Project Proposal: **LLama3.1 and LLama3.2 Inference Implementation in Pure C**

Vishal Vardhan Adepu (334003567)
Texas A&M University

October 3, 2024

Project Title

LLama3.1 and LLama3.2 Inference Implementation in Pure C

Problem Statement

Context: Large Language Models (LLMs) like Meta's LLaMA series have significantly advanced the field of natural language processing by achieving state-of-the-art results across various tasks. These models are predominantly implemented in high-level languages such as Python, which may not provide the performance optimization required for deployment in resource-constrained or real-time environments. Implementing inference in pure C can offer performance gains and greater control over system resources, making it highly relevant for applications where efficiency is critical.[1]

Problem: Currently, there is a lack of pure C implementations for the inference of LLama3.1 and LLama3.2 models, especially for models in the 1B, 3B, and 11B parameter range. Existing implementations are either in higher-level languages or do not focus on the latest versions of LLaMA. Given that the differences between LLama3.1 and LLama3.2 in terms of inference code are trivial, this project aims to develop an efficient inference engine in pure C that supports LLama3.1 models (8B) and LLama3.2 models (1B, 3B, 11B). While the implementation will be scalable to handle the 90B model in LLama3.2, and 405B model in LLama3.1, running this model locally is not feasible due to memory constraints, and will therefore not be tested in this project.

Project Objectives

- **Implement** an efficient inference program in pure C for the LLama3.1 8B and LLama3.2 1B, 3B, and 11B models.
- **Design** the implementation to be scalable for larger models like the 90B parameter model, though it will not be run due to memory constraints.

[1] Karpathy's `llama2.c`: `https://github.com/karpathy/llama2.c`

- **Optimize** the code for performance and memory usage to enable execution on resource-constrained hardware.
- **Evaluate** the performance, correctness, and accuracy of the implementation against official benchmarks for both LLama3.1 and LLama3.2 models.

Methodology

Approach: The project involves building a system from scratch to perform inference on the LLama3.1 8B and LLama3.2 1B, 3B, and 11B models using pure C. The approach includes:

- Studying the architectures of LLama2, LLama3.1, and LLama3.2 to understand their components and operational mechanisms.
- Leveraging Karpathy's `llama2.c` codebase as a reference point, given its simplicity and efficiency.[2]
- Implementing transformer blocks, including attention mechanisms and feed-forward networks.[3]
- Developing the inference engine with a focus on modularity to allow seamless support for both LLama3.1 and LLama3.2 models, ensuring scalability to handle the 90B model, even though it will not be run.

LLM(s) and Techniques: The focus will be on the LLama3.1 8B model and LLama3.2 1B, 3B, and 11B models. Key techniques include implementing transformer architectures,[4] optimizing computational routines, and efficient memory management. BLAS libraries for optimized linear algebra operations will be considered.[5]
Architecture/Process: The implementation will replicate the transformer architecture of LLama3.1 and LLama3.2, which includes:

1. **Tokenization:** Processing input text into tokens compatible with the models, ensuring alignment with the official LLaMA tokenizer.
2. **Embedding Layer:** Converting tokens into vector representations.
3. **Transformer Blocks:** Implementing multi-head self-attention and feed-forward neural network layers.
4. **Output Layer:** Generating predictions based on the processed input.

Evaluation: Success will be measured through:

[2]Karpathy's `llama2.c`: `https://github.com/karpathy/llama2.c`
[3]Transformer architecture: `https://arxiv.org/abs/1706.03762`
[4]Transformer architecture: `https://arxiv.org/abs/1706.03762`
[5]BLAS library for C: `https://www.netlib.org/blas/`

- **Accuracy Comparison:** Comparing the outputs of the C implementation with Meta's official evaluation datasets for both LLama3.1 and LLama3.2, including tasks like TriviaQA,[6] MMLU,[7] and CommonsenseQA.[8]
- **Performance Benchmarking:** Measuring inference speed, memory usage, and scalability compared to existing implementations in Python.
- **Perplexity Testing:** Calculating the perplexity of the models on held-out datasets to assess prediction accuracy.

Related Work

Literature Review:

- **LLama: Open and Efficient Foundation Language Models** by Hugo Touvron et al.: Introduces the LLaMA family and provides insights into the architecture and benchmark results.[9]
- **Efficient Implementation of Transformer Models in Low-Level Languages**: Discusses methodologies for implementing transformer-based models in languages like C/C++ and explores optimization techniques.
- **Karpathy's** `llama2.c`: A minimalist implementation of LLama2 inference in C, which will be used as a reference for efficient code structuring and optimization.[10]

Positioning: This project builds on Karpathy's `llama2.c` by extending the approach to LLama3.1 and LLama3.2, focusing on implementing the 8B model for LLama3.1 and 1B, 3B, and 11B models for LLama3.2. Given the trivial differences in inference code between LLama3.1 and LLama3.2, the project aims to support both models within the same framework. The code will be designed to support the 90B model as well, although this model will not be run due to memory constraints. The project emphasizes creating a scalable architecture in pure C that can be adapted for larger models in the future, ensuring efficiency and performance gains for resource-constrained environments.

