Prompt Engineering in Large Language Models

2406792

Submitted for the Degree of Master of Science in

Artificial Intelligence

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

September 5, 2024

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 11453

Candidate Number: 2406792

Date of Submission: September 5, 2024

Signature:

Abstract

This project explores the impact of various prompt engineering techniques on the performance of large language models (LLMs), with a specific focus on arithmetic tasks using the GSM8K dataset. Prompt engineering, which involves carefully crafting inputs to guide LLMs towards producing accurate and relevant outputs, is crucial in optimizing the capabilities of these models. The study implemented and compared three key prompting methods: Zero-Shot, Chain of Thought, and Least to Most. Zero-Shot prompting served as a baseline, while Chain of Thought prompting encouraged models to break down complex problems into logical steps, and Least to Most prompting guided models to solve subproblems sequentially. The research faced computational challenges due to the limitations of local hardware, which were overcome by deploying an external AI server via Docker and tunneling through ngrok. The study concludes with suggestions for future research, including the exploration of diverse datasets, integration of Few-Shot Learning, automation of prompt optimization, and cross-model comparisons. This research underscores the importance of prompt engineering in leveraging the full potential of LLMs and provides a foundation for further advancements in this field.

Contents

1 Introduction

Large language models (LLMs) have become very useful in recent years in the area of natural language processing. They make it possible for machines to understand and write text that sounds very much like it was written by a person. Systems like GPT-3, LLAMA, and Falcon are good examples of these models. They have the potential to change industries by automating chores that used to require human cognitive abilities, like talking to customers, making content, and analyzing data.[16]

In order to get the most out of LLMs, though, it's important to come up with good ways to connect with them. This is called "prompt engineering." Carefully planning input prompts to help LLMs produce the most useful and correct outputs is what prompt engineering is all about. As LLMs get more complex, it gets harder to come up with good prompts. This means that we need a methodical way to test and improve these methods. The main goal of this project is to improve LLM performance across a wide range of jobs and datasets by implementing and testing different prompt engineering methods.

1.1 Aim

The primary aim of the project on Prompt Engineering in Large Language Models is to implement various prompt engineering methods on large language models (LLMs) and evaluate their performance on a dataset. This involves selecting a suitable LLM, setting it up for testing, and developing a proof of concept program to implement the prompt engineering methods.

1.2 Objective

The objective of this project is to implement and evaluate various prompt engineering methods on large language models (LLMs) to optimize their performance on question-answering and reasoning tasks. The project seeks to identify effective prompt engineering techniques, such as chain of thought and develop a robust framework for applying this method to LLAMA3 with 8 billion parameters. By assessing the performance of the LLM using the technique on GSM8K dataset, the project aims to measure accuracy, relevance, and the quality of responses, comparing the model's answers to answers given in the dataset to calculate accuracy and error rates. Comprehensive reports will be produced to document the methods, implementation processes, and findings, providing actionable insights and recommendations

for future research and application in the field of prompt engineering for LLMs. Through these efforts, the project aims to contribute significantly to the advancement of natural language processing by demonstrating how different prompt engineering techniques can improve the practical utility of large language models.

1.3 Motivation

The motivation for this initiative arises from the transformative impact of large language models (LLMs) on several sectors and the crucial significance of prompt engineering in fully unleashing their capabilities. LLMs, such LLAMA, Falcon, and Mistral, have transformed the way in which humans engage with technology by facilitating natural language conversations that are both instinctive and reminiscent of human speech. These technological breakthroughs have a wide range of uses, such as automating customer service and creating top-notch content. As a result, they greatly improve production and efficiency in several industries.

The effectiveness of LLMs is greatly influenced by the quality and organization of the prompts they receive, despite their tremendous capabilities. Prompt engineering plays a crucial role in directing these models to generate precise, pertinent, and contextually suitable responses. As LLMs advance in complexity, the intricacies of rapid engineering become increasingly crucial. Methods such as chain of thought prompting, few-shot learning, and producing "knowledges" are crucial for accessing the sophisticated reasoning abilities of these models.

The need to systematically explore and improve these prompt engineering methods to make LLMs more reliable and useful led to this project. The objective of the project is to furnish practical information by examining various methodologies and their impact on the performance of the model. This will ensure that LLMs may be effectively utilized in diverse contexts. This project aims to enhance people's understanding of rapid engineering in educational and practical settings by developing innovative approaches to address challenges such as model biases and the need for context management.

Ultimately, the motivation behind this project is to advance the field of natural language processing by demonstrating how strategic prompt engineering can optimize the performance of LLMs, making them more robust, reliable, and useful for a wide range of applications. The objective of the research is to establish optimal methodologies and criteria for effectively handling LLMs by means of rigorous testing and analysis. This will help

both researchers and practitioners, and they will also encourage more innovation and improvement in this field that is so rapidly changing.

2 Background

2.1 Large Language Models

A Large Language Model (LLM) is an advanced type of artificial intelligence specifically developed to comprehend and produce human language. These models are constructed via deep learning techniques, particularly neural networks, that are trained on vast quantities of textual data. The design commonly employs transformer models, such as OpenAI's GPT (Generative Pre-trained Transformer) series, which can analyze and produce text by comprehending the context and meaning of the input data.

Training an LLM involves exposing it to diverse and extensive datasets, including books, articles, websites, and other textual resources, allowing the model to learn the nuances of language, grammar, facts, and even some reasoning abilities. This training process involves adjusting the weights of the neural network through backpropagation to minimize the difference between the predicted and actual outcomes. The result is a model that can perform a wide range of natural language processing (NLP) tasks, such as translation, summarization, question-answering, and text generation [13]. LLMs operate by predicting the next word in a sentence, given the preceding words, which allows them to generate coherent and contextually relevant text. They can understand and generate text in multiple languages, recognize and adhere to various writing styles, and even complete code snippets in programming languages. The capabilities of LLMs are continually improving as models become larger (with more parameters) and as they are trained on increasingly diverse datasets.

Despite their impressive capabilities, LLMs are not without limitations. They can generate plausible but incorrect or nonsensical answers, exhibit biases present in the training data, and lack true understanding or consciousness. They function based on patterns and data they have seen during training, without genuine comprehension or awareness [16]. Nevertheless, LLMs represent a significant advancement in AI, offering powerful tools for a wide array of applications, from customer service chatbots to advanced research assistants.

2.2 LLama3

Meta, formerly known as Facebook, has been at the forefront of artificial intelligence (AI) research and development. As a parent company of various social media platforms, including Facebook, Instagram, and WhatsApp, they have been driving innovation in AI. Specifically, Meta has been working

on Large Language Models (LLMs), which are neural network-based models trained on massive amounts of text data. These models learn to predict the next word in a sentence, capturing intricate language patterns and nuances. Their latest LLM, LLama 3, is their third-generation model and builds upon the success of previous models like BERT and GPT. What sets LLama 3 apart from its predecessors is that it's open source, making it accessible to a wider community of developers and researchers. This openness fosters collaboration and innovation across the AI community.

LLama 3 comes in two versions: an 8-billion-parameter model and an even more powerful 70-billion-parameter model. Larger parameter sizes allow for better representation of complex linguistic structures, enabling the model to understand context, sarcasm, and subtle cues [5] . Additionally, LLama 3 supports multiple languages, making it versatile for global applications. The potential applications of LLama 3 are vast. It can power chatbots, virtual assistants, and customer support systems by generating coherent and contextually relevant responses. It excels in generating natural language text, from articles to creative writing. Moreover, LLama 3 enhances search relevance by understanding user queries better, making it a valuable tool for search engines. However, like all LLMs, LLama 3 faces challenges related to biases present in training data. Efforts are ongoing to reduce bias and improve fairness. Additionally, training and deploying large models require significant computational resources, which can be a challenge. Ethical considerations are also crucial to ensure the positive impact of LLama 3.

In summary, LLama 3 represents a leap forward in language modeling, empowering developers and researchers to create sophisticated NLP applications. Its open-source nature fosters collaboration and innovation across the AI community, and its multilingual capabilities make it versatile for global applications. As we move forward with the development of AI, responsible AI practices are crucial to ensure LLama 3's positive impact. [8]

2.3 Prompt Engineering

Prompt engineering is a technique used in working with large language models (LLMs) to optimize their performance by carefully crafting the input prompts. Since LLMs generate responses based on the input they receive, the quality, structure, and context of these prompts significantly influence the output. Prompt engineering involves designing these prompts in a way that maximizes the relevance and accuracy of the model's responses, essentially guiding the model to produce desired results. The process of

prompt engineering begins with understanding the specific task or problem at hand. This might involve tasks such as summarization, translation, question-answering, or creative writing. The prompt is then tailored to provide the model with clear and concise instructions, often including examples or specific formats to follow [3].

For instance, if the task is to generate a summary of a given text, the prompt might include phrases like "Summarize the following text" or "Provide a brief summary." Effective prompt engineering often requires iterative testing and refinement. Initial prompts may yield suboptimal results, necessitating adjustments in wording, structure, or additional context. This iterative process helps in identifying the most effective way to phrase the prompt to elicit high-quality responses. Additionally, advanced techniques like few-shot or zero-shot learning can be employed, where the prompt includes a few examples of the desired output (few-shot) or relies on the model's generalization ability without specific examples (zero-shot) [10].

Another critical aspect of prompt engineering is mitigating biases and ensuring ethical use. Since LLMs can reflect biases present in their training data, carefully designed prompts can help minimize these biases and promote fair and balanced responses. Moreover, prompt engineering can be used to set boundaries and provide clear guidelines to prevent the generation of harmful or inappropriate content. In summary, prompt engineering is a crucial skill in harnessing the full potential of large language models. By crafting well-structured and contextually appropriate prompts, users can guide these models to generate accurate, relevant, and high-quality outputs, making them valuable tools across a wide range of applications.

2.4 Prompt Engineering Techniques

Key techniques in prompt engineering include few-shot learning, which involves providing the model with a few examples of the desired output along with the prompt to help the model understand the task better and generate more accurate responses. Chain-of-thought prompting encourages the model to think through a problem step-by-step, improving its ability to handle complex reasoning tasks. Self-consistency involves generating multiple responses to the same prompt and selecting the most consistent one to enhance the reliability of the model's outputs.

Prompt engineering is used in a wide range of applications, including customer support, content creation, education, and healthcare. While it has shown great promise, it also presents several challenges, such as the time-consuming nature of crafting effective prompts and the need for a deep

understanding of both the task and the model. As language models continue to evolve, prompt engineering techniques must also adapt to leverage new capabilities and address emerging issues. The future of prompt engineering lies in developing more automated and scalable methods for prompt creation and refinement, leveraging advancements in AI to generate and test prompts more efficiently, ultimately making the process more accessible to a broader range of users. [12]

2.4.1 Zero Shot Prompting

Zero-shot prompting is a technique used in natural language processing, particularly with large language models (LLMs), where the model is given a task or query without any prior examples or specific task-related training. In essence, the model is expected to generate a response or perform a task based purely on the information contained within the prompt and its pre-existing knowledge, which was acquired during its training phase on large-scale datasets.

In zero-shot prompting, the model is not provided with any examples that are specific to the task at hand. For instance, if you ask a model to translate a sentence from English to French without showing it any examples of English-to-French translations, that would be a zero-shot task. Large language models, like GPT-3 or LLAMA, are trained on vast amounts of text data from diverse sources. This broad training allows the models to have a general understanding of many topics, languages, and tasks. Zero-shot prompting leverages this broad, general knowledge to perform tasks without additional task-specific training. The effectiveness of zero-shot prompting depends on the model's ability to generalize from its training data to new, unseen tasks. The model uses patterns it has learned from the training data to infer what the prompt is asking for, even if it has never encountered the exact task before.

Imagine you want the model to summarize a piece of text, but you haven't trained the model specifically for summarization, nor have you provided any examples of summaries. A zero-shot prompt might look like: "Summarize the following text: [Insert text here]." The model will attempt to generate a summary based purely on its understanding of the task derived from the prompt and its knowledge of summarization from the training phase. Zero-shot prompting allows a single model to perform a wide variety of tasks without requiring task-specific training or fine-tuning, making it highly versatile. It can be simpler and quicker to implement since it doesn't require gathering task-specific training data or fine-tuning the model on new

tasks. It is useful for quickly testing how well a model can handle new tasks or domains without the need for additional data or training. However, the model's performance in zero-shot tasks is generally lower than in tasks where it has been specifically trained or fine-tuned with examples. It might misunderstand the task or produce less relevant outputs. If the prompt is not clear or if the task is complex, the model might produce incorrect or irrelevant results because it lacks the context provided by task-specific examples. The success of zero-shot prompting depends heavily on the model's training data and architecture. Some tasks may be too specialized or nuanced for a model to handle effectively in a zero-shot scenario [14].

Zero-shot prompting is particularly valuable in scenarios where there is no available labeled data for a specific task, a quick solution is needed for a novel task, or testing a model's generalization ability to new tasks is desired. In summary, zero-shot prompting is a powerful technique that leverages the broad, general knowledge embedded in large language models to perform tasks without additional training. It is a key component of the versatility and adaptability of modern AI systems, enabling them to tackle a wide range of tasks with minimal preparation[15].

2.4.2 Chain of Thought Prompting

Chain of thought prompting is an advanced technique used in natural language processing with large language models (LLMs) to enhance their reasoning and problem-solving abilities. Unlike simple prompts that directly ask for an answer, chain of thought prompting guides the model to break down a complex problem into a sequence of intermediate steps, mirroring the way humans often approach problem-solving. This approach helps the model to systematically consider each aspect of the problem, leading to more accurate and reliable outcomes [1].

In practice, chain of thought prompting involves structuring the prompt so that the model is encouraged to "think aloud" by articulating the reasoning process step-by-step before arriving at a final answer. For instance, if a model is asked to solve a math problem, instead of just asking for the solution, the prompt would guide the model to first explain the steps needed to solve the problem, such as identifying the relevant mathematical principles, performing calculations, and then synthesizing the final result.

This method leverages the model's ability to generate and process sequences of text, using each step as a foundation for the next. By making the reasoning process explicit, chain of thought prompting can significantly improve the model's performance on tasks that require logical reasoning,

multi-step problem-solving, or complex decision-making. It is particularly useful in scenarios where simple, direct answers might overlook important nuances or lead to incorrect conclusions [6].

Chain of thought prompting also enhances transparency, as it allows users to see how the model arrived at its answer, making it easier to spot errors in reasoning and to understand the model's decision-making process. This technique has been shown to improve performance on a variety of tasks, including arithmetic problems, commonsense reasoning, and even ethical decision-making. By fostering a more deliberate and structured approach to problem-solving, chain of thought prompting helps to unlock the full potential of large language models, enabling them to tackle complex challenges with greater accuracy and insight.

2.4.3 Least to Most Prompting

Least to most prompting is a structured technique used in natural language processing with large language models (LLMs) designed to help the model solve complex problems by breaking them down into simpler, more manageable sub-tasks. This approach mimics how humans often tackle difficult problems: by addressing the easiest parts first and progressively moving toward the more challenging aspects. The method enhances the model's ability to handle complex tasks by guiding it through a logical sequence of steps, starting from the least challenging to the most difficult.

In a typical least to most prompting scenario, the prompt is designed to first ask the model to solve the simplest part of the problem. Once the model successfully completes this, the prompt then introduces the next level of difficulty, building on the previous solution. This process continues incrementally until the most challenging part of the problem is addressed. By structuring the problem-solving process in this way, the model is less likely to become overwhelmed by the complexity of the task, and can more effectively use the simpler solutions as a foundation for solving the more difficult parts [2].

For example, if the task involves solving a multi-step reasoning problem, the prompt might first ask the model to identify and solve the basic elements of the problem. After each successful step, the prompt would then gradually introduce more complexity, asking the model to integrate the previous answers into solving the next, more difficult portion of the problem. This stepwise approach ensures that the model focuses on and masters each component of the problem before moving on to the next.

Least to most prompting is particularly useful in tasks that involve multi-

ple layers of reasoning or complex decision-making processes, where a direct approach might lead to errors or incomplete answers. By guiding the model to tackle the problem incrementally, this technique not only improves accuracy but also enhances the model's overall reasoning capabilities. It helps the model to maintain focus, avoid confusion, and build upon each previous success, ultimately leading to a more robust and reliable final outcome [2].

3 Literature Review

3.1 Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond [7]

Yang et al. (2023) present a comprehensive survey on the development, capabilities, limitations, and future directions of large language models (LLMs), with a particular focus on ChatGPT. The paper begins by tracing the evolution of LLMs, highlighting critical advancements such as the introduction of transformers, pre-training and fine-tuning techniques, and the significant scaling of model parameters that have led to current state-of-the-art models [7]. The authors emphasize the groundbreaking nature of the GPT-3 model, which demonstrated unprecedented capabilities in natural language understanding and generation, setting the stage for subsequent innovations in the field.

The survey delves into the diverse applications of ChatGPT, showcasing its versatility across domains such as customer support, content creation, education, and healthcare. In customer support, ChatGPT enhances efficiency by automating responses to inquiries, while in content creation, it aids in generating articles and blogs. In education, it offers personalized tutoring, and in healthcare, it provides preliminary medical advice and mental health support. The technical foundations of LLMs are explored in detail, with a focus on the transformer architecture, attention mechanisms, and the importance of large-scale datasets in training these models. The authors discuss the critical balance between model size and performance, noting that larger models generally exhibit superior language understanding and generation capabilities.

However, the paper also highlights significant ethical considerations, such as bias and fairness, misinformation, and privacy concerns. LLMs can perpetuate biases from their training data, potentially leading to unfair or discriminatory outcomes, and their ability to generate coherent text can be misused to spread false information. Additionally, the handling of sensitive data necessitates robust privacy safeguards to prevent misuse. Despite their strengths, LLMs face several limitations. Interpretability remains a challenge, as understanding the decision-making processes of these models is complex. The resource intensity of training and deploying LLMs raises concerns about environmental impact and accessibility, and their ability to maintain context over long conversations or generate responses that align with nuanced human intent can be limited.

Looking ahead, the authors suggest several areas for future research and

development, including enhancing model efficiency to reduce computational costs, improving robustness to make LLMs more reliable, and exploring human-AI collaboration to ensure LLMs complement rather than replace human expertise. They also advocate for the establishment of regulatory frameworks and standards to address the ethical and societal implications of LLM deployment. In conclusion, Yang et al. underscore that while LLMs like ChatGPT represent significant advancements in artificial intelligence, addressing their technical, ethical, and practical challenges is crucial to realizing their full potential. They call for a multidisciplinary approach to ensure these models are developed and used responsibly, ultimately benefiting society as a whole [7].

3.2 Prompt Engineering Guide [11]

The "Prompt Engineering Guide" by DAIR (Data and Artificial Intelligence Research) offers a comprehensive exploration of the strategies and principles behind designing effective prompts for large language models (LLMs) such as GPT-3 and its successors[11]. The guide underscores the critical role that prompt engineering plays in optimizing the performance of LLMs, noting that well-crafted prompts can significantly influence the accuracy, relevance, and reliability of the model's outputs. It begins by introducing the fundamental concepts of prompt engineering, explaining the distinctions between zero-shot, one-shot, and few-shot prompting. Zero-shot prompts require the model to generate responses based solely on its pre-trained knowledge without any example guidance.

In contrast, one-shot prompts provide a single example to shape the model's response, while few-shot prompts offer multiple examples to establish a pattern for the model to follow. The guide further explores the intricacies of prompt design, including the importance of clarity, context, and specificity in crafting prompts. It emphasizes that clear and specific prompts help mitigate ambiguity and improve the quality of the responses generated by the LLMs. The use of contextual information within prompts is highlighted as a means to enhance the relevance of the outputs, making the responses more aligned with the user's intent. Additionally, the guide addresses common challenges in prompt engineering, such as handling model biases and ensuring ethical usage. It provides practical tips and best practices for designing prompts that minimize biases and promote fairness. In the practical section, the guide offers numerous examples and case studies illustrating successful prompt engineering across various applications. These include customer service automation, content generation, educational tools,

and more.

The examples demonstrate how different prompt structures can lead to vastly different outcomes, showcasing the importance of iterative testing and refinement in the prompt engineering process. The guide also introduces advanced techniques, such as prompt chaining and the use of meta-prompts, to further enhance the capabilities of LLMs. Overall, the "Prompt Engineering Guide" by DAIR serves as an invaluable resource for anyone looking to leverage the power of LLMs effectively. It combines theoretical insights with practical advice, providing a well-rounded approach to mastering prompt engineering. The guide concludes by encouraging ongoing experimentation and learning, as the field of prompt engineering continues to evolve alongside advancements in artificial intelligence technology [11].

3.3 Chain-of-Thought Prompting Elicits Reasoning in Large Language Models [6]

The paper "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models" by Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou, published by Google Research, addresses the limitations of large language models (LLMs) in handling complex reasoning tasks. Despite their impressive capabilities in natural language processing, traditional prompting methods often fall short in eliciting the deep reasoning required for solving multi-step problems [6]. To overcome these limitations, the authors propose a technique called chain-of-thought (CoT) prompting, which involves breaking down a complex problem into a sequence of intermediate reasoning steps that the model follows to arrive at a solution. This method mimics human problem-solving strategies, making the reasoning process transparent and easier to follow.

The researchers experimented with CoT prompting on several LLMs, including GPT-3, across diverse domains such as arithmetic reasoning, common-sense tytgh0-98pwreasoning, and symbolic reasoning. For each task, they designed prompts that explicitly instructed the model to break down the problem into smaller steps, demonstrating how intermediate steps lead to the final answer. The results showed that CoT prompting significantly enhances the performance of LLMs on complex reasoning tasks compared to standard prompting methods, with marked improvements in accuracy, particularly on tasks requiring multiple reasoning steps. The technique proved effective across different LLMs, indicating its broad applicability, and demonstrated robustness across various problem types and difficulty levels.

Detailed examples in the paper illustrate how CoT prompting works in

practice. For instance, in an arithmetic problem, instead of asking the model directly for the final answer, the prompt guides the model to first understand the problem, perform intermediate calculations, and then arrive at the solution. This step-by-step guidance helps the model maintain coherence and accuracy throughout the reasoning process. The findings have significant implications for the development and application of LLMs, suggesting that by enabling models to reason through problems in a human-like manner, CoT prompting can enhance the utility of LLMs in practical applications[6].

3.4 Google guide on Prompt Engineering [4]

The Google guide on Prompt Engineering provides a thorough exploration of techniques and best practices for crafting effective prompts to optimize the performance of large language models (LLMs) [4]. The guide emphasizes that the effectiveness of an LLM can be significantly enhanced by carefully designing the prompts used to elicit responses. It begins by explaining the fundamental concepts of prompt engineering, including the different types of prompts—such as zero-shot, one-shot, and few-shot learning—and their respective impacts on model outputs.

Zero-shot prompts rely on the model's pre-trained knowledge without any examples, one-shot prompts offer a single example to guide the model, and few-shot prompts provide several examples to establish a response pattern. The guide delves into practical strategies for creating effective prompts, highlighting the importance of clarity, context, and specificity. Clear and precise prompts help minimize ambiguity, resulting in more accurate and relevant responses. It also stresses the need for providing sufficient context within the prompt to ensure that the model's responses are aligned with the user's intent. Specificity in prompts helps in reducing generalization errors and enhances the model's ability to produce targeted and useful outputs.

The guide also addresses common challenges in prompt engineering, such as dealing with model biases and ensuring that prompts are designed to promote fairness and ethical use. To illustrate these principles, the guide includes practical examples and case studies from various applications, such as automated customer support, content generation, and educational tools. These examples demonstrate how different prompt structures can lead to different outcomes, underscoring the necessity of iterative testing and refinement. Advanced techniques, such as prompt chaining—where multiple prompts are linked to guide the model through a more complex task—and the use of meta-prompts—prompts that help generate other effective prompts—are also discussed to showcase how prompt engineering can

be taken to the next level.

In conclusion, the Google guide on Prompt Engineering serves as a valuable resource for developers, researchers, and practitioners looking to harness the full potential of LLMs. By combining theoretical insights with practical advice, it offers a comprehensive approach to mastering the craft of prompt engineering. The guide encourages ongoing experimentation and adaptation as the field evolves, ensuring that users can continuously improve their prompt designs to achieve the best possible outcomes from LLMs [4].

3.5 Microsoft guide on Prompt Engineering [9]

The Microsoft guide on Prompt Engineering provides a comprehensive overview of techniques and best practices for designing effective prompts to enhance the performance of large language models (LLMs). The guide emphasizes the pivotal role that prompt engineering plays in leveraging LLMs effectively, highlighting that the way prompts are crafted can significantly influence the quality and relevance of the model's responses. It begins by introducing the fundamental concepts of prompt engineering, explaining how various types of prompts—such as zero-shot, one-shot, and few-shot—impact the behavior and output of LLMs [9]. Zero-shot prompts require the model to respond based solely on its pre-trained knowledge without any provided examples, one-shot prompts include a single example to guide the response, and few-shot prompts present several examples to establish a more defined pattern for the model to follow.

The guide details practical strategies for constructing effective prompts, underscoring the importance of clarity, context, and specificity. Clear and well-defined prompts help reduce ambiguity, leading to more precise and relevant outputs from the model. Context is crucial as it provides the necessary background information that helps the model generate responses that are aligned with the user's intent. Specific prompts also aid in minimizing generalization errors, making the model's responses more targeted and useful. The guide addresses challenges in prompt engineering, such as managing biases inherent in the training data and ensuring that prompts promote ethical use and fairness. To provide practical insights, the guide includes numerous examples and case studies across various applications like customer support automation, content creation, and educational tools. These examples illustrate how different prompt structures can lead to varying outcomes, highlighting the importance of iterative testing and refinement in the prompt engineering process.

The guide also introduces advanced techniques such as prompt chaining,

where a series of interconnected prompts are used to guide the model through complex tasks, and the use of meta-prompts, which are prompts designed to generate other effective prompts, thereby enhancing the model's utility. Overall, the Microsoft guide on Prompt Engineering serves as a valuable resource for developers, researchers, and practitioners aiming to maximize the potential of LLMs. By combining theoretical insights with practical advice, it offers a holistic approach to mastering prompt engineering. The guide encourages continuous experimentation and adaptation, ensuring that users can refine their prompt designs to achieve optimal results from LLMs as the field evolves [9].

4 Implementaion

This section delves into the practical implementation of prompt engineering methods, specifically focusing on the use of the GSM8K dataset and various prompting techniques. The goal is to provide a comprehensive understanding of how to effectively utilize LLMs for complex tasks by leveraging well-crafted prompts. The implementation begins with setting up the necessary system requirements and preparing the GSM8K dataset, which is widely used for benchmarking mathematical problem-solving capabilities in LLMs. The original dataset questions as well as two prominent methods, "chain of thought" and "least to most," are explored in detail.

The provided code in section 4.3 demonstrates how to implement these prompting techniques in a practical setting. It includes functions for generating prompts, sending requests to the model, and saving the responses for further analysis. The "chain of thought" technique encourages the model to break down a problem into smaller, manageable steps, while the "least to most" technique prompts the model to identify and solve subproblems sequentially. By comparing the outputs of these methods, we can gain insights into their effectiveness and identify best practices for prompt engineering. Throughout this section, we will also discuss the structure and significance of the GSM8K dataset and examples of how prompts are formulated and processed. The results of the implementation will be analyzed to highlight the strengths and limitations of each prompting technique.

4.1 GSM8k Dataset

The GSM8K dataset, short for "Grade School Math 8K," is a specialized dataset in the field of natural language processing (NLP) and machine learning, specifically designed to evaluate the mathematical reasoning and problem-solving abilities of large language models. Developed to challenge models with grade-school-level math problems, GSM8K consists of a diverse collection of word problems that require not just basic arithmetic skills but also a good understanding of reasoning and logic to arrive at the correct solution.

Each problem in the GSM8K dataset is presented in natural language, mimicking the way math problems are typically formulated in educational settings. For example, a problem might describe a real-world scenario where the model needs to calculate the total number of items given certain conditions or determine how long it will take for two moving objects to meet. The problems range in complexity, from simple addition and subtraction to

	question	answer
0	Natalia sold clips to 48 of her friends in Apr...	Natalia sold 48/2 = <<48/2=24>>24 clips in May...
1	Weng earns $12 an hour for babysitting. Yester...	Weng earns 12/60 = $<<12/60=0.2>>0.2 per minut...
2	Betty is saving money for a new wallet which c...	In the beginning, Betty has only 100 / 2 = $<<...
3	Julie is reading a 120-page book. Yesterday, s...	Maila read 12 x 2 = <<12*2=24>>24 pages today....
4	James writes a 3-page letter to 2 different fr...	He writes each friend 3*2=<<3*2=6>>6 pages a w...
5	Mark has a garden with flowers. He planted pla...	There are 80/100 * 10 = <<80/100*10=8>>8 more ...
6	Albert is wondering how much pizza he can eat ...	He eats 32 from the largest pizzas because 2 x...
7	Ken created a care package to send to his brot...	To the initial 2 pounds of jelly beans, he add...
8	Alexis is applying for a new job and bought a ...	Let S be the amount Alexis paid for the shoes....
9	Tina makes $18.00 an hour. If she works more ...	She works 8 hours a day for $18 per hour so sh...

Figure 1: First 10 questions and answers of GSM8k dataset

more involved tasks that require multiple steps of reasoning and combining different mathematical operations.

