Imperial College London

MEng Individual Project

Department of Computing

Imperial College of Science, Technology and Medicine

A New Scalable Runtime for LLM Inference

Author:
Hamish McCreanor

Supervisor:
Peter Pietzuch

January 16, 2025

Submitted in partial fulfillment of the requirements for the MEng Computing of Imperial College London

Contents

Chapter 1

Introduction

As large language models (LLMs) are found useful for ever-wider classes of applications, a trend has arisen focusing on the low-cost, local deployment of these systems. While the training of LLMs like LLaMA, BERT and OpenAI's GPTs typically requires months of training and is prohibitive for all but the most well-funded of organisations, performing inference on these models locally is comparatively more feasible. This enables developers to create services with tighter LLM integrations - instead of calling a black-box API provided by an LLM provider, they can instead run a local version of the LLM, tuning the inference runtime to more appropriately match the context in which it is called.

As a result, there is currently a vast body of research aiming to improve existing inference systems. The aim of this is to improve LLM inference performance along various axes. These include running on lower-powered hardware; running with improved throughput and running at greater energy efficiencies. These optimisations focus on specific elements of the inference pipeline, particularly improving KV cache usage and kernel fusion. To build systems containing these optimisations, developers frequently turn to high level languages like Python in order to quickly develop the infrastructure surrounding their new technique. Developing this way limits the ability of the system to exploit memory-access patterns and application parallelism (especially in a language like Python, with its global interpreter lock) and incurs unnecessary overhead.

This project aims to build on the existing llama.cpp inference server (see 2.2.5) to deliver a system that improves the dispatch of compute kernels by better parallelising the inference pipeline.

Chapter 2

Background

We provide an overview of transformer architecture in 2.1.1, as well as key components of the inference pipeline in 2.1.2, before detailing existing work around LLM inference runtimes in 2.2.

2.1 Preliminaries

2.1.1 LLMs Explained

Transformer Architecture

While the term "Large Language Model" can be used to describe any model trained on large volumes of textual data, it is frequently used to refer to models that use a variant of the transformer architecture described in [1]. This architecture has superseded recurrent neural networks for language-based tasks owing to its ability to capture long range dependencies between tokens. It does this via a unique attention mechanism. This, and the other components present in a transformer, are described below.

- **Embedding Layers:** Attention blocks are fed a matrix where each row represents the semantic meaning of each token. This is done by using a learned embedding and applying this to the input tokens.
- **Positional Encoding:** Unlike recurrent neural networks, transformer models contain no a priori knowledge of the order of the input sequence. To remedy this, positional encodings are added to the input embeddings before they are fed to the attention block. There are several ways of doing this, however the approach adopted in the [1] is to use apply sine and cosine functions to the position and dimension of the input vector as follows:

$$\text{PE}_{pos,2i} = sin(pos/10000^{2i/d_{model}})$$
$$\text{PE}_{pos,2i+1} = cos(pos/10000^{2i/d_{model}})$$

- **Multi-Head Attention:** Transformers use an attention block to describe the relationship between different tokens in a sequence. This takes as input a key,

query and value matrix K, Q, V respectively. The key and query matrices are multiplied together to produce a representation of how each token in the sequence relates to each other token. After normalising and applying a softmax function, the matrix is multiplied by V, which represents the semantic meaning of each input token.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

In practise, multiple attention heads are used and their outputs concatenated. This allows the model to attend to information learned from different projections W_i^Q, W_i^K and W_i^V at the same time.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- **Position-wise Fead-Forward Networks:** The output of the multi-head attention block is fed to a fully connected neural-network. This takes as input the representation at each position (making it "position-wise") and applies two linear transformations separated by a ReLU activation:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Multiple blocks consisting of the multi-head attention and feed-forward layers are stacked on top of one another to produce a deeper transformer model.

Model Variants

The encoder-decoder model introduced in [1] is not the only variant of the transformer architecture that exists. Modifications to the transformer architecture have been made in order to better support different tasks:

- **Encoder-decoder:** With encoder-decoder models, an input sequence is first embedded, then added to a positional encoding before being passed as input to a stack of attention and feed-forward network layers (the "encoder") to produce a representation of the input sequence. As the output sequence is generated, it is fed as input to a similarly structured "decoder" block. This takes an embedded output sequence, with positional encoding, as input. The output of the encoder block is fed to attention blocks in the decoder block in order to model the relationship between the input and output sequence. These models [2] tend to be used for tasks like sequence to sequence transformation, for example language translation.
- **Encoder-only:** Encoder-only models use only the encoder block of an encoder-decoder model to produce a vector representation of an input sequence. These models do not produce text directly, but are instead meant to pass their output to a downstream component for further inference, potentially for applications to sentiment analysis or named-entity recognition. Examples of these include the BERT [3] family of models.

