Basic Python Programming (MBPP) dataset, which has been used to evaluate the CodeLlama as well. It will help for a consistent and fair comparison of performance of our model with the original CodeLlama using similar metrics. A part of this dataset shall also be used as a representation dataset for static quantization.

3.2.2 Ablations

- **Individual Compression Evaluations:**
  - Pruning - Using **magnitude pruning**, **structured pruning**, and **attention heads analysis** to observe the effects of removing or altering attention heads within specific blocks.
  - Quantization - Utilize the MBPP dataset for calibration for **post-training static quantization** to at least int8. While dynamic quantization is of interest, we are cognizant of its potential to elevate inference time, especially on our constrained hardware.

- **Combination of Compression Techniques** - Recognizing that only one technique might be insufficient to meet our hardware constraints, we will explore combinations of pruning and quantization methods to achieve the desired model size and performance.

- **Model Distillation** (if time and compute permits) - We will experiment with model distillation, training a student model using the soft labels of CodeLlama. This approach is contingent upon overcoming the challenge posed by the unavailability of Meta's original training data.

3.2.3 Hardware

- Primary - MacBook Pro with 8GB shared RAM for our final compressed model to work on.

- Secondary (for training and initial experiments): For initial experiments, baselines, and inferences of models we shall either use A10 GPUs on AWS or high memory instances without GPUs.

3.2.4 Other Experiments or Analyses

Energy and battery usage - To quantify the energy consumption of the model during various stages, especially during inference on target hardware utilize hardware and software tools. To conduct tests to measure the rate of battery drain during continuous model inference compared to the device's idle state.

4 Related work and baselines

With the launch of ChatGPT in November 2022, there has been a flurry of work in language models and LLMs. While much of the effort has been put in to achieve better models with more generation and multimodal capabilities, all of this has come at the cost of the size of models getting larger and larger with model parameters in 10s of billions. Deploying these massive models therefore has proven to be a challenge due to their limited scalability and extremely high running costs and power consumption. Researchers therefore have been exploring novel ways to reduce the model size (in terms of storage) to be able to fit on smaller hardware and make the models more manageable. Quantization has emerged as the top choice for compressing large language models due to its relative ease and speed with which it can be implemented. Pruning has also shown promise however it is much more time-intensive to remove weights while maintaining similar accuracy.

To this end, LLM-QAT (Liu et al., 2023) proposes a novel Quantization aware Training(QAT) technique that uses knowledge distillation to distill 7B, 13B and 30B LLaMA models to their equivalent 4-bit (weights and key-value caches) quantized models. The authors also experimented with quantization activations to 4 bits however were unsuccessful due to the a significant increase in perplexity. Before exploring their approach, it is important to understand the reasons limiting Knowledge Distillation QAT's for LLMs. Quantization-aware Training or QAT for short is the process of training a model specifically keeping quantization and compression in mind. Knowledge distillation on the other hand is the process of using the unquantized large model as a teacher model and transferring its knowledge to the student model. This student model can be sparser or smaller in size (here size refers to the number of parameters) or have the same parameters but with less precision. QAT and Knowledge Distillation for LLMs is extremely hard owing to two main problems:

- It is challenging to find the same pre-training and training data for the LLM. Even if the pre-training data is available, it massive in size. Also, LLMs are sometimes trained in steps with instruction tuning and it is often difficult to replicate this

- LLMs are optimized for zero-shot generation and are trained on a wide array corpus encompassing a lot of domains. Any attempt to quantize an LLM must ensure that the zero-shot capabilities of LLM are maintained across domains.

The authors therefore propose a simple solution to the above two problems by using the non-quantized model to generate outputs given a prompt and then training the quantized student model using knowledge distillation. This technique allows the authors to train their language models in relatively small data (100k samples). Their novelty is in proposing thi data generation technique which the authors call *next token generation*. Furthermore, the authors propose a hybrid sampling strategy for their data generation by using a combination of top-1 prediction and stochastic sampling.The authors then compare their proposed QAT Data Free Knowledge Distillation approach with the original model and other SOTA Post Training Quantization (PTQ) techniques on Zero-Shot performance on Common Reasoning Tasks (PIQA (Bisk et al., 2019), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019)). While all models quantized using PTQ and QAT are able to retain the performance of original model upto 8-bit quantization, their approach significantly outperforms SOTA PTQ techniques when quantization is more intense at 4 bits for weights and KV Cache (The activations are still quantized to 8 bits).
For our proposed methodology of quantizing 7B Code LLaMA, we believe using this approach can be very useful if we quantize our models further down to 4 bits since PTQ techniques are insufficient and significantly worsen the performance to unacceptable levels. Lastly, we would want to highlight that 4 bits do not have any

native hardware support as of now.

SmoothQuant (Xiao et al., 2023) is another related approach in which the authors propose a training-free PTQ technique that is able to quantize both the weights and activations of any large language model. Based on empirical observations, it is a fact that weights of LLM are easy to quantize while activations are not because activations have a huge variance in their range and can have outliers. This is possibly due to weight decay and other regularization techniques that prevent the magnitude of weights from getting too high, however no such technique exists for activations. There have been attempts to find a hybrid PTQ approach such as LLM.int8() (Dettmers et al., 2022) which proposes an approach to quantize all the weights and activations to 8 bits while keeping the outlier activations to 16 bits using mixed precision. Therefore, the authors of SmoothQuant (Xiao et al., 2023) propose a novel technique to migrate the variance of activations to weights enabling the 8-bit quantization of both weights and activations without any loss to accuracy and model performance.

Moving to baselines for our proposed quantized 7B Code-LLaMA, to the best of our knowledge, we were not able to find any quantized model out there. We believe that the main research focus of LLMs quantization is for general reasoning tasks and there has been less to no work on the quantization of LLMs designed for niche tasks such as coding.

5 Potential challenges

Challenges:

a) Limited Computational Resources: One potential challenge could be encountered if the 8GB RAM MacBook does not provide sufficient computational resources to run the optimized CodeLlama model effectively. To adapt to this challenge, the project team could explore further compression techniques, such as model distillation, consider using a bigger shared RAM (16GB) or using disk and peform inference using shards.

b) Model Accuracy and User Satisfaction: If users find the suggestions generated by the optimized CodeLlama model to be inaccurate or unsatisfactory, it could affect the project's usability. To adapt to this challenge, continuous feedback loops and user testing could be implemented.

References

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. Piqa: Reasoning about physical commonsense in natural language.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm.int8(): 8-bit matrix multiplication for transformers at scale.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023. Llm-qat: Data-free quantization aware training for large language models.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence?