CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Analyzing Factors Influencing Performance in LLM Inference Systems

Master's thesis in Computer science and engineering

Ziyu Fang
Mujtaba Al Neama

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

MASTER'S THESIS 2025

Analyzing Factors Influencing Performance in LLM Inference Systems

Ziyu Fang

Mujtaba Al Neama

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Analyzing Factors Influencing Performance in LLM Inference Systems

Ziyu Fang, Mujtaba Al Neama

Supervisor: Huaifeng Zhang, Department of Computer Science and Engineering
Examiner: Ahmed Ali-Eldin Hassan, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in $\LaTeX$
Gothenburg, Sweden 2025

Analyzing Factors Influencing Performance in LLM Inference Systems

Ziyu Fang, Mujtaba Al Neama
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Large Language Models (LLMs) lead to numerous innovative applications, including virtual assistants, content generation, and recommendation systems, which made a revolution in daily life. LLM inference is crucial as it allows these models to process and generate human-like text in real-time, making them adaptable to a wide range of practical applications. This capability has not only enhanced the functionality of AI-driven technologies but also expanded their accessibility and impact across various industries. Latency and throughput are essential metrics for evaluating the performance of LLMs, as they directly influence user experience and system efficiency. They are critical for optimizing the deployment and operation of LLM-based solutions in real-world scenarios. In this thesis, we evaluate the performance of LLM inference by analyzing how different factors affect LLM inference throughput and latency. We aim to study the current performance and internal working mechanism of Llama 3 and GPT-2 to explore potential methods that could keep improving LLM inferences. The approach of this project includes 2 steps which are data collection and data analysis. Our results show that increasing the batch size significantly improves throughput but can also lead to higher latency, indicating a trade-off between speed and responsiveness in LLM inference. Additionally, a larger model size can provide more accurate outputs. Interestingly, allocating more GPU resources reduced overall GPU utilization for both Llama 3 and GPT-2. Inefficiencies in resource allocation could negatively impact the cost-effectiveness of deploying LLM inference's performance.

Keywords: LLM Inference, Latency, Throughput, Transformer Model, GPU, Parallelism Computation, Empirical Analysis.

Acknowledgements

We would like to express our great appreciation to Huaifeng and Ahmed for giving us this project. In particular, we would like to thank Huaifeng for all the support and patient guidance to our thesis, and we thank Ahmed for your great advice on our project. Especially, Ziyu expresses her thanks to Filip, Robin, Gabriel, Edvard, and Björn for being accompanied with her to go through all the difficulties in this thesis work. In the end, thanks to everything that happened in our thesis work; for equipping us with the bravery to face future challenges.

Ziyu Fang, Mujtaba Al Neama, Gothenburg, 2025-04-23

Contents

List of Figures

List of Tables

1

Introduction

With the development of Artificial Intelligence (AI), a lot of new applications like virtual assistants, content generation, and recommendation systems have risen to the market. One of the key drivers behind these innovations is the development of Large Language Models (LLMs). LLMs enable AI systems to understand and generate human-like text. LLM inference is the process of generating output based on input data. Unlike training, which is typically a one-time, resource-intensive process, inference happens every time a user interacts with the model. As a result, LLM inference could be computationally expensive and resource-intensive[1]. Efficient inference is therefore crucial for reducing cost, enhancing user experience. This thesis would like to get a greater insight into the performance and internal working mechanism of LLM's inference systems, allowing us to explore the potential method that could keep improving LLM's performance.

In this chapter, we will briefly introduce our work starting with the goals and challenges of the project. Followed by the scope of the thesis and the approach we use during the project. Finally, we introduce our thesis outline.

1.1 Goals and Challenges

The main goal of our thesis is to evaluate the performance of LLM inference systems by analyzing how key factors affect LLM inference throughput and latency. Specifically, we are basing our analysis on GPT-2 [2] and Llama 3 [3], two of the most popular and widely used LLMs.

Latency and throughput are key metrics for evaluating LLMs because they directly affect user experience and system efficiency. Latency measures the time taken to process and respond to individual requests, which is crucial for real-time applications to ensure quick and responsive interactions [4]. Throughput gauges the number of requests the model can handle per unit of time, reflecting the system's ability to scale and manage high demand efficiently, thus influencing cost-effectiveness [5] [6]. Therefore, by evaluating these metrics, we ensure that we better understand LLM inference performance requirements for practical deployments.

The list below includes the challenges we faced in this project:

- The first challenge is how to design an efficient data collection pipeline, that collects sufficient useful data for analysis.

- Designing the analysis framework is another challenge. It is important to correctly understand how each factors influence LLM inference performance.
- The deployment of LLM inference is often constrained by GPU resource limitations. Therefore, we need to carefully determine how many GPUs we can allocate to each instance of GPT-2 and Llama 3.
- Accurately determining input length using tokenizers is a challenge, as it involves understanding how different tokenization strategies affect sequence length, which is critical for optimizing model performance and managing computational resources.
- The maximum sequence length limitation in LLM poses a challenge, as it restricts the model's ability to process and generate long sequences.

1.2 Scope of this Thesis

In our project, we focus on the LLM inference workloads. We select two models with different parameter sizes to perform the analysis: GPT-2 and Llama 3. The parameter size of GPT-2 and Llama 3 is 137 million and 70 billion, respectively. The factors influencing inference performance are vast. To limit the scope of this thesis, we will focus on 5 key parameters: batch size, input length, output length, model size, and number of GPUs. We will give a detailed description of these 5 contributing factors.

Batch Size specifies the number of input and output sequences in the LLMs. Larger batch sizes may increase latency for individual requests because the model processes more input requests simultaneously, leading to longer wait times. However, large batch sizes should improve throughput as they enable more GPU parallelism, allowing more data to be processed per unit of time [4]. Therefore, batch size is one of the key factors to evaluate LLM inference scalability and efficiency.

Input Length means how many tokens are in each input sequence. Longer input sequences usually require more computation and memory, thus may increase latency. Additionally, longer input lengths could reduce throughput as they consume more resources and limit the number of concurrent requests the system can handle effectively.

Output Length means how many tokens are generated for each output sequence. Same as input length, generating longer outputs means more time and computational power, leading to higher latency. This increased demand may reduce throughput because each request takes longer to process. Although the input length and output length will influence latency and throughput, the interesting thing is to which degree they affect the LLM inference processing.

Model Size is the number of parameters that an LLM contains. A larger model size means the LLM could work better in more complex languages and deal with nuanced prompts [7]. Therefore, comparing the model size supports a practical method to weight LLM performance.

Number of GPUs is important for finding the best GPU utilization for LLM inference. Effective GPU utilization may reduce latency while increasing the number of GPUs can improve throughput by distributing the computational load. However, a larger number of GPUs does not lead to effective GPU utilization. So, it is meaningful to take the number of GPUs as one of the key factors and analyze how much GPU utilization influences latency and throughput.

1.3 Approach

Our project involves two main steps: data collection and data analysis. In the data collection step, we design a collection pipeline and collect two comprehensive datasets of GPT-2 and Llama 3 through varying batch sizes, input lengths, output lengths, and number of GPUs. In the data analysis step, we perform extensive analysis on the data collected from the previous step, to explore relations between latency and throughput.

We conducted experiments on the Berzelius computer cluster [8] which can provide computational power of up to 752 NVIDIA A100 GPUs. Furthermore, Berzelius enables non-blocking connections between GPUs featuring 200 GB/s bandwidth and microsecond latency. It is crucial to provide stable and sufficient computation resources.

1.4 Outline

The rest of the thesis organized as follows. Chapter 2 describes the development of LLMs, an introduction to transformer model, an overview of a state-of-the-art LLM-inference framework, vLLM, and the concept of tokenizer. Chapter 3 presents approaches used in our research. The analysis results are presented in Chapter 4. The research conclusion, discussion, optimization, and future work are given in Chapter 5.

2

Background

In this chapter, we provide an overview of the key concepts and components that are fundamental to understanding the performance of Large Language Model (LLM) inference systems. The chapter begins with the preliminary knowledge overviewing AI as a whole. Further, the development of LLM is introduced. This is followed by a discussion of Transformer Models including the model architecture and attention mechanisms.

What's more, we delve into the role of the tokenizer and parallelism techniques. We also explore vLLM, a high-throughput distributed LLM inference that integrates with models seamlessly. Together, these sections provide a comprehensive foundation for analyzing the factors that affect performance in LLM inference systems.

2.1 Machine Learning Primer

Artificial Intelligence (AI) is the simulation of human intelligence processes by machines, particularly computer systems. Today, AI has become an integral part of our daily lives, seamlessly embedded in various technologies and services. Hopefully, our project can help AI continue to evolve, driving cutting-edge research and innovation, and exploring new methodologies to push the boundaries of what is possible, including advances in (Natural Language Processing) NLP, autonomous systems, and ethical practices.

Mitchell et al. [9] introduce machine learning (ML) as a subfield of AI that focuses on the development of algorithms and the usage of data to enable computer systems to learn in a human way. It can effectively improve systems by facilitating the ability to make predictions or decisions based on large amounts of data and new information [9]. Different from traditional programming tasks that are coded by developers carefully, machine learning leverages data to create models that can automatically generalize from past examples to handle new data with better accuracy. Machine learning can be categorized into 4 main types which are supervised learning [10], unsupervised learning [11], semi-supervised learning [12], and reinforcement learning [13]. Supervised learning is defined as training models on a labeled dataset, which could make predictions by learning inputs and outputs [10]. Unsupervised learning, on the other hand, uses unlabeled data to discover hidden patterns or structures within the data through ways like clusters, dimensionality reduction, visualization, and so on [11]. Semi-supervised learning is a hybrid method that utilizes both labeled and unlabeled data. It learns from the small amount of labeled data and uses the amount of unlabeled data to improve the learning progress [12].

Natural Language Processing (NLP) is a subfield of artificial intelligence that focuses on the interaction between computers and human (natural) languages. The goal of NLP is to enable machines to understand, interpret, and generate human language that is both meaningful and useful [14].

A LLM is a type of artificial intelligence model designed to understand and generate human-like text by leveraging vast amounts of data and advanced neural network architectures. These models are trained on extensive corpora, enabling them to perform a wide range of language-related tasks, such as translation, summarization, and conversation.

2.2 Development of Large Language Models

Large Language Models (LLMs) are a branch of language models known for their size and power. They leverage large amounts of data and computational power to achieve state-of-the-art performance on a variety of NLP tasks. In this section, we will introduce the LLMs development progress.

2.2.1 Statistical Methods

Natural language processing (NLP) had significant advancements through statistical methods like Hidden Markov Models (HMMs) and N-gram models [15] [16]. These approaches relied on mathematical models to understand and generate human language.