One of the key features of the GSM8K dataset is that it doesn't just test a model's ability to perform calculations, but also its understanding of the underlying concepts and its ability to apply them in context. This makes it a valuable tool for assessing how well a language model can handle tasks that go beyond surface-level text processing, requiring deeper comprehension and logical thinking.

GSM8K is widely used to benchmark and improve the performance of large language models, particularly in the context of "chain of thought" prompting and other advanced prompting techniques that aim to enhance a model's reasoning capabilities. By providing a standardized set of problems, GSM8K enables researchers to measure and compare the effectiveness of different models and techniques in solving math problems that are representative of those found in real-world educational contexts.

4.2 Sample Prompting

To initiate the implementation of our project, we began by presenting several questions from the GSM8K dataset to the LLAMA3 model to evaluate its accuracy, as illustrated in Figure 2. For deploying LLAMA3 on a laptop, we utilized the Ollama platform to install the model. While Ollama can be operated via the terminal, we opted to use OpenwebUI to provide a more user-friendly interface for the prompting process.

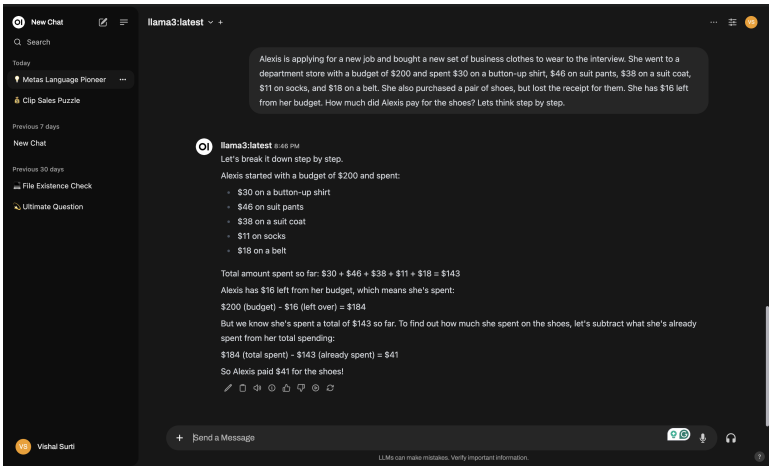


Figure 2: Sample prompting a question to LLAMA3

4.3 Automating prompts

The GSM8K dataset comprises of 8,794 questions, making manual prompting a highly impractical task. Additionally, we need to prompt this set of questions three times, as we are implementing three distinct prompt engineering methods. To address this challenge, we developed a Python script to automate the entire process. This Python code is designed to evaluate the performance of a large language model (in this case, named "llama3") on a dataset using two different prompting techniques: "chain of thought" and "least to most." The code reads a dataset of questions and answers, applies the chosen prompting technique to each question, and then sends the prompt to the model to generate a response. The responses are saved to a file for further analysis. Lets look at the code in detail.

The code begins by importing several essential Python libraries: os, requests, pandas, and time. Each of these libraries serves a specific purpose in the program. The os library is used for interacting with the operating system, particularly for tasks like checking if a file exists and managing file paths, which is critical when saving and loading data. The requests library is a popular HTTP library in Python that allows the code to make HTTP requests, which in this case, are used to interact with the locally hosted language model via its API. This enables the program to send prompts to the model and receive generated responses. The pandas library, referred to as pd, is a powerful tool for data manipulation and analysis. It is used here to read the dataset from CSV files into DataFrames, which makes it easy

```
df_train = pd.read_csv('dataset/gsm8k_train.csv')
df_test = pd.read_csv('dataset/gsm8k_test.csv')

ds = {'train': df_train, 'test': df_test}

#prompting_technique = 'zero_shot'
#prompting_technique = 'chain_of_thought'
prompting_technique = 'least_to_most'
```

Figure 3: Loading dataset

to handle, manipulate, and analyze the data. Finally, the time library is used to measure the time taken for the model to generate responses, which is useful for performance evaluation.

The dataset used for evaluating the language model is loaded into the program using the pandas library as seen in Figure 3. The code reads two CSV files, 'gsm8ktrain.csv', which are expected to contain the training and test data for the GSM8K dataset. These CSV files are read into pandas DataFrames, 'dftrain' and 'dftest', respectively. A DataFrame is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns). In this case, 'dftrain' holds the training data, and 'dftest' holds the test data. These DataFrames are then stored in a dictionary ds with keys 'train' and 'test', making it easy to access and iterate over the training and test datasets in a uniform way. This setup allows the code to process the data systematically, whether it is the training or test set.

The code includes a mechanism to switch between different prompting techniques as seen in Figure 3. The 'promptingtechnique' variable is set to either 'chainofthought' or 'leasttomost', depending on the desired prompting strategy. By commenting out one line and uncommenting the other, the user can easily toggle between these two techniques. This design makes the code flexible, allowing for quick experimentation with different prompting methods. The chosen technique will later determine how the prompts are constructed before being sent to the language model. This is crucial because the way a prompt is structured can significantly influence the model's output, especially in tasks that require logical reasoning or multi-step problem-solving.

The Timer class is a utility class designed to measure the time taken for certain operations, particularly the time it takes for the model to generate a response. The class has three main methods: start, end, and elapsed. When

the start method is called, it records the current time, which marks the beginning of the operation to be timed. The end method is called once the operation is complete, recording the end time and calculating the elapsed time by subtracting the start time from the end time. The elapsed method formats this duration into a string that shows the time in seconds with three decimal places. This functionality is essential for performance monitoring, as it provides insights into how long the model takes to process each prompt, which can be important for evaluating the efficiency of different prompting techniques.

The 'getresponse' function is responsible for sending the prompt to the language model and retrieving the generated response. It takes three parameters: model, which specifies the model to use (in this case, 'llama3'); question, which is the prompt to be sent to the model; and stream, a boolean flag that determines whether the response should be streamed or sent in one go. The function constructs a JSON object with these parameters and sends it to the model's API via a POST request using the requests library. The response from the API is expected to be in JSON format, and the function extracts the actual text response from this JSON object and returns it. This function abstracts away the details of the HTTP request, providing a simple interface for sending prompts and receiving responses.

As seen in figure 4 two prompting functions, 'chainofthought' and 'leasttomost', are defined to create specific types of prompts based on the chosen technique. The 'chainofthought' function appends the phrase "Let's think about it step by step." to the original question. This phrase encourages the model to break down the problem into smaller, sequential steps, which can improve the quality of the reasoning process and the accuracy of the final answer. On the other hand, the 'leasttomost' function appends the phrase "What subproblems must be solved before answering the question?" to the question. This prompt guides the model to first identify and solve simpler subproblems before tackling the main problem, a technique that can help manage complexity and lead to better problem-solving outcomes. These functions are critical in shaping how the model approaches the problem, and they directly impact the quality of the generated responses.

The 'savetofiletest' function is responsible for saving the generated responses to a CSV file. It takes four parameters: filename, which is the name of the file where the data will be saved; question, which is the original question or prompt; answer, which is the model's generated response; and 'datasetans', which is the correct answer from the dataset. The function first checks if the specified file already exists in the current working directory. If the file exists, it reads the existing data into a pandas DataFrame, appends

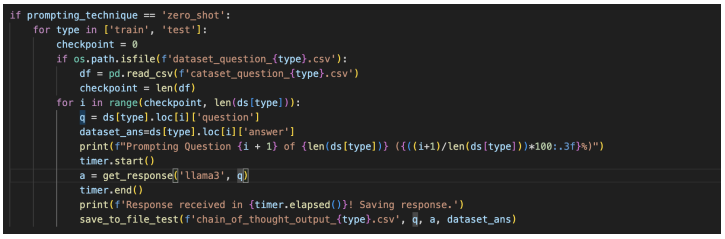
```
def chain_of_thought(question):
    chain_question=question+"\nLets think about it step by step."
    return chain_question


def least_to_most(question):
    return question+"\n What subproblems must be solved before answering the question?"
```

Figure 4: Automating chain of thought and least to most prompting

the new data (question, answer, and dataset answer) as a new row, and then writes the updated DataFrame back to the file. If the file does not exist, the function creates a new DataFrame with the appropriate columns, adds the data, and saves it as a new CSV file. This function ensures that the responses are persistently stored for later analysis, allowing for easy comparison of the model's output with the correct answers from the dataset.