- **Decoder-only:** Decoder-only models like those in the GPT [4] series make up the bulk of models used for auto-regressive text generation and completion. Instead of sending an input prompt to an encoder block, the prompt is instead passed in in conjuction to the output sequence and the model is trained to predict the next token in the sequence [5].

2.1.2 LLM Inference

KV Cache

Parallelism

Request Batching

2.2 Related Work

2.2.1 vLLM

PagedAttention

2.2.2 Triton

2.2.3 SGLang

2.2.4 Triton

2.2.5 llama.cpp

Chapter 3

Project Plan

Chapter 4

Evaluation Plan

4.1 Functional Requirements

4.2 Performance Metrics

Chapter 5

Ethical Issues

The principal two ethical concerns of a project in this field relate to the potential for misuse as well as provenance issues surrounding the dataset on which the model was trained.

The potential for misuse of LLMs is vast, with many instances of LLM abuse already being documented. LLM abuse typically involves the use of the model to produce harmful or misleading content. There already exist proof-of-concepts for LLMs [6] being used to generate phishing messages, with the intent to produce emails that sound more plausible and are more likely to be engaged with by a target. In addition to this, LLMs can be used to produce vast quantities misinformation or biased content that are then published to social media platforms [7]. End users may be unable to distinguish between content created by a genuine user and content generated by an LLM and thus end up misinformed.

The large size of the datasets required to train these models create potential ethical and data protection issues. Concerns exist regarding the ability for generative models to amplify existing biases in their training data, with some of these concerns borne out in cases like Microsoft's Tay chatbot [8]. AI fairness is still an open area of research [9] and it is unlikely that existing LLM models will be completely free of bias at inference time. At the same time, the provenance of this training data is also an important ethical consideration. Private or sensitive data has the potential to be incorporated into training sets and there exist cases [10] where this training data has then been generated verbatim at inference time, exposing this sensitive data to an end user.

If successful, our project broadens access to LLMs by making better use of available hardware to perform inference. This increases the viability of local inference and opens up these models to a greater proportion of hardware configurations and thus a greater number of users. While this represents a boon for the accessibility of this technology, with users no longer limited to a handful of offerings by large companies, it also increases the number of potentially malicious actors who are able to use LLMs. Small and local deployments likely have less of the oversight that large LLM providers experience, and thus are more able to misuse this technology. These two elements must be carefully managed in order to produce a project that adheres to reasonable ethical standards.

As this project represents a proof-of-concept, rather than a full-featured inference

engine, any advances made are unlikely to immediately be adopted and thus any ethical concerns are likely to be uncovered at a pace with which they can be identified early on and mitigated quickly.

Chapter 6

Bibliography

[1] Vaswani A. Attention is all you need. Advances in Neural Information Processing Systems. 2017. pages 3, 4

[2] Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of machine learning research. 2020;21(140):1-67. pages 4

[3] Kenton JDMWC, Toutanova LK. Bert: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of naacL-HLT. vol. 1. Minneapolis, Minnesota; 2019. p. 2. pages 4

[4] Radford A. Improving language understanding by generative pre-training. 2018. pages 5

[5] Dai AM, Le QV. Semi-supervised sequence learning. Advances in neural information processing systems. 2015;28. pages 5

[6] Hazell J. Spear phishing with large language models. arXiv preprint arXiv:230506972. 2023. pages 8

[7] Williams AR, Burke-Moore L, Chan RSY, Enock FE, Nanni F, Sippy T, et al. Large language models can consistently generate high-quality content for election disinformation operations. arXiv preprint arXiv:240806731. 2024. pages 8

[8] Lee P. Learning from Tay's introduction; 2016. Accessed: 2025-01-14. `https://web.archive.org/web/20241127051442/https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/`. pages 8

[9] Xivuri K, Twinomurinzi H. A systematic review of fairness in artificial intelligence algorithms. In: Responsible AI and Analytics for an Ethical and Inclusive Digitized Society: 20th IFIP WG 6.11 Conference on e-Business, e-Services and e-Society, I3E 2021, Galway, Ireland, September 1–3, 2021, Proceedings 20. Springer; 2021. p. 271-84. pages 8

[10] Nasr M, Carlini N, Hayase J, Jagielski M, Cooper AF, Ippolito D, et al. Scalable extraction of training data from (production) language models. arXiv preprint arXiv:231117035. 2023. pages 8