Hidden Markov Models (HMMs) became widely used for various NLP tasks such as part-of-speech tagging, speech recognition, and named entity recognition. HMMs provided a framework for dealing with sequential data by modeling the probabilities of sequences of observed and hidden states. For example, in the context of language, the observed states might be words or phonemes, and the hidden states could be part-of-speech tags or phonetic features. By capturing the probabilistic dependencies between these sequences, HMMs allowed for more accurate and efficient tagging and recognition systems [15]

N-gram models also emerged as a staple in language modeling at the same time. It aimed to predict the next word in a sequence based on the previous n-1 words, capturing the contextual dependencies between words. N-grams provided a simple but effective way to model language, enabling improvements in various applications like text prediction and machine translation which have been explained by Brown et al.[16].

Another crucial development for statistical methods is Latent Dirichlet Allocation (LDA). LDA is a generative probabilistic model used for topic modeling. It classifies documents into topics based on word distributions, assuming that documents are mixtures of topics and each topic is a distribution over words. LDA provided valuable insights into the thematic structure of large text corpora, enhancing information retrieval and content recommendation [17].

However statistical methods have limitations of restricted context understanding, data sparsity, high computational costs, and insufficient semantic depth. Therefore, it is hard to capture long-range dependencies and handle nuanced meanings in language, as Beran et al.introduced in their paper [18]. Consequently, the need for more sophisticated approaches led to the development of neural networks.

2.2.2 Neural Networks

Neural networks capture more complex patterns and dependencies in text [15], [16], which effectively improves the drawback of statistical methods. With the capacity to learn non-linear relationships, neural networks address the limitations of statistical methods by learning rich, distributed representations of words and contexts. It effectively handles long-range dependencies, reduces data sparsity issues through embeddings, and captures complex semantic relationships, thus providing more robust and scalable solutions for modern NLP tasks.

Feedforward Neural Networks (FNNs) were among the earliest neural architectures employed for NLP tasks. FNNs, consisting of layers where each layer's output is fed into the next, was used for classification and regression tasks. However, FNN had a problem with sequential data due to their lack of temporal context[19]. The introduction of Recurrent Neural Networks (RNNs) marked a significant advancement. RNNs are designed to handle sequential data by incorporating feedback connections, allowing them to maintain a form of memory and capture temporal dependencies. This made them more suitable for tasks involving sequences of words, such as language modeling and sequence prediction. Despite their potential, traditional RNNs faced challenges with long-term dependencies due to issues like the vanishing gradient problem [20].

Long Short-Term Memory (LSTM) networks are one of the most significant breakthrough developments of neural networks, published in 1997 [21]. LSTMs address the limitations of traditional RNNs by incorporating gating mechanisms that control the flow of information and maintain long-term dependencies. This enhancement allowed LSTMs to remember information over longer sequences, significantly improving their performance in tasks such as machine translation and speech recognition.

2.2.3 Transformer Model Era

The Transformer model[22] addresses several shortcomings of previous models such as FNNS, RNNs, and LSTM networks. The Transformer model relies on attention mechanisms and discards the use of recurrence entirely.

The transformer model has a significant impact on various NLP tasks. In 2018, Devlin et al[23]. introduced Bidirectional Encoder Representations from Transformers (BERT), which is designed to pre-train deep bidirectional representations from an unlabeled text by joint conditioning on both left and right context in all layers

[23]. In 2019, the Generative Pre-trained Transformer (GPT) series by OpenAI was published. The GPT began with GPT-1 in 2018 and was followed by GPT-2 in 2019, demonstrating the effectiveness of large-scale unsupervised pre-training. GPT-3, in particular, highlighted the capabilities of very large models in generating coherent and contextually relevant text, showcasing advanced performance in numerous language tasks[24][1].

2.2.4 Open-Source Large Language Models

Open-Source Large Language Models profoundly impacted the field of NLP by making advanced AI technologies available to a wide audience. Models like GPT-Neo [25], GPT-J [26], and GPT-NeoX [27] by EleutherAI made cutting-edge NLP capabilities available to researchers, developers, and organizations without the high costs. These models foster collaboration within the AI community, enabling researchers to build upon existing architectures, share improvements, and drive innovation more rapidly. Additionally, open-source models such as Large Language Model Meta AI (LLaMA) [28], and BLOOM[29] provide transparency in terms of architecture and methodologies, which is crucial for understanding model behavior, debugging, and analysis. What's more, Llama published by Meta, supports a wide range of applications, including text generation, comprehension, and translation, while being more accessible and adaptable compared to previous models[28]. Therefore, we pick Llama 3 to study its performance.

Hugging Face [30] is a collaboration platform that hosts models, datasets, and applications. Hugging Face is best known for its Transformers library, which includes implementations of numerous transformer-based models for various NLP tasks [31]. This library, along with their Datasets and Tokenizers libraries, has become a cornerstone for researchers and developers working with state-of-the-art NLP models. Hugging Face's Model Hub facilitates the sharing of pre-trained models, fostering a collaborative environment

2.3 Transformer Model Basics

The transformer model was first described in the paper "Attention is all you need" by Vaswani et al.[22] which can be seen as a landmark of modern artificial intelligence. In this section, we will introduce the transformer model from the perspectives of model architecture, attention mechanism, encoder and decoder.

2.3.1 Model Architecture

The transformer model is built upon a sequence-to-sequence architecture. The sequence-to-sequence architecture is the mechanism that takes an input sequence with processing and generates an output sequence. The transformer model also follows an encoder-decoder structure based on sequence-to-sequence. The encoder-decoder structure consists of 2 parts which are the encoder and decoder. The encoder processes an input sequence into a difference representation, which the decoder will use to generate the desired output sequence. Based on this overall sequence model architecture, the transformer model itself is built entirely on self-attention mechanisms and point-wise, fully connected layers for both the encoder and decoder, see Figure 2.1. therefore, it enables more efficient parallelization and improved performance on a range of tasks.

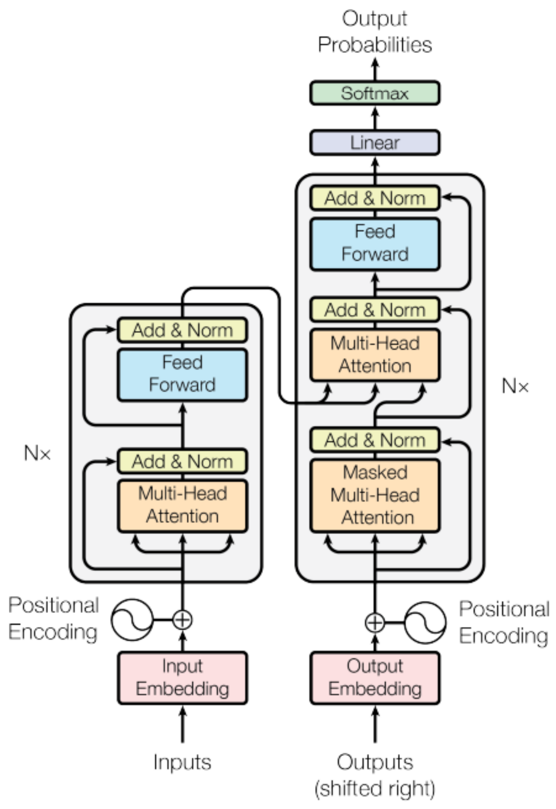


Figure 2.1: Transformer Architecture [22]

2.3.2 Attention Mechanism

Attention is a mechanism that allows the model to focus on matching different parts of the input sequence when producing each output element from Vaswani et al.[22]. The attention mechanism enables the model to weigh the sum of the different values for the output, with the weight for each value determined by a compatibility function applied to the query and the corresponding key. Generally, there are 2 key aspects of attention in the transformer model which are the Scaled Dot-Product Attention and the Multi-Head Attention.

Scaled Dot-Product Attention is the core operation of the attention mechanism. There are 3 metrics in this formula which are Query Matrix Q, Key Matrix K, and Value Matrix V.

- Query Matrix Q represents the current token for which attention is being computed. It determines which parts of the input sequence should be focused on.

- Key Matrix K represents all elements in the sequence. It is used to determine the relevance to the query.
- Value Matrix V contains the value that models are trying to attend in.

It works by first taking the dot product of a query vector (Q) with key vectors (K) to measure their similarity. Then this attention score for each pair of query and key is calculated using the dot product of the query and key, scaled by the inverse square root of the dimension of the keys $\sqrt{d_k}$ to ensure stable gradients during training.

The scaled scores are passed through a softmax function to convert them into a probability distribution, which represents the attention weights. Finally, the output is calculated as a weighted sum of value vectors (V), using these attention weights. This process enables the model to selectively attend to important information in the input sequence, facilitating better handling of dependencies regardless of their distance. The figure 2.2 is a concrete representation of the formula as follow:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

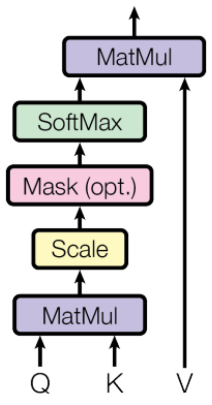


Figure 2.2: Scaled Dot-Product Attention [22]

Multi-Head Attention focuses on different parts of the input sequence simultaneously, instead of performing a single attention function with d_{model}-dimensional keys, values, and queries. The mechanism of Multi-Head Attention can be intuitively visualized by figure 2.3. By performing the attention operation multiple times in parallel with different sets of queries, keys, and values, known as "heads", multi-head attention significantly enhances its ability. Instead of computing a single set of attention scores, Multi-Head Attention runs several attention mechanisms (or heads) in parallel. Each head has its own set of learned linear projections for Q, K, and V.

Each attention head computes scaled dot-product attention independently. This means each head focuses on different parts of the sequence or different aspects of the relationships between tokens. The outputs of all attention heads are concatenated. In the end, the concatenated output is then passed through a final linear layer to produce the final output of the multi-head attention mechanism, the formula can be shown as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W_O \tag{2.2}$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.3}$$

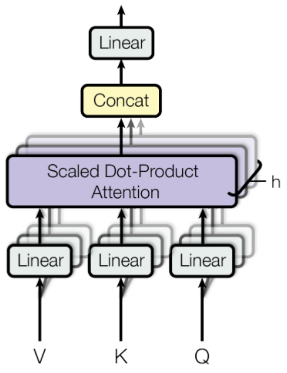


Figure 2.3: Multi-Head Attention [22]

2.3.3 Encoder & Decoder

Encoder: As figure 2.1 shown, the encoder processes the input sequence and is composed of multiple identical layers. Each layer consists of two primary sub-layers: The first one is a multi-head self-attention mechanism, and the second is a feed-forward neural network. Each of the two sub-layers is followed by layer normalization and has a residual connection to increase training stability and efficiency. Vaswani et al.[22] use six encoders. The multi-head self-attention mechanism allows the encoder to attend to different parts of the input sequence to capture dependencies between words, irrespective of their position in the sentence. And for the sub-layer feed-forward neural network refines the representation of each word after self-attention. It consists of two linear transformations with a ReLU activation in between.

Decoder: As the right side of transformer architecture (see figure 2.1), it is composed by Masked Multi-Head Self-Attention, Feed-Forward Neural Network, and Encoder-Decoder Attention. The decoder also begins with a self-attention mechanism, but with a crucial difference: it is masked. The Masked Multi-Head Self-Attention sub-layer ensures that each position in the output sequence can only attend to earlier positions, maintaining the auto-regressive nature of the decoder. This prevents the model from cheating by looking at future words during training, ensuring that the generation of the output sequence is auto-regressive. Then the sub-layer Encoder-Decoder Attention enables the decoder to attend to the entire input sequence by incorporating the output of the encoder, allowing it to gather context necessary for generating the next word in the output sequence.

2.4 Inference

In our project, we are focusing on analyzing the contributing factors (batch size, input length, output length, model size, and number of GPUs) and how they influence the latency and throughput in GPT-2 and Llama 3 inference systems. The goal of inference is to use a trained model to make predictions or generate outputs based on a new prompt. The trained LLMs can take the input sequences, then process them and produce output sequences or predictions based on the patterns and relationships model learned. It is widely used in image classification, face identification, trend prediction, and so on.

2.4.1 Parallelism Techniques

Given the large size of LLMs, several parallelism techniques can be used to accelerate the training of the inference process. Existing methods to parallelize computation of LLMs include data parallelism, tensor parallelism, and pipeline parallelism.

Data Parallelism The input data of a model is usually arranged in batches. In data parallelism, the batched input data is split into several sub-batches, each sub-batch allocated to a device. Each device holds a full copy of the model and computes the output for the sub-batch. Data parallelism is widely used due to its simplicity [32].

Pipeline Parallelism is a computation technique which divides the layers of the Deep Neural Network (DNN) model into multiple consecutive stages [33], see Figure 2.5. Each stage is allocated to a single GPU performing forward pass and backward pass. In forward pass, each stage passes an output activation to the next stage asynchronously. Meanwhile, The last stage begins with the backward pass as soon as the forward pass is finished. Similar to the forward pass, the backward pass will transfer the input back to the previous stage. One of the significant advantages of pipeline parallelism is that GPU computation happens simultaneously, which could increase the throughput.

Tensor Parallelism involves splitting model tensors, which include model weights (parameters size in each model), gradients (the rate of change of loss with respect to the weights), and optimizer states (additional variables maintained by the optimizer to enhance the training process), into multiple slices across different GPUs [34]. Instead of processing the entire set of weights on a single GPU like pipeline parallelism, tensor parallelism devides individual weights [35]. The methodology of tensor parallelism can be shown as Figure 2.4. It involves distributed computation of specific operations, modules, or layers of the model, ensuring the correctness of the computation while enhancing computational efficiency and performance. Therefore, tensor parallelism is particularly useful for the models that a single parameter consumes most of the GPU memory.

2.4.2 vLLM

vLLM is a state-of-the-art open-source library designed to enhance the efficiency of LLM inferencing and model serving, built by Kwon et al. [37]. This library is

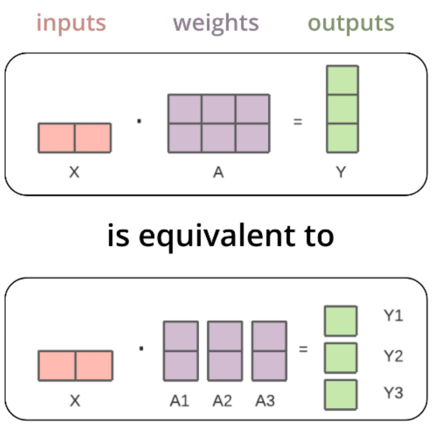


Figure 2.4: Tensor Parallelism [36]

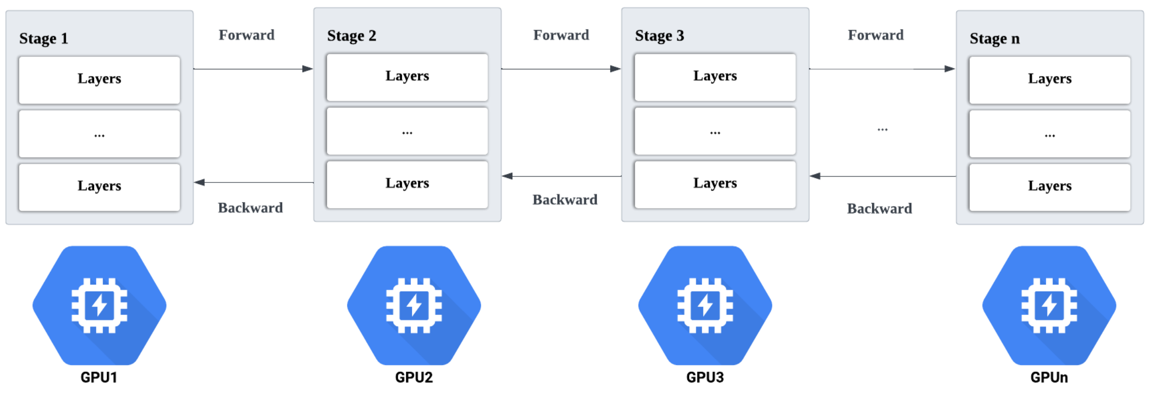


Figure 2.5: Pipeline Parallelism

notable for its advanced serving throughput, achieved through innovative features like PagedAttention—an attention mechanism inspired by the classical virtual memory and paging techniques from operating systems [37]. PagedAttention allows vLLM to manage large-scale models and datasets more efficiently, optimizing both memory usage and computational performance.

A key strength of vLLM is its ability to handle continuous batching of incoming requests, significantly boosting throughput and reducing latency. The library includes optimized CUDA kernels that maximize performance on NVIDIA GPUs, ensuring that computations are both fast and efficient. Furthermore, vLLM is designed with flexibility in mind, supporting tensor parallelism and pipeline parallelism, which enables seamless distributed inference across multiple GPUs, including both NVIDIA and AMD hardware.

In addition to its technical prowess, vLLM offers seamless integration with popular models from Hugging Face, making it easy for developers to deploy and serve a wide range of pre-trained models. The library also supports various high-throughput decoding algorithms, such as parallel sampling and beam search, which are essential for delivering quick and accurate results. This combination of advanced features and ease of use makes vLLM an ideal choice for deploying large language models in diverse environments, from research to production.

2.4.3 Tokenizer

Tokenizer is a tool to break down text into smaller units, such as words, subwords, or characters. This process is known as tokenization. Tokenization takes an important step in NLP for transferring original text into a specific format so that machine learning algorithms can easily process it. It is essential for preparing textual data for computational processing by machine learning models, particularly in tasks like language modeling, translation, and sentiment analysis.

Sampling Parameters is used for text generation. **max_tokens** controls the maximum number of tokens for the output sequence. **min_tokens** limits the minimum number of tokens for the output sequence. Setting up both **max_tokens** and **min_tokens** gives us the desired output sequence length.

RequestMetrics class is associated with the request. **first_token_time** is used for measuring the time when the first token was generated. By setting **first_token_time**, we got the time that could show how fast the model responses.

3
Methods

This chapter goes through our approaches and experiments environment that we utilized in the project implementation.

3.1 Approach Overview

Our approach involves two stages: data collection and data analysis. Data collection is our first stage. By varying batch sizes, input lengths, output lengths, and number of GPUs, we aim to collect comprehensive datasets which will be critical for us to understand how these factors influence model performance and resource utilization. The next step is data analysis based on our datasets, where we explore patterns and correlations to identify optimal settings that balance performance and resource consumption. The results gained from this analysis will inform best practices for deploying language models in diverse applications, and come up with some potential optimization.

3.2 Data Collections

The first stage of our project is data collection for evaluating the performance of a language model, GPT-2 and Llama 3, using the vLLM library. The key steps in this workflow include measuring latency, calculating throughput, and monitoring GPU utilization. An overview of the data collection workflow can be seen as Figure 3.1

1. **Define Parameters:** Set up a test value for each of the parameters `batch sizes`, `input lengths`, `model size`, and `output lengths` to determine the different configurations to be tested.
2. **Initialize the LLM:** Load the GPT-2 model and Llama 3 model using `vllm` with setting multiple GPUs for tensor operations specific for parallel computation.
3. **Batch Generation:** Generates different combinations of `batch sizes`, `input lengths`, and `output lengths`.
4. **LLM Work Mechanism:** Processes each batch configuration to LLM, see Figure 3.2 for how LLM inference processes each batch of request.
5. **Metrics Computation:**

- **Measure Latency:** Measure the latency, including the `Time to First Token` (TTFT) and `Time to Last Token` (TTLT), for a given input sequence and batch size.
- **Measure Throughput:** Measure `throughput` and `normalized latency`.
- **Measure GPU Utilization:** Retrieves the `GPU utilization` details, including `memory usage` and `GPU utilization`.

6. **Write Results to CSV:** Open a CSV file to save the results. Iterate over the generated batches, measuring performance metrics for each configuration. Capture GPU usage details before and after processing each batch to determine GPU utilization. Write the collected data, including `GPU details`, `TTFT`, `TTLT`, `throughput`, and `normalized latency`, into the CSV file.

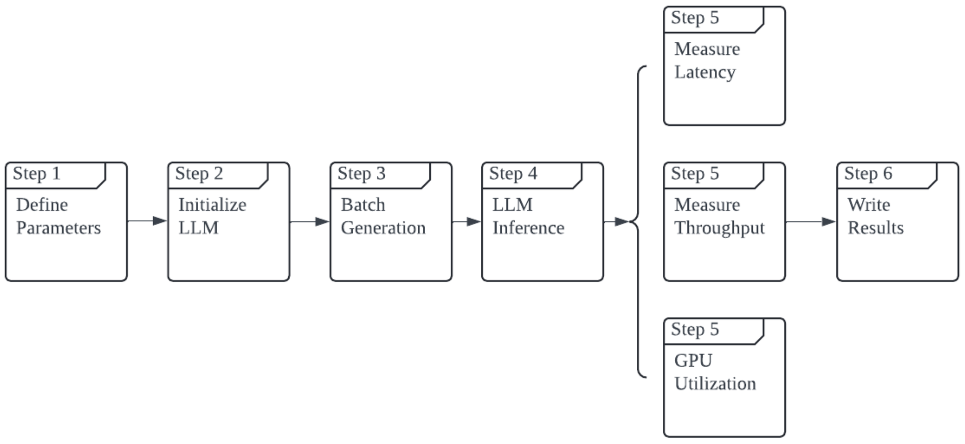


Figure 3.1: An overview of data collection workflow

3.2.1 Contributing Factors:

We focus on 5 factors: `batch size`, `input length`, `output length`, `model size`, and `number of GPUs`.

Batch Size refers to the number of samples processed together in a single pass through the model. In our project, as shown in the figure 3.2, the batch size determines how many sequences are in each batch. So the LLM will take the batch size of different input sequences simultaneously, process them together, and then create the same batch size of different output sequences. For each batch, we measure how long it takes to generate output sequences (latency) and how much data is processed per second (throughput).

Input Length: refers to the number of tokens in each input sequence that fed into the model to get generation. Using a tokenizer object from the vLLM library to convert the input text into tokens, it is ensured that the input text matches each input length in our data collection process. Tokenizers tokenize the text and truncate it to a specific number of tokens, which are then fed into the large

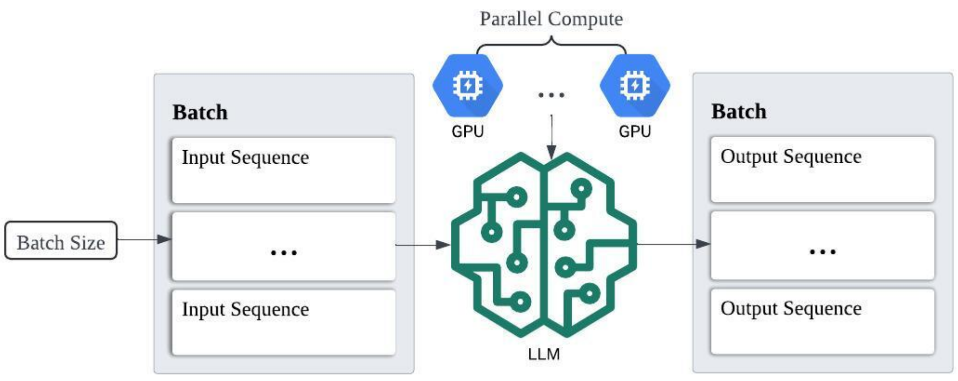


Figure 3.2: An overview of LLM working mechanism

language models. The ’tokenizer(input_text, return_tensors=’pt’, truncation=True, max_length=input_length)’ method is used in our project to convert the input text into tokens and truncate it to the specified input length. It ensures that the resulting number of tokens matches the desired input length, either by truncating longer texts or leaving shorter texts as they are.

Output Length refers to the number of tokens in each output sequence that the model is expected to generate as output in response to a given input. It determines how long the generated text should be. We set up the generation parameters structure SamplingParams from the vLLM library, which includes min_tokens and max_tokens. Therefore, LLM can generate output text as the output length we want, ensuring the generated text exactly the desired number of tokens.

Model Size for LLM is often measured by the number of parameters. A larger model size indicates more parameters, which require additional computational resources. This increased resource demand can lead to higher latency, as more time is needed to process each input, and decreased throughput, as fewer inferences can be performed in parallel. In our project, we selected GPT-2 with 124 million parameters and LLaMA 3 with 8.03 billion parameters as our research targets. Given the significant difference in model size between GPT-2 and LLaMA 3, we can explore how these variations affect performance in terms of LLM inference.

Number of GPUs is crucial for understanding how efficiently the GPU resources are being utilized during model inference. In our code, specifying the tensor parallel size (e.g., tensor_parallel_size=6) configures the model to use multiple GPUs for tensor operations, allowing parallel computation of model layers. Besides, setting the distributed_executor_backend to a specific backend (e.g., "mp") allows for distributed execution, where tasks are split and executed across multiple processors or GPUs. This parallelization enhances performance by increasing throughput and reducing latency.

3.2.2 Metrics

During our evaluation, there are 4 metrics we focused which are Time to First Token (TTFT) and Time to Last Token (TTLT) latency, normalized latency, and Throughput.

Time to first token (TTFT) is the time that it takes for the model to produce the first token of the response after receiving the input. We record the start time before sending input to the model, then subtract this start time from the timestamp when the first token is generated. For the first token timestamp, we utilized the first_token_time attribute from the RequestMetrics in the vLLM library to provide the timestamp for when the first token is generated. This gives the TTFT, indicating how quickly the model produces the first token after receiving input which is crucial for applications utilizing streaming to get immediate feedback.

Time to Last Token (TTLT) measures the overall time taken by the model to process the input sequences and generate the response. Therefore, we refer to Sheng et al.[38] latency computation methods. By considering an effective batch size of b, an input sequence length of s, and an output sequence length of n, the latency t is defined as the total number of seconds spent to process the input sequences and generate all the bn tokens.

Normalized latency is a key metric that assesses the efficiency of the model in generating output relative to the amount of data processed. In our data collection process, we utilize the normalized latency method from Kwon et al.[37] where normalized latency is calculated by dividing the total latency (TTLT) by the product of the batch size and the output length. Normalized latency is also an important parameter for evaluating throughput. Kwon et al.[37] point out that a high-throughput serving system should retain low normalized latency against high request rates.

$$\text{Normalized Latency} = \frac{\text{Total Latency (TTLT)}}{\text{Batch Size} \times \text{Output Length}}$$

Throughput refers to the amount of data or number of tasks that the LLM can process within a specific period. It is a measure of the model's efficiency and performance in handling and completing computational tasks. It usually can be described as token per second, latency inference, and queries per second. In our project, we use the Generation Throughput as our metric. Generation Throughput is defined as $\frac{bn}{t}$ to calculate how many tokens the model generates per second on average over the total duration, as proposed by Sheng et al. [38]. The batch size is b, n is the output sequence length, and t is TTLT.

3.3 Analysis Method

The second step of this project is analyzing the datasets we collected from the previous stage. By starting the correlation analysis in this stage, we get an overview of how the models' performance is affected by contributing factors. What is more, we find the most important factors for the latency and throughput through feature importance analysis. Both correlation and feature importance analysis play a solid role in the realization of the further evaluation of LLM inference performance. Therefore, this section will go through these two analysis methods.

3.3.1 Correlation Analysis

Correlation analysis is a statistical method used to measure a linear relationship between two variables. The Pearson correlation coefficient (PCC) is one of the most common measures of correlation [39]. It is a descriptive statistic, which summarizes the characteristics of a dataset. Specifically, PCC describes the strength and direction of the linear relationship between two quantitative variables [40]. The formula of the Pearson correlation coefficient is given below.

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{3.1}$$

Where r is the Pearson correlation coefficient. X_i and Y_i are individual values of variables.$\bar{X}$ and $\bar{Y}$ are mean values for each variable X and Y [41]. The formula 3.1 ensures the results of PCC will only be a scalar between -1 and 1. The value of PCC is explained in Table 3.1.

r	**Strength of correlation**
$-1.0 < r < 0.0$	Negative correlation
$0.0 < r < 0.1$	No correlation
$0.1 < r < 0.3$	Low positive correlation
$0.3 < r < 0.5$	Medium positive correlation
$0.5 < r < 0.7$	High positive correlation
$0.7 < r < 1$	Very high positive correlation

Table 3.1: Correlation Strength Based on Amount of r [41]

In this project, we calculate the Pearson correlation coefficient to quantify the relationships between key factors (Batch Size, Input Length, and Output Length) with performance metrics (TTFT, TTLT, throughput, and normalized latency).

3.3.2 Feature Importance Analysis

Feature Importance Analysis is a class of techniques to calculate all the input feature scores, which is used for determining the relative importance of each feature in datasets to make a decision [42]. Input feature scores can provide valuable insights into data relationships by understanding which features most drive model results, and guide future model development and system optimization. Usually, the feature with a higher score has a bigger influence on the model. Therefore, we aim to identify which features (output length, input length, batch size) most strongly influence the performance metrics (see Section 3.2.2) in this project, providing insight into optimizing models and improving system performance.

During this evaluation, we utilized permutation importance to calculate feature importance. Permutation feature importance is one of the methods to calculate feature importance [43]. It calculates each feature's contribution to a given dataset. For non-linear or opaque estimators, permutation feature importance is particularly useful. By breaking the relationship between the feature and the target, we could determine how much the model depends on such specific features. The permutation feature importance formula is defined as follows:

$$i_j = s - \frac{1}{K} \sum_{k=1}^{K} s_{k,j} \tag{3.2}$$

where i_j is the importance score of feature j, and s is the reference score (baseline performance) of the model (machine learning algorithm, e.g., decision tree, random forest, and linear models) on the original dataset. $s_{k,j}$ is the score of the model on the dataset where each feature j is shuffled for k times. Each k is in 1, ..., K. Therefore, formula 3.2 shows the features with higher scores are more critical to the model's performance.

3.4 Experiment Setup

In this section, we will explore the computer cluster Berzelius that we used during the data collection process (see Section 3.2). Additionally, we will also introduce the methods and tools for getting GPU usage details and utilization.

3.4.1 Hardware configuration Berzelius

In our project, the experiment utilizes the Berzelius computer cluster which can support computational power of up to 752 NVIDIA A100 GPUs. For parallel utilization of GPU computation resources, we allocate a maximum of 8 GPUs for data collection. For each GPU requested, an additional 16 CPU cores and 128 GB RAM are added. The maximum running time we allocated is 4 hours.

3.4.2 GPU Utilization

We utilize nvidia-smi as the tool to get GPU usage details including utilization. The NVIDIA System Management Interface (nvidia-smi) is a command line utility, based on the NVIDIA Management Library (NVML). It is intended to aid in the management and monitoring of NVIDIA GPU devices. It can display detailed information about the GPUs installed in a system, including utilization, temperature, memory usage, and more. By examining a list of GPU details, we can identify which GPUs are currently in use by checking if their utilization is greater than 0%. These active GPUs are then filtered into a new list so that we can count the number of used GPUs and generate a comma-separated string of their names. Additionally, it calculates the total, used, and free memory across all GPUs. Specifically, it sums up the total memory capacity, the memory currently being used, and the available memory for all GPUs in the system. This provides a comprehensive overview of GPU resource utilization, helping to monitor and manage the system effectively.

3.5 Models

In our project, we studied the performance of 2 LLMs: GPT-2 and Llama 3.

3.5.1 GPT-2

We chose GPT-2 as one of the pre-trained transformer models with 124M parameters. The training object is to predict the next word in a target input sequence so that the model can learn grammar, language patterns, and reasoning ability. It uses a mask mechanism internally as we mentioned in Chapter 2 (see Section 2.3.1) to ensure the predictions for the token i only use the inputs from 1 to i but not the future tokens. The default maximum sequence length of the GPT-2 model is 1024 tokens which means the combination of input context provided to the model and the text generated by the model in response shouldn't exceed 1024 tokens. Therefore, due to the limitation of the default maximum sequence length, we set the range of both of input and output sequence length to be 16 - 512 tokens.

The tokenizer used in GPT-2 is based on a variant of Byte Pair Encoding (BPE) [6], an algorithm designed to split a text into subword units which become the tokens of the LLM. BPE tokenization starts by treating the entire vocabulary as individual characters. It then iteratively merges the most frequent pairs of characters or character sequences into single tokens. This process continues until a predefined vocabulary size is reached. By using subword tokenization, GPT-2 can handle out-of-vocabulary words more gracefully by breaking them down into known subword units. This tokenizer approach helps balance capturing word-level and subword-level information, which is particularly useful for handling rare or complex words and effectively deals with the vast diversity of text found in natural language.

However, GPT-2 is still one of the classic standard NLP models because it is based on the transformer architecture to handle long-range dependencies in a text efficiently. It plays a critical role in the LLM's development, and this is also why we chose it as one of the targets to research.

3.5.2 Llama 3

Llama 3 is an LLM that was developed and released by Meta. Similar to GPT-2, it is also a collection of pre-trained and instruction-tuned generative text models. It is designed to handle instruction-following tasks effectively, making it suitable for applications where the model needs to understand and act upon specific user instructions. Unlike GPT-2 which only has 124M parameters, Llama 3 has two sizes which are 8 billion or 70 billion parameters. In our project, we choose the Llama 3 model with 8.03 billion parameters.

The Llama 3 is an auto-regressive language model that follows the transformer architecture. It supports a maximum sequence length that 4 times the amount of GPT-2 model. This means the model can handle up to 4096 tokens for input and output combined, ensuring that if the input text approaches this limit, the output will be correspondingly limited to fit within the total token count.

Llama 3 uses a subword tokenization method similar to the BPE which we mentioned in the GPT-2 section. This approach allows the tokenizer to efficiently handle both common and rare words by breaking them down into manageable subword components. This method enables the model to work with a large vocabulary of 128256 tokens, which allows for more detailed and nuanced representations of language compared to earlier models with smaller vocabularies.

Llama 3 is an up-to-date LLM, which has a bigger model size and maximum sequence length. Therefore, it will be an interesting model for us to research how it affects the performance compared with the GPT-2 model.

4

Results

This chapter presents the results of our experiments evaluating the performance of GPT-2 and Llama 3, using the vLLM library for inference. The data we used for evaluation is from the previous data collection stage, which varied batch size with different input and output lengths based on GPT-2 and Llama 3. The evaluation focused on understanding how factors such as batch size, input length, output length, and number of GPUs that affect key performance metrics, including latency and throughput. The results are organized into sections covering correlation analysis, feature importance analysis, and detailed examinations of latency and throughput under different conditions.

4.1 Performcance Metrics Analysis

In this section, we will dive deeper into analyzing performance metrics such as throughput, and latency (both TTFT and TTLT), and the individual impact of input variables on each metric. The results are shown in Figure 4.1.

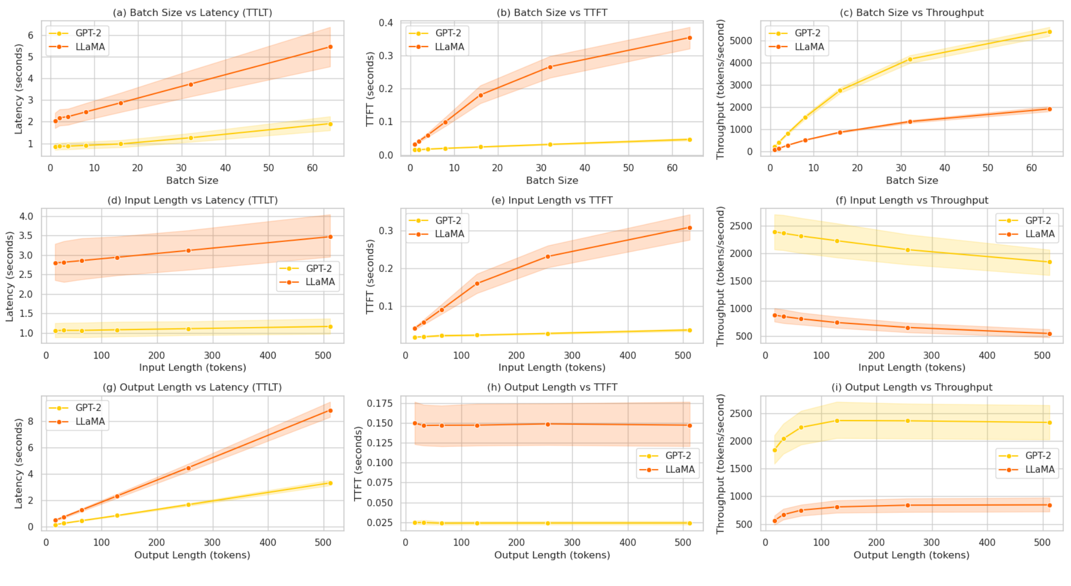


Figure 4.1: Factors Comparison between Llama 3 and GPT-2

4.1.1 Latency

Latency measures the time taken to process and respond to individual requests. We focus on two key latency metrics: TTFT and TTLT (see Section 3.2.2). This section analyzes the impact of batch size, input length, and output length on latency.

4.1.1.1 Batch Size

In the Batch Size vs TTLT Figure 4.1 (a), both GPT-2 and Llama 3 show a positive linear relationship between TTLT and batch size. However, Llama 3's TTLT rises much more steeply, indicating that GPT-2 handles larger batch sizes more efficiently with respect to TTLT. Similarly, In the Batch Size vs TTFT Figure 4.1 (b), TTFT increases with batch size for both models, but the increase is much more significant in Llama 3. GPT-2 maintains a relatively low TTFT even as batch size increases, showing it is more efficient at generating the first token with larger batch sizes. The non-linear increase in TTFT for Llama 3 suggests a fractional power model ($y = x^n$, x is batch size, y is TTLT, $0 < n < 1$) is more appropriate, while GPT-2's stable performance could be modeled with a simple linear or constant function.

4.1.1.2 Highers latency with Bigger batch size

During our analysis process, we found for the GPT-2 model, the latency increases when the number of GPUs increases for all the batch sizes, see Figure 4.4. On the contrary, for Llama 3, when the number of GPUs increases from 1 to 4, the latency decreases for all the batch sizes. However, when the number of GPUs increases from 4 to 8, the latency increases too, similarly to GPT-2. Notably, the large batch size (32 and 64) shows a higher latency increase. The bigger batch size means more input sequences and output sequences for each batch process. Combine with the factors we talked in section 4.1.1.5, this situation could also be explained from the perspectives of an increase in the computational load, memory requirements, and data transfer overheads, while also exposing inefficiencies in parallel processing and causing longer wait times for batch completion.

A larger batch size means that more data must be processed simultaneously during each inference step, which increases the **computational load**. Although it may improve overall throughput, it increases the amount of work that needs to be done in each individual step. This additional workload increases the time it takes to complete each step, leading to higher latency.

Data transfer overheads and memory constraints also contribute to higher latency with larger batch sizes. Because more data needs to be loaded from storage and transferred between the CPU and GPU, it could increase the overhead associated with data movement. Also larger batch sizes demand more memory, both on the GPU for storing inputs and intermediate calculations. The increased demand on computational resources like GPU cores can lead to resource contention, where different parts of the computation pipeline compete for the same resources, slowing down processing and increasing latency.

Last but not least, **GPU utilization** determines whether parallel processing is efficient. High GPU utilization is generally beneficial when dealing with large batch sizes because it indicates that the GPU's computational resources are being fully leveraged. However, we observed that in our project we have low GPU utilization, especially with the rise of GPU-allocated numbers. What's more, we found that for Llama 3 with the increase in batch size, the average GPU utilization decreased, see Figure 4.2. This means the GPU is not fully used efficiently for each batch process, which could cause longer processing time. It also makes GPU idling and underperformance which leads to increased latency and reduced throughput.

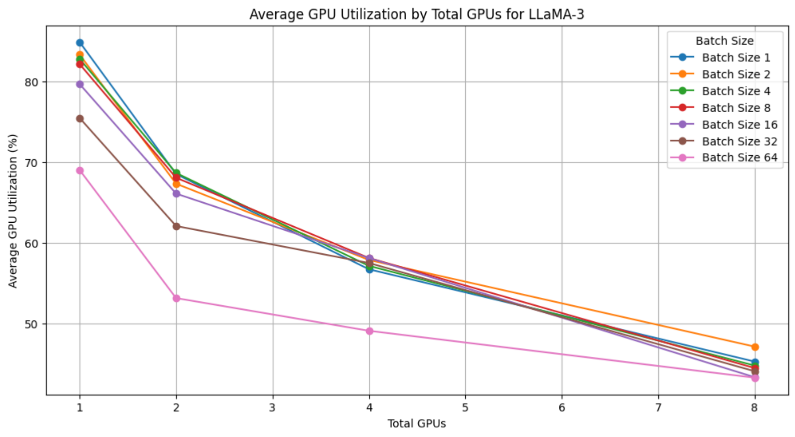


Figure 4.2: Llama 3: Average GPU utilization for each batch size

4.1.1.3 Input Length

In Input Length vs Latency (TTLT) Figure 4.1 (d), latency also increases as input length increases for both models, but Llama 3 again shows a steeper rise in latency compared to GPT-2. This means GPT-2 is more efficient at processing longer inputs, resulting in lower latency. The linear trend observed in both models supports using a linear function to describe this relationship. For Input Length vs TTFT Figure 4.1 (e), TTFT increases with input length for Llama 3 but stays almost constant for GPT-2, highlighting GPT-2's ability to maintain a stable time to the first token regardless of input length. The non-linear trend in Llama 3 suggests a fractional power model ($y = x^n$, x is input length, y is TTFT, $0 < n < 1$) for input length and TTFT, while GPT-2's consistency suggests a constant function might be best.

4.1.1.4 Output Length

Figure 4.1 (g) shows output length vs the time to last token. In the Figure, latency increases with output length for both models in a linear relationship, but Llama 3's latency rises more sharply than GPT-2's. This suggests that GPT-2 is better at handling longer outputs, maintaining lower latency. Shown in Figure 4.1 (h), TTFT remains relatively stable for GPT-2 across different output lengths, while it generally increases for Llama 3 with slight fluctuation. This indicates that GPT-2 provides more stable performance in generating the first token across varying output lengths. A linear model could describe Llama 3's slight fluctuation, while GPT-2's performance might be best represented by a constant model.

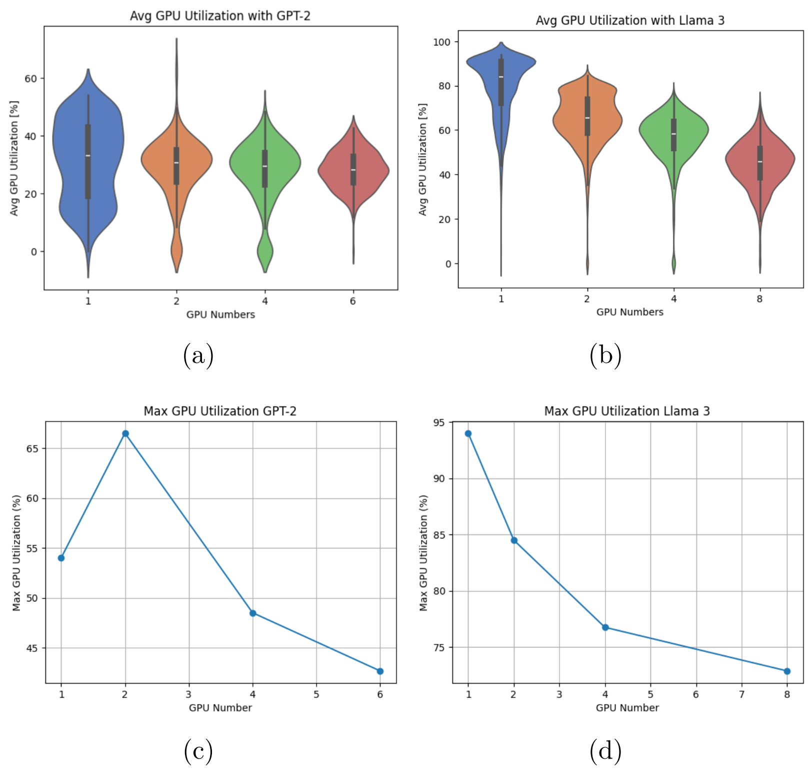


Figure 4.3: (a) GPU Utilization GPT-2 (b) GPU Utilization Llama 3 (c) Max GPU Utilization GPT-2 (d) Max GPU Utilization Llama 3

4.1.1.5 Number of GPUs with TTLT

From Figure 4.4, we found that for the GPT-2 model, increasing the number of GPUs can lead to higher TTLT. The following are the potential causes for this situation, which are synchronization and communication overheads, low GPU utilization, memory and data transfer.