The main loop in the code processes each question in the dataset using the selected prompting technique and saves the results. The loop is structured to handle both the "train" and "test" datasets separately, iterating over the questions in each set. Before starting the iteration, the code checks if an output file '(chainofthoughtoutputtype.csv' or 'leasttomostoutputtype.csv)' already exists. If such a file exists, the code reads it to determine how many questions have already been processed, setting a checkpoint to resume processing from where it left off. This prevents reprocessing the same data in case the code is interrupted and needs to be rerun. For each question, the loop constructs the prompt using the chosen technique, sends it to the model via the 'getresponse' function, and measures the time taken using the Timer class. Once the model's response is received, it is saved to the appropriate output file using the savetofiletest function. This loop structure ensures that all questions are processed systematically, and the results are stored for further analysis.

When the 'promptingtechnique' is set to 'chainofthought', the code follows a specific process tailored to this technique. The loop starts by checking for any existing output file '(chainofthoughtoutputtype.csv)' to determine if there is a checkpoint, allowing it to resume from where it left off in case the process was previously interrupted. For each question in the dataset, the code constructs a "chain of thought" prompt by appending the phrase "Let's think about it step by step." to the original question. This modified prompt is then sent to the language model using the 'getresponse' function, and the time taken for the model to generate a response is recorded. The model's



```
if prompting_technique == 'zero_shot':
    for type in ['train', 'test']:
        checkpoint = 0
        if os.path.isfile(f'dataset_question_{type}.csv'):
            df = pd.read_csv(f'cataset_question_{type}.csv')
            checkpoint = len(df)
        for i in range(checkpoint, len(ds[type])):
            q = ds[type].loc[i]['question']
            dataset_ans=ds[type].loc[i]['answer']
            print(f"Prompting Question {i + 1} of {len(ds[type])} ({((i+1)/len(ds[type]))*100:.3f}%)")
            timer.start()
            a = get_response('llama3', q)
            timer.end()
            print(f'Response received in {timer.elapsed()}! Saving response.')
            save_to_file_test(f'chain_of_thought_output_{type}.csv', q, a, dataset_ans)
```

Figure 5: Main loop for prompting zero shot

```
if prompting_technique == 'chain_of_thought':
    for type in ['train', 'test']:
        for i in range(len(ds[type])):
            q = ds[type].loc[i]['question']
            dataset_ans=ds[type].loc[i]['answer']
            print(f"Prompting Question {i + 1} of {len(ds[type])}")
            cq=chain_of_thought(q)
            a = get_response('llama3', cq)
            print('Response received! Saving response.')
            save_to_file_test(f'chain_of_thought_output_{type}.csv', q, a, dataset_ans)
            print(a)

if prompting_technique == 'least_to_most':
    for type in ['train', 'test']:
        for i in range(len(ds[type])):
            q = ds[type].loc[i]['question']
            dataset_ans=ds[type].loc[i]['answer']
            print(f"Prompting Question {i + 1} of {len(ds[type])}")
            ltm = least_to_most(q)
            a = get_response('llama3', ltm)
            print('Response received! Saving response.')
            save_to_file_test(f'least_to_most_output_{type}.csv', q, a, dataset_ans)
            print(a)
```

Figure 6: Main loop for prompting Chain of thought and Least to most

response, along with the original question and the correct answer from the dataset, is saved to the output file using the 'savetofiletest' function. This process continues until all questions in the dataset have been processed. The "chain of thought" technique is particularly useful for problems that require multi-step reasoning, as it encourages the model to break down the problem into smaller, more manageable steps. When the 'promptingtechnique' is set to 'leasttomost', the code follows a similar process but tailored to the "least to most" prompting technique. The same can be seen in Figure 6.

The code provides a framework for testing different prompting techniques on a dataset using a large language model. It allows for easy switching between different techniques and saves the model's responses for later analysis. The use of the Timer class helps track how long the model takes to generate each response, providing insights into the performance efficiency of different prompting strategies.

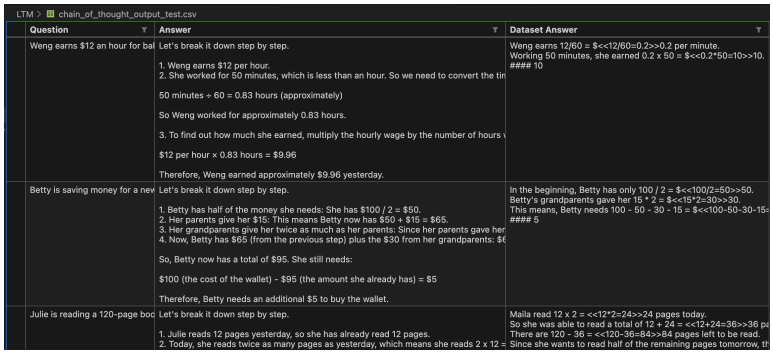


Figure 7: Chain of Thought output

5 Evaluation

5.1 Accuracy Calculation

To calculate the accuracy of the 3 prompt engineering methods, we will write a code that can compare the answers generated by LLAMA3 to the answers in the dataset. The code starts by importing two important Python libraries: pandas and re. The pandas library, imported as pd, is essential for handling and manipulating tabular data in the form of DataFrames. It allows easy reading, writing, and processing of data from CSV files, which is crucial in this script as it deals with multiple CSV files containing model outputs and correct answers. The re library, which stands for "regular expression," is used for pattern matching and searching within strings. In this script, re is particularly used to extract numerical values from the text, which is key to comparing the model's generated answers with the correct answers.

The core functionality of this code is encapsulated within the accuracy function. This function is designed to calculate the accuracy of the model's outputs by comparing the answers generated by the model to the correct answers in the dataset. The function takes a single argument, file, which is the name of the CSV file that contains the dataset's questions, the model's generated answers, and the correct answers. The function begins by reading the CSV file into a pandas DataFrame using 'pd.readcsv(file)'. The CSV file is expected to have columns labeled "Dataset Answer" and "Answer." These columns contain the correct answers from the dataset and the model's generated answers, respectively. The function converts these columns into two separate lists, rows and rowsans, for easier processing.