Timeline

Phase 1 (Month 1):

- **Week 1–2:** Research existing C implementations and study the architectures of LLama3.1 and LLama3.2. Investigate official evaluation datasets such as Meta-Llama-3.1-8B-evals[11].

[6] TriviaQA dataset: `https://huggingface.co/datasets/mandarjoshi/trivia_qa`

[7] MMLU dataset: `https://huggingface.co/datasets/cais/mmlu`

[8] CommonsenseQA dataset: `https://huggingface.co/datasets/tau/commonsense_qa`

[9] LLama: Open and Efficient Foundation Language Models by Hugo Touvron et al.: `https://arxiv.org/abs/2302.13971`

[10] Karpathy's `llama2.c`: `https://github.com/karpathy/llama2.c`

[11] Hugging Face Datasets: `https://huggingface.co/datasets`

- **Week 3–4:** Begin implementing core components, including tokenization, embedding layers, and the transformer blocks for the 8B, 1B, 3B, and 11B models.

Phase 2 (Month 2):

- **Week 5–6:** Complete the implementation of the transformer blocks, fine-tune the architecture, and optimize for performance on the 8B, 1B, 3B, and 11B models. Incorporate support for both LLama3.1 and LLama3.2.
- **Week 7–8:** Perform evaluation against official Meta datasets (e.g., TriviaQA, MMLU, CommonsenseQA), benchmark performance metrics (speed and memory), and prepare the final report and deliverables.

Milestones:

1. **Milestone 1 (End of Week 2):** Completion of research phase with detailed implementation plans.
2. **Milestone 2 (End of Week 4):** Basic inference engine for the 8B, 1B, 3B, and 11B models operational.
3. **Milestone 3 (End of Week 6):** Optimized models with evaluation metrics collected for both LLama3.1 and LLama3.2.
4. **Milestone 4 (End of Week 8):** Final report and code submission with scalability features for larger models, including the 90B model.

Challenges and Risks

- **Model Complexity:** The complexity of LLama3.1 and LLama3.2 may present challenges during implementation, particularly in translating the transformer architecture into efficient C code.
  - **Mitigation:** Allocate sufficient time for studying the models and refer to existing implementations. Use unit testing and gradient checking to ensure correctness.
- **Evaluation and Benchmarking:** Accessing proper datasets and benchmarking the implementation against Meta's results may be challenging.
  - **Mitigation:** Utilize Meta-Llama-3.1-8B-evals and Hugging Face datasets for evaluation.[12] Incorporate tools like `lm-evaluation-harness` to assist with benchmarking.[13]
- **Resource Constraints:** Running the 8B, 1B, 3B, and 11B models may strain local resources, and testing larger models like the 90B model will not be feasible.
  - **Mitigation:** Optimize code for memory efficiency and consider cloud services (AWS, Google Cloud) for additional testing. Focus on scalable design for future model support.

[12] Hugging Face Datasets: `https://huggingface.co/datasets`

[13] `lm-evaluation-harness`: `https://github.com/EleutherAI/lm-evaluation-harness`

Resources Needed

- **Hardware/Software:**
  - A development machine with sufficient CPU power and RAM to run the 1B, 3B, 8B, and 11B models.
  - Access to cloud computing services (e.g., AWS, Google Cloud) for potential benchmarking and testing.
  - C development tools (GCC, debugging utilities) for efficient implementation.
- **Data Requirements:**
  - Pre-trained LLama3.1 and LLama3.2 1B, 3B, 8B, and 11B model weights from authorized sources.
  - Official evaluation datasets, such as Meta-Llama-3.1-8B-evals, available via Hugging Face.[14]

Expected Deliverables

- **Code:** An efficient, modular C codebase for LLama3.1 and LLama3.2 1B, 3B, 8B, and 11B model inference, with the flexibility to extend to larger models, including the 90B model (though it will not be run due to memory constraints).
- **Final Report:** A comprehensive report detailing the methodology, challenges faced, solutions, and performance results, including:
  - Methodology, implementation, and optimization strategies.
  - Performance benchmarking (speed, memory) and comparison with official benchmarks for both LLama3.1 and LLama3.2 models.
- **Additional Artifacts:**
  - A demonstration of the models running on sample inputs.
  - Performance metrics visualizations and benchmarking results.
  - Presentation materials summarizing project outcomes.

[14]Hugging Face Datasets: `https://huggingface.co/datasets`