Synchronization and communication overheads could be one of the primary reasons for the increase in latency while adding more GPUs during GPT-2 inference. Each input sequence data needs to be split and distributed across the GPUs, which requires inter-GPU communication via interconnects like NVLink or PCIe. After the GPUs process their portions of the data, the results need to be gathered and merged, which introduces delays. Additionally, GPUs will wait for each other to complete their tasks before moving on to the next step. This can cause some GPUs to be idle while waiting for others to catch up increasing the overall latency, and leading to synchronization bottlenecks.

Furthermore, the GPU utilization does not behave as high as we expected, (see Figure 4.3 (a) and (b)). Therefore, we analyzed the relationship between GPU number with maximum average GPU utilization. Figure 4.3 (c) and (d) show the **max GPUs' utilization decreased** with the rise of GPU numbers through our analysis, which could be another main issue that causes the increase of latency. This happens because the workload is not always perfectly divisible across all GPUs, leading to underutilization. When GPUs are not fully utilized, their computational power is wasted, resulting in slower processing and higher latency. Distributing the workload evenly across multiple GPUs can be another challenge, especially for the attention mechanisms in LLMs, which may not scale linearly.

Additionally, **memory and data transfer** could also contribute to increased latency. As more GPUs are used, moving data between them, or between the CPU and GPUs, can become a problem. If the data transfer rate cannot keep up with the processing speed, GPUs may sit idle, waiting for data, which also reduces utilization and increases latency.

4.1.1.6 Number of GPUs with TTFT

Similar to TTLT analysis, we used the same experiment to evaluate TTFT when LLM models run on different number of GPUs and processing input with different batch sizes. Figure 4.4 shows the results of this experiment.

Figure 4.4 (c) shows the TTFT by number of GPUs and different batch sizes, we notice that GPT-2 performed the best in a single GPU setup similar to TTLT and TTFT remains relatively stable on multi GPUs setup.

Figure 4.4 (F) shows the TTFT by number of GPUs and different batch sizes for Llama 3 model, we notice that the model respond faster for smaller batch sizes i.e. 1, 2, and 4 compared to larger batch sizes where the model perform the best on 4 GPUs following the same behavior of TTLT.

This analysis aligns with the correlation and feature importance findings presented later in this chapter, confirming that TTFT is significantly influenced by batch size and input length across both models.

4.1.2 Throughput

Throughput, measured in tokens per second, reflects the efficiency of the models in processing data. This section examines how batch size, input length and output length affect throughput (Figure 4.1). Understanding these factors is crucial for scaling models to handle high demand.

4.1.2.1 Batch Size

Figure 4.1 (c) illustrates the relationship between throughput and batch size. In the Figure, throughput increases with batch size for both models, but GPT-2 consistently achieves higher throughput than Llama 3 across all batch sizes, showing greater efficiency. The non-linear increase in throughput, especially the saturation effect in GPT-2, suggests that a fractional power model ($y = x^n$, x is batch size, y is throughput, $0 < n < 1$) is suitable for the relationship between throughput and batch size.

4.1.2.2 Input Length

Figure 4.1 (f) shows input length with throughput, throughput decreases as input length increases for both models, though GPT-2 maintains higher overall throughput. This indicates GPT-2 is more efficient at handling longer inputs, preserving better throughput. The non-linear decline supports using an inverse proportional function ($y = \frac{k}{x}$, $k > 0$), which the throughput decrease with the increase of input length.

4.1.2.3 Output Length

Finally, Figure 4.1 (i) shows throughput with output length, throughput decreases as output length increases for both models, with GPT-2 consistently achieving higher throughput. This suggests GPT-2 is more efficient in generating longer outputs while maintaining throughput. The non-linear decrease suggests a fractional power model ($y = x^n$, x is output length, y is throughput, $0 < n < 1$) for output length and throughput is appropriate.

4.1.2.4 Number of GPUs with Throughput

Similar to the reason that causes an increase in latency while adding more GPUs for LLMs inference, it will also cause throughput to go down like Figure 4.4. When increasing the number of GPUs for batch processes with models like GPT-2 or Llama 3, throughput can decrease due to several factors. These include increased communication and synchronization overheads between GPUs, which slow down processing; inefficiencies in load distribution, where some GPUs may be underutilized; and the overhead of managing a larger number of GPUs. Additionally, memory and data transfer bottlenecks can arise, and certain operations within the models may not parallelize efficiently, leading to diminishing returns as more GPUs are added, ultimately reducing throughput.

These findings emphasize the importance of optimizing input and output lengths and managing batch size to maintain high throughput, particularly in large-scale applications.

4.1.3 GPT-2 vs. Llama 3

Figure 4.4 shows GPT-2 inference generally exhibits 3 times lower latency and higher throughput compared to Llama 3. Here are the reasons for comparing the difference between these 2 models.

Firstly, **the size and complexity of the models** play a significant role. GPT-2 is a smaller and less complex model, with fewer parameters and layers than Llama 3. This reduced complexity means that GPT-2 requires less computational power and time to process each inference step, resulting in lower latency and the ability to handle more inferences in a given time frame, thus achieving higher throughput. In contrast, Llama 3 is a larger and more advanced model with more parameters and layers, which naturally increases the computational load and processing time, leading to higher latency and lower throughput.

Additionally, **architectural differences** between the models contribute to these performance disparities. GPT-2's architecture is simpler and more streamlined, allowing for more efficient processing during inference. While Llama 3 may incorporate advanced mechanisms to improve accuracy or other metrics, these enhancements can introduce additional computational overhead, slowing down inference and reducing throughput.

What's more, GPT-2 has been extensively optimized for inference on various hardware platforms, benefiting from techniques like pruning, quantization, and specialized inference engines that reduce computational demands. Therefore we suspect that Llama 3 might have not done the same level of optimization, leading to less efficient operation and, consequently, higher latency and lower throughput.

Moreover, **the inference paths** of the two models also differ, contributing to their performance differences. GPT-2's inference process involves a relatively straightforward computational path with fewer operations per layer and simpler mechanisms for generating outputs. This simplicity enables faster processing for each token, resulting in lower latency and higher throughput. In contrast, Llama 3's inference might involve more complex operations, which require more computation per token. These additional demands significantly increase the time required for inference, leading to the observed higher latency and lower throughput.

4.2 GPU Scaling and Performance Impact

Number of GPUs plays a critical role in the performance of LLMs, particularly in large-scale deployments (Figure 4.4). This section examines the effects of varying batch sizes and the number of GPUs on both latency and throughput.

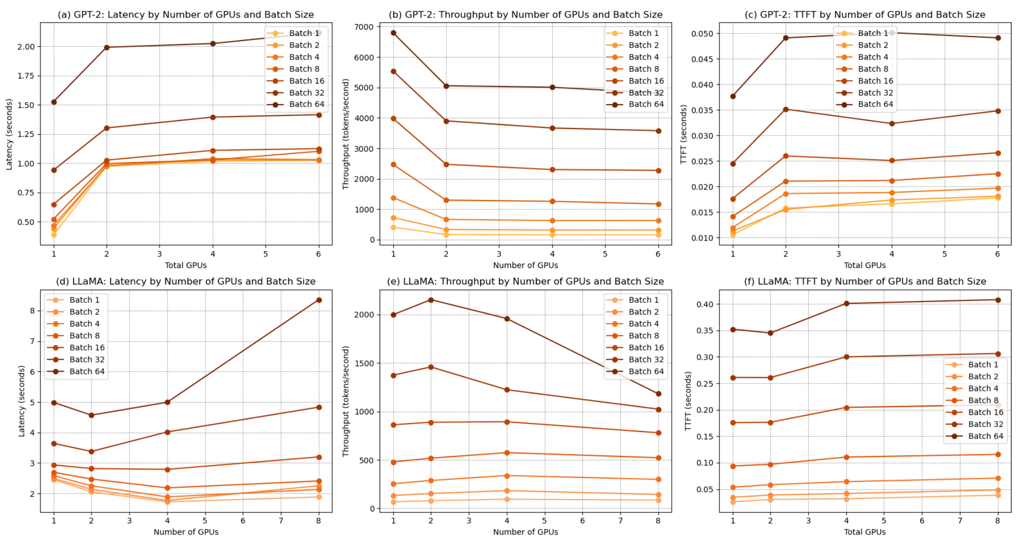


Figure 4.4: Performance Metrics with Different Batch Sizes and number of GPUs

We analyzed the datasets collected from the experiment described in the beginning of the chapter by comparing the two models' performance on different numbers of GPU setups while processing input with different batch sizes. In this experiment, batch sizes of 1, 2, 4, 8, 16, 32, and 64 were processed across different GPU configurations: 1, 2, 4, and 6 GPUs for GPT-2, and 1, 2, 4, and 8 GPUs for Llama 3.

For **GPT-2**, the highest performance was consistently observed with a single GPU. Upon transitioning to 2 GPUs, there was a significant drop in TTLT, throughput and TTFT, with smaller drops as more GPUs were added, as shwon in Figure 4.4(a), 4.4(b) and 4.4(c). This suggests that GPT-2 may face bottlenecks in parallel processing that limit its scalability when utilizing multiple GPUs.

In contrast, **Llama 3** demonstrated a different scaling behavior, as shown Figure 4.4. It performed better on 4 GPUs compared to 1 or 2 GPUs in this experiment when batch size is smaller than 32. However, for larger batch sizes (32 and 64), the model achieved better performance when running on 2 GPUs. This indicates that Llama's performance is more sensitive to both batch size and GPU configuration, likely due to differences in its architecture and parallel processing capabilities.

To further investigate impact of batch size and number of GPUs on Llama 3's performance, we conducted a detailed analysis of the Llama 3 in §4.3.1.