The accuracy of the model is calculated by comparing the numerical

```
adf=pd.read_csv('dataset_question_test.csv')
print(adf.head(3))
```

✓ 0.0s

```
                                            Question  \
0  Janet’s ducks lay 16 eggs per day. She eats th...
1  A robe takes 2 bolts of blue fiber and half th...
2  Josh decides to try flipping a house.  He buys...

                                              Answer  \
0  Janet's ducks lay 16 eggs, but she eats 3 and ...
1  The amount of white fiber needed is 1/2 the am...
2  The original price is 80000. After 50000 was s...

                                      Dataset Answer
0  Janet sells 16 - 3 - 4 = <<16-3-4=9>>9 duck eg...
1  It takes 2/2=<<2/2=1>>1 bolt of white fiber\nS...
2  The cost of the house and repairs came out to ...
```

Figure 8: Zeroshot output

LTM > least_to_most_output_test.csv

Question	Answer	Dataset Answer
Natalia sold clips to 48 of her friend	Let's break this down step by step! To find out how many clips Natalia sold altogether in April and May, we ne **April:** Natalia sold 48 clips in April. That's a given fact! **May:** In May, Natalia sold half as many clips as she did in April. Since 48 ÷ 2 = 24 So, Natalia sold 24 clips in May. **Altogether:** Now, we need to add the number of clips sold in both mo April: 48 May: 24 Total: 48 + 24 = 72 Therefore, Natalia sold a total of **72** clips in April and May.	Natalia sold 48/2 = <<48/2=24>>24 clips in May. Natalia sold 48+24 = <<48+24=72>>72 clips altogether in April and May. #### 72
Weng earns $12 an hour for babysitt	To solve this problem, we need to determine how many hours Weng sper The first subproblem is to convert the time from minutes to hours. Since 50 minutes ÷ 60 = 0.833 hours So, Weng spent approximately 0.83 hours babysitting. The second subproblem is to multiply the time spent babysitting by her h 0.833 hours × $12/hour = $10.00	Weng earns 12/60 = $<<12/60=0.2>>0.2 per minute. Working 50 minutes, she earned 0.2 x 50 = $<<0.2*50=10>>10. #### 10

Figure 9: Least to most output

```
def accuracy(file):
    # Test accuracy for train data
    dftrain=pd.read_csv(file)
    rows=list(dftrain['Dataset Answer'])
    rowsans=list(dftrain['Answer'])
    true=0
    false=0
    for i in range(len(rows)):
        a= re.findall("\d+",rows[i])
        b= re.findall("\d+",rowsans[i])
        if len(b)==0:
            true+=1
            continue
        else:
            if a[len(a)-1] == b[len(b)-1]:
                true+=1
            else:
                false+=1
    truetest= ((true/(true+false))*100)
    falsetest= ((false/(true+false))*100)
    return truetest
✓ 0.0s
```

Figure 10: Accuracy Function

values in the correct answers (rows) with those in the model's generated answers (rowsans). The function initializes two counters, true and false, to keep track of the number of correct and incorrect answers, respectively. The function then iterates over each pair of correct and generated answers using a for loop. Within the loop, it uses the re.findall("d+", rows[i]) function to extract all the numerical values from the current correct answer and the generated answer. These numerical values are stored in lists a and b. If the generated answer (b) does not contain any numerical values (i.e., len(b) == 0), the function assumes the answer is correct and increments the true counter. This is a specific assumption made by the code, which might be based on the nature of the task or dataset where an absence of numbers in the generated answer could imply correctness in some contexts. If the generated answer does contain numerical values, the function compares the last numerical value in the correct answer (a[len(a)-1]) with the last numerical value in the generated answer (b[len(b)-1]). If these values match, the answer is considered correct, and the true counter is incremented. If they do not match, the false counter is incremented.

Once all the answers in the dataset have been evaluated, the function calculates the accuracy as a percentage. The accuracy is defined as the ratio of correct answers (true) to the total number of answers (true + false). The function then returns this accuracy percentage. This accuracy metric

```
cot_train=accuracy('chain_of_thought_output_train.csv')
cot_test=accuracy('chain_of_thought_output_test.csv')
data_train=accuracy('dataset_question_train.csv')
data_test=accuracy('dataset_question_test.csv')
ltm_train=accuracy('least_to_most_output_train.csv')
ltm_test=accuracy('least_to_most_output_test.csv')

print(f'Zero Shot {(data_train+data_test)/2:.3f}%')
print(f'Chain of thought {(cot_train+cot_test)/2:.3f}%')
print(f'Least to Most {(ltm_train+ltm_test)/2:.3f}%')

✓ 1.4s
```

Figure 11: Calling accuracy function for all three prompt engineering methods

provides a simple yet effective way to evaluate how well the model's generated answers match the correct answers in the dataset. It is particularly useful when the task involves generating specific numerical answers, such as in mathematical problem-solving or numerical reasoning tasks.

The code then proceeds to evaluate the accuracy of different prompting techniques by calling the accuracy function on various CSV files that contain the outputs of these techniques. Each of these evaluations calls the accuracy function with the corresponding CSV file as an argument, and the returned accuracy percentage is stored in the respective variable.

Finally, the code calculates the average accuracy for each prompting technique and prints the results. For each technique, the average accuracy is computed by taking the mean of the training and test accuracies. The results are printed in a formatted string that displays the average accuracy for "Zero Shot" (baseline), "Chain of Thought," and "Least to Most" prompting techniques.

This output provides a clear comparison of how each prompting technique performs in terms of accuracy. By averaging the training and test accuracies, the script offers a balanced view of the model's performance across different data splits. This is particularly useful for understanding the effectiveness of different prompting strategies in guiding the model to produce correct answers.

5.2 Accuracy Output

The bar chart in Figure 13 compares the accuracy of three different prompting techniques: Zero Shot, Chain of Thought, and Least to Most. Zero Shot is the default approach where the model is given a question without any specific prompting strategy. The model attempts to answer the question based solely on its pretrained knowledge without any additional context or step-by-step guidance. Chain of Thought technique involves prompting the model to think through a problem step by step before providing an answer. This approach encourages the model to break down the problem into smaller logical steps, which can lead to more accurate answers, particularly for complex questions. Least to Most technique prompts the model to solve subproblems before arriving at the final answer. It guides the model to first identify and address smaller, more manageable parts of the problem, then combine these solutions to form a complete answer.

The Zero Shot technique has the highest accuracy at 68.51. This suggests that, for the dataset used, the model's default ability to generate answers without additional prompting is quite effective. The Chain of Thought technique achieves an accuracy of 67.01. This is slightly lower than the Zero Shot approach, indicating that breaking down the problem step by step did not significantly improve accuracy over the baseline. The Least to Most technique has the lowest accuracy at 52.38. This suggests that guiding the model to solve subproblems before answering the main question may not have been as effective for this particular dataset or task, leading to a decrease in overall accuracy.

The chart indicates that while prompting techniques like Chain of Thought and Least to Most are often expected to improve the model's performance, in this case, the Zero Shot approach outperformed the others. This could be due to the nature of the dataset or the specific implementation of these techniques. The lower accuracy of the Least to Most technique might suggest that the model struggled to effectively break down the problems into subproblems or that the subproblems did not contribute effectively to the final answer. The Chain of Thought technique, while slightly less accurate than Zero Shot, still performed reasonably well, suggesting that step-by-step reasoning is somewhat helpful but not necessarily superior to the model's default answering strategy.