4.3 Further Analysis on Llama 3

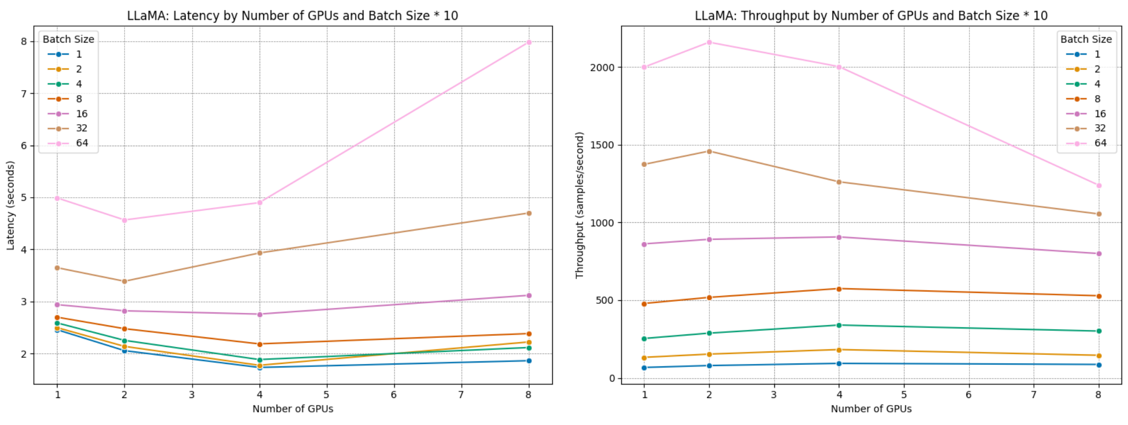
Previously, we found that Llama 3's performance was not consistent for each collection. Generally, it performs best for TTLT latency and throughput when the number of GPUs is 4 and the batch size is smaller than 32, see Figure 4.4. However, we also noticed once that Llama 3 achieved its best performance in all batch size cases while using 4 GPUs. Therefore, in this section, we set up 2 test experiments to verify the consistency of Llama 3's performance and the output quality.

4.3.1 Verifying Llama 3 performance

To validate the previous results (Figure 4.4), we collected the data 10 times and analyzed them with shaded areas. Comparing Figure 4.4 and Figure 4.5, Llama 3 using 4 GPUs reaches the general best performance is more nuanced. For smaller batch sizes (1, 2, 4, 8, and 16), the model performed better on 4 GPUs compared to the others. For larger batch sizes (32 and 64), the model achieved the best performance when running on 2 GPUs.

4.3.2 Llama 3 Output Accuracy

In this section, we evaluate the quality of output sequences generated by GPT-2 and Llama 3 as well as identify factors influencing this quality. The quality of output can be defined as the correlation between input and output. Our comparison shows that the Llama 3 model consistently outperforms GPT-2 in output quality. Though both models exhibit better coherence with shorter outputs, Llama 3's architecture supports more stable handling of longer texts compared to GPT-2.



(a) (b)

Figure 4.5: (a) Llama 3: TTLT latency with GPU numbers (b) Llama: Throughput with GPU numbers

Notably, the number of GPUs and batch size do not significantly impact output quality for either model. Increasing batch size or the number of GPUs does not lead to noticeable changes in the quality of the outputs, which remains stable.

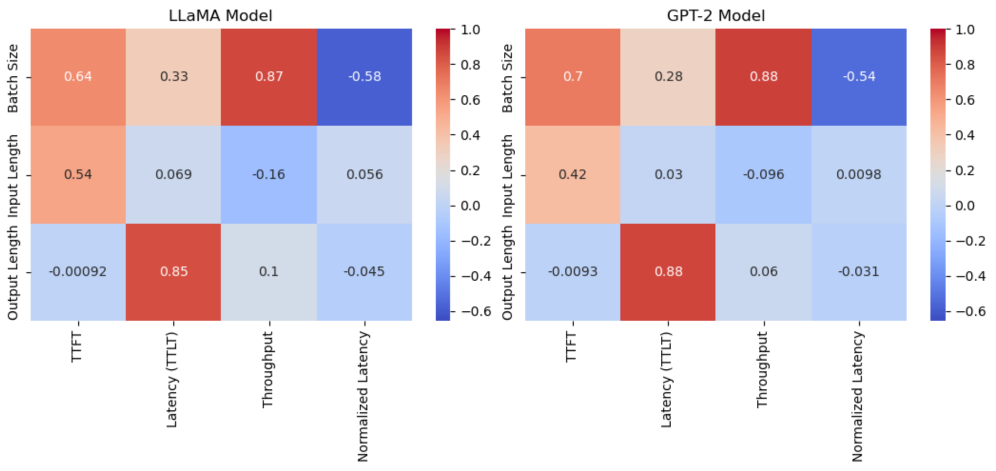
Conversely, input and output lengths have a significant effect on the models' performance. For a given output length, longer input lengths generally result in more coherent and contextually relevant outputs. However, when the input length is fixed, longer outputs tend to lose coherence, often drifting from the initial prompt and becoming less focused.

4.4 Correlation Analysis

The correlation analysis conducted for the Llama 3 and GPT-2 models provides insights into the interactions between the input variables and performance metrics. Figures 4.6 present these correlation scores (refer to Table 3.1), reflecting how each model responds to different input variables such as batch size, input length and output length.

According to the classification shown in Table 3.1, both LLM inferences show very high positive correlations ($0.7 < r < 1$) between batch size and throughput, indicating that increasing batch size can improve throughput. Similarly, latency has a strong positive correlation with output length, meaning larger output leads to higher latency. However, batch size negatively correlates with normalized latency for both models, suggesting that as batch size increases, the normalized latency tends to decrease relative to throughput.

Input Length and batch size also show high correlations ($0.5 < r < 0.7$) with TTFT, indicating that the increase in batch size or input length result in delaying the response of processing the first token. we also find low to medium correlations between batch size and latency (TTLT) around 0.3 for both models, suggesting that batch size also influences latency (TTLT) but not as much as Output length.



(a) Correlation Analysis for LLaMa

(b) Correlation Analysis for GPT-2

Figure 4.6: Correlation Matrices for GPT-2 and Llama 3 Models

Figure 4.6 illustrates the discussed correlations, providing a visual representation of how each model's performance is influenced by the input variables, aiding in the understanding of their operational dynamics.

4.5 Feature Importance Analysis

In this study, the feature importance is calculated using permutation importance. This method evaluates the importance of each feature by observing the decrease in model accuracy after permuting the feature's values. The rationale is that if shuffling the values of a feature significantly decreases the model's accuracy, then the feature is important for making predictions. Figure 4.7 shows the results of the feature importance analysis.

Throughput: As shown in Figure 4.7(a) and Figure 4.7(d), for both GPT-2 and Llama 3, batch size is the most important factor in determining throughput. In GPT-2, the number of GPUs also plays a significant role, followed by input and output length, which have a minor impact. In Llama 3, however, input and output length are more important than the number of GPUs. This indicates that Llama 3 is more affected by variations in sequence length, whereas GPT-2 is affected more by the number of GPUs.

TTLT: Figure 4.7(b) and Figure 4.7(e) show feature importance for TTLT. In the Figure, Output length is the main contributing factor influencing the time required to process the last token in both models, which is expected since longer outputs naturally take more time to generate. Batch size also affects TTLT, but to a less degree. The number of GPUs and input length have little impact, indicating that these factors are less critical once the model starts processing the input.

TTFT: Feature importance for TTFT is shown in Figure 4.7(c) and Figure 4.7(f). Batch size is again the most important factor for TTFT in both models. Input length also matters, especially in Llama 3, where longer input sequences can delay the time to generate the first token. The number of GPUs and output length have minimal influence on TTFT, suggesting that they become more relevant later in the processing rather than at the start.

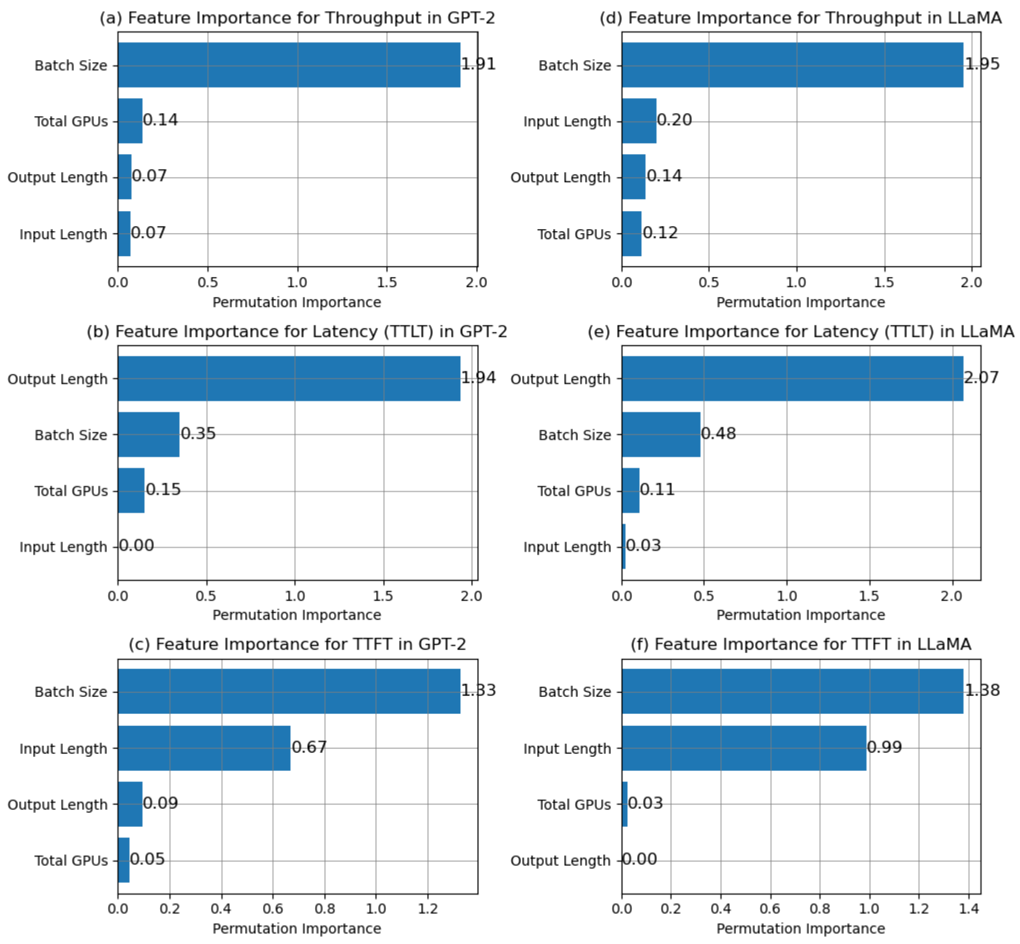


Figure 4.7: Feature Importance Analysis for GPT-2 and Llama 3

In summary, while batch size is the most influential factor across all metrics, the importance of other factors varies between GPT-2 and Llama 3. Both GPT-2 and Llama 3 are affected by adding GPUs, especially for throughput, while Llama 3's performance is more influenced by the lengths of input and output sequence length.

4.6 Summary of Key Findings

In this section, we summarize the critical observations from our performance evaluation of GPT-2 and Llama. The experiments explored how different input variables impact the models' latency and throughput. The findings highlight the strengths and limitations of each model, offering insights into their scalability and efficiency in various configurations. This summary presents the key performance trends and contrasts between GPT-2 and Llama.

Overall Performance: GPT-2 generally requires less resources than Llama 3 when processing the same input variables, resulting in higher throughput and lower latency particularly with larger batch sizes. GPT-2 also demonstrates better scalability with increasing batch sizes and additional GPUs, a key strength for real-world applications. However, both models show scalability limitations, with diminishing returns in throughput and latency as more GPUs are added or batch sizes increase.

Scalability and Resource Management: GPT-2 scales more effectively than Llama 3, particularly in throughput. Llama 3 handles longer sequences with slightly better latency but struggles with throughput scaling. Effective resource management, including careful tuning of batch sizes and GPU usage, is essential to optimize performance and manage trade-offs between latency, throughput, and GPU utilization.

5

Conclusion

This chapter presents a summary analysis of the results and interesting findings of our project. The key findings can be summarized as follows.

From correlation analysis, both the GPT-2 and Llama 3 inferences show a positive result between batch size and throughput, which showed a strong positive relationship between them. This means that each of the inferences could process more sequences per unit of time (throughput) with the increase in batch size. From the perspective of practice, it indicates that both the GPT-2 and Llama 3 inferences can handle more columns of data as the batch size rises, which shows good scalability and efficiency. However, latency also has a positive relationship with batch size, the correlation between them is approx 0.3, which is a weak positive correlation. This will cause the latency also slightly increase. In summary, the strong positive correlation with throughput and the weak positive correlation with latency suggest that both GPT-2 and Llama 3 scale well with larger batch sizes. They significantly increase throughput with only a marginal increase in latency. This is advantageous in many scenarios, as it allows for high processing efficiency without severely impacting response times.

Besides, we found the feature importance of batch size is over 1.90 for the throughput metric. It indicates that, when adjusting for different factors, the increase in batch size relative to the throughput gain can be achieved about 1.9 times. This suggests that throughput gets significant improvement by increasing batch size. At the same time, the output length's feature importance for TTLT to both of the models can be a maximum of 1.59 more than the second biggest factor batch size. It highlights that the length of the generated output sequences has a considerable impact on latency. This is because generating longer output sequences requires more computational steps, as the model needs to sequentially predict each token based on the previous context. Thus, the longer the output, the longer it takes for the model to compute and deliver the complete result. However, TTFT's batch size feature importance is 0.66 higher than the second biggest factor input length. This suggests that the model's latency for TTFT is more sensitive to the number of sequences processed at once (batch size) than to the length of the individual sequences (input length).

Compared with GPT-2, Llama 3 is more sensitive to the change of each factor. However, GPT -2 has better scalability with each factor change. What's more, high GPU utilization typically results in lower latency and higher throughput. Conversely, in our project, we observed that as the number of GPUs increased, the average GPU utilization decreased, leading to higher latency and lower throughput. It is an interesting finding, which could be mitigated effectively by reducing stored previous sequence token information in key/value(KV) [44].

5.1 Future Work

In this section, we want to deeply analyze some future optimization and work to enhance this project.

5.1.1 Inference vs. Training Analysis

In our project, we only focus on analyzing the factors that influence the performance of LLMs inference. However, this work could involve a comparative analysis of how the factors we used in our project influence the performance of LLM training. Such a study could provide deeper insights into optimizing LLM performance across different stages of model development, leading to more efficient resource use and better overall model performance. Understanding these distinctions could help in designing more effective strategies for both inference and training, ultimately enhancing the application of LLMs in various domains.

5.1.2 Increase GPU utilization

We observed that the average GPU utilization was lower than expected, which contributed to increased latency and decreased throughput during generative inference. As part of our future work, we aim to explore strategies for increasing GPU utilization to reduce latency. Jin et al. [44] designed a system with a priori knowledge of the output sequence to reduce the large memory reserved by previous sequence token information in key/value (KV) which could increase the LLM inference's GPU utilization and throughput. By finding a way to enhance GPU utilization, we anticipate not only improving throughput but also achieving lower latency, thereby optimizing the overall performance of our system.

5.1.3 Extend LLMs numbers & GPU numbers

In our current project, we utilized two LLMs as research targets, Llama 3 and GPT-2, with a maximum allocation of 8 GPUs for parallel computing. For future work, we aim to expand the scope of our analysis by incorporating more LLMs. By increasing the number of models under consideration, we intend to conduct a more comprehensive analysis of how various factors—such as GPU utilization, batch size, input length, output length, and model architecture—affect the performance of LLM inference. Additionally, by increasing the GPU numbers that could be used for parallel computation, we may explore more of the models' GPU limitations and different parallelism techniques' performance. This expanded study will provide deeper insights into optimizing inference performance across different models, helping us identify best practices for scaling and efficiency in large-scale generative inference tasks.

Bibliography

[1] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[2] O. Community, *Gpt-2 model on hugging face*, Accessed: 2024-09-16, 2023. [Online]. Available: `https://huggingface.co/openai-community/gpt2`.

[3] M. AI, *Meta llama 3 8b model on hugging face*, Accessed: 2024-09-16, 2023. [Online]. Available: `https://huggingface.co/meta-llama/Meta-Llama-3-8B`.

[4] T. Wolf, L. Debut, V. Sanh, *et al.*, *Huggingface's transformers: State-of-the-art natural language processing*, 2020. arXiv: `1910.03771 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/1910.03771`.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[6] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[7] web.dev, *Understanding large language model (llm) sizes*, `https://web.dev/articles/llm-sizes`, Accessed: Day-Month-Year, n.d.

[8] National Supercomputer Centre, *Berzelius getting started*, Linköping University, Accessed: Day-Month-Year. [Online]. Available: `https://www.nsc.liu.se/support/systems/berzelius-getting-started/`.

[9] T. M. Mitchell and T. M. Mitchell, *Machine learning.* McGraw-hill New York, 1997, vol. 1.

[10] E. Alpaydin, *Introduction to machine learning.* MIT press, 2020.

[11] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction.* Springer, 2009, vol. 2.

[12] O. Chapelle, B. Scholkopf, and A. Zien, "Semi-supervised learning. 2006," *Cambridge, Massachusettes: The MIT Press View Article*, vol. 2, p. 1, 2006.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[14] D. Jurafsky and J. H. Martin, "Speech and language processing: An introduction to speech recognition, computational linguistics and natural language processing," *Upper Saddle River, NJ: Prentice Hall*, 2008.

[15] L. Rabiner and B. Juang, "An introduction to hidden markov models," *ieee assp magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[16] P. F. Brown, V. J. Della Pietra, P. V. Desouza, J. C. Lai, and R. L. Mercer, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–480, 1992.

[17] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[18] J. Beran, "Statistical methods for data with long-range dependence," *Statistical science*, pp. 404–416, 1992.

[19] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," *Advances in neural information processing systems*, vol. 13, 2000.

[20] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[24] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[25] S. Black, L. Gao, P. Wang, C. Leahy, and S. Biderman, "Gpt-neo: Large scale autoregressive language modeling with mesh-tensorflow," *If you use this software, please cite it using these metadata*, vol. 58, no. 2, 2021.

[26] B. Wang and A. Komatsuzaki, *Gpt-j-6b: A 6 billion parameter autoregressive language model*, 2021.

[27] S. Black, S. Biderman, E. Hallahan, *et al.*, "Gpt-neox-20b: An open-source autoregressive language model," *arXiv preprint arXiv:2204.06745*, 2022.

[28] H. Touvron, T. Lavril, G. Izacard, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[29] B. Workshop, T. L. Scao, A. Fan, *et al.*, "Bloom: A 176b-parameter open-access multilingual language model," *arXiv preprint arXiv:2211.05100*, 2022.

[30] Hugging Face, *Hugging face: The ai community building the future*, `https://huggingface.co/`, Accessed: Day-Month-Year.

[31] T. Wolf, L. Debut, V. Sanh, *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[32] *Paradigms of Parallelism | Colossal-AI — colossalai.org*, `https://colossalai.org/docs/concepts/paradigms_of_parallelism/`, [Accessed 30-08-2024].

[33] D. Narayanan, A. Harlap, A. Phanishayee, *et al.*, "Pipedream: Generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM symposium on operating systems principles*, 2019, pp. 1–15.

[34] D. Narayanan, M. Shoeybi, J. Casper, *et al.*, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the In-*

ternational Conference for High Performance Computing, Networking, Storage and Analysis, 2021, pp. 1–15.

[35] Amazon Web Services, Inc., *Pytorch tensor parallelism: How it works*, *Amazon SageMaker Documentation*, `https://docs.aws.amazon.com/sagemaker/latest/dg/model-parallel-extended-features-pytorch-tensor-parallelism-how-it-works.html`, n.d. [Online]. Available: `https://docs.aws.amazon.com/sagemaker/latest/dg/model-parallel-extended-features-pytorch-tensor-parallelism-how-it-works.html`.

[36] H. Face, *Tensor parallelism*, *Text Generation Inference Documentation*, `https://huggingface.co/docs/text-generation-inference/conceptual/tensor_parallelism`, n.d. [Online]. Available: `https://huggingface.co/docs/text-generation-inference/conceptual/tensor_parallelism`.

[37] W. Kwon, Z. Li, S. Zhuang, *et al.*, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[38] Y. Sheng, L. Zheng, B. Yuan, *et al.*, "Flexgen: High-throughput generative inference of large language models with a single gpu," in *International Conference on Machine Learning*, PMLR, 2023, pp. 31 094–31 116.

[39] J. Lee Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coefficient," *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988.

[40] Scribbr, *Pearson correlation coefficient: Definition, formula & calculation*, Accessed: 2024-08-28, n.d. [Online]. Available: `https://www.scribbr.com/statistics/pearson-correlation-coefficient/`.

[41] DataTab, *Pearson correlation*, `https://datatab.net/tutorial/pearson-correlation`, Accessed: 2024-08-28.

[42] Built In, *Feature importance: What it is and how to measure it*, Accessed: 2024-08-28, 2023. [Online]. Available: `https://builtin.com/data-science/feature-importance`.

[43] Scikit-learn, *Permutation importance*, Accessed: 2024-08-28, 2024. [Online]. Available: `https://scikit-learn.org/stable/modules/permutation_importance.html`.

[44] Y. Jin, C.-F. Wu, D. Brooks, and G.-Y. Wei, "S^3: Increasing gpu utilization during generative inference for higher throughput," *Advances in Neural Information Processing Systems*, vol. 36, pp. 18 015–18 027, 2023.

A

Appendix 1