This bar chart visually conveys the effectiveness of different prompting techniques on a given dataset, showing that in this instance, the default (Zero Shot) approach yielded the highest accuracy. The results could guide further experimentation with these techniques, perhaps refining the prompts

```
Zero Shot 68.514%
Chain of thought 67.008%
Least to Most 52.382%
```

Figure 12: Accuracy of the prompting techniques

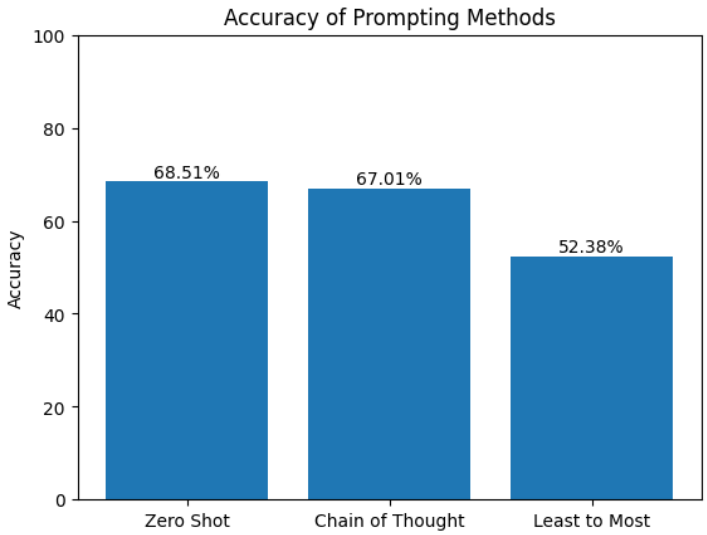


Figure 13: Accuracy graph for different prompts

or exploring additional datasets to see if these trends hold.

6 Challenges

6.1 Computational Challenges and Resource Limitations

One of the most significant challenges encountered during the project was related to computational resources. As large language models like LLAMA3 require substantial processing power and memory to perform inference, my personal laptop was quickly overwhelmed by the demands of the task. Running inference on a high-capacity language model involves not only the loading of large model weights into memory but also intensive processing during the generation of responses. This was particularly problematic when experimenting with different prompt engineering techniques like Chain of Thought and Least to Most, as these techniques often require multiple iterations or complex calculations, further straining limited computational resources.

To overcome this, I had to explore alternative setups that could provide the necessary computational power without compromising the project's progress. The solution involved setting up an external AI server on a different, more powerful machine. This server was equipped with Docker, a platform that allows the creation of lightweight, isolated containers to run applications. I used Docker to deploy Ollama, a tool designed for running and managing large language models. However, accessing this server remotely posed an additional challenge, which was elegantly solved by using ngrok, a tunneling tool that creates secure tunnels to localhost. Ngrok enabled seamless access to the external server, allowing me to perform model inference as if it were being done locally on my laptop. This setup not only mitigated the computational limitations but also provided a scalable solution for further experimentation and testing.

6.2 Dataset Answer Discrepancy

Another challenge that surfaced during the project was related to the accuracy and reliability of the GSM8K dataset, which was used for evaluating the performance of different prompting techniques. The GSM8K dataset, although widely recognized as a standard for evaluating arithmetic and reasoning tasks, was found to contain several discrepancies. In some instances, the answers provided in the dataset were incorrect or incomplete, leading to confusion and potential misinterpretation of the model's performance. This was particularly evident when using advanced prompting techniques like Chain of Thought and Least to Most. In several cases, LLAMA3, when guided by these techniques, produced answers that were more accurate or logically sound than the ones provided in the dataset. As seen in Figure 14,

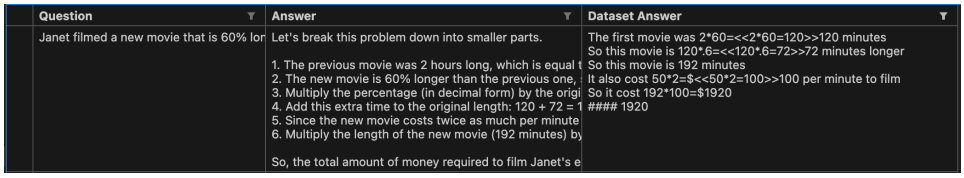


Figure 14: Incorrect answer in GSM8K

The answer is GSM8K is 1920 which is wrong, the model on the other hand predicted the correct value which is 19200.

This discrepancy posed a significant challenge during the evaluation phase, as it complicated the process of determining whether a model's output was truly correct or if it was being unfairly penalized due to errors in the dataset. I have not been able to tackle this problem yet but I do have an approach by which this issue can be solved to an extent. Addressing this issue requires a critical approach to dataset evaluation, where each model-generated answer has to be manually cross-checked against the dataset and, when necessary, corrected or validated independently. This process will be time-consuming and require careful consideration to ensure that the evaluation metrics accurately reflect the model's capabilities rather than the limitations of the dataset.

6.3 Interpretability and Evaluation of Model Outputs

Another significant challenge was the interpretability and evaluation of the outputs generated by the model. Large language models like LLAMA3 are capable of producing human-like text that can be difficult to evaluate objectively. In many cases, especially with reasoning tasks, the model's answer might be technically correct but expressed in a way that differs from the expected format, making automated evaluation tricky. This issue was particularly pronounced when comparing the outputs of different prompting techniques, as each technique might guide the model to produce answers in different styles or formats.

7 Future work

7.1 Exploration of More Diverse Datasets

One of the limitations of the current project is its reliance on a single dataset, GSM8K, which focuses on grade-school math problems. While this dataset is useful for evaluating reasoning and arithmetic tasks, it does not cover the broad spectrum of challenges that large language models might encounter in different domains. Future work could involve applying the same prompt engineering techniques to datasets from various fields, such as legal reasoning (e.g., case law datasets), medical diagnostics (e.g., clinical notes), or technical troubleshooting (e.g., software bug reports). This would allow for a more comprehensive understanding of how these techniques generalize across different types of reasoning tasks and could reveal domain-specific strengths or weaknesses in the models.

7.2 Integration of Few-Shot Learning

While this project focused on zero-shot prompting techniques, integrating few-shot learning could provide a significant boost in performance, especially for more complex or nuanced tasks. Few-shot learning involves providing the model with a few examples of correct behavior or problem-solving before asking it to generate its own responses. This approach can help the model better understand the context and expectations, leading to more accurate and relevant outputs. Future research could explore how few-shot learning interacts with techniques like Chain of Thought and Least to Most, potentially uncovering synergies that improve overall accuracy and reliability. Additionally, experiments could be conducted to determine the optimal number of examples needed to achieve the best performance without overwhelming the model or increasing computational costs significantly.

7.3 Automated Prompt Optimization

Crafting effective prompts is a nuanced task that often requires manual tweaking and a deep understanding of the model's behavior. To streamline this process and potentially discover more effective prompts, future work could involve developing or integrating automated prompt optimization tools. Techniques like reinforcement learning, where the model is rewarded for generating accurate or relevant responses, or genetic algorithms, which iteratively evolve prompts based on their performance, could be employed to systematically refine and improve prompts. This automation would not only

save time but also could lead to the discovery of novel prompting strategies that outperform manually crafted prompts.

7.4 Cross-Model Comparison

The current project is limited to evaluating prompting techniques on a single model, LLAMA3. However, the effectiveness of these techniques can vary significantly across different models, depending on their architecture, training data, and inherent capabilities. Future work could involve a comprehensive cross-model comparison, where the same prompt engineering methods are applied to other popular large language models, such as GPT-4, Falcon, or Mistral. This research would help determine which models are most responsive to specific prompting techniques and provide insights into the strengths and limitations of each model in various reasoning tasks. Additionally, this comparison could guide practitioners in choosing the most appropriate model and prompting strategy for their specific applications.

8 Conclusion

This project has focused on exploring and evaluating various prompt engineering techniques applied to large language models (LLMs) in the context of reasoning tasks, specifically utilizing the GSM8K dataset. Through the implementation and comparison of different prompting methods, such as Zero-Shot, Chain of Thought, and Least to Most, the research has provided valuable insights into how the structuring and framing of prompts can significantly influence the performance of LLMs like LLAMA3.

Prompt engineering has emerged as a critical tool in harnessing the full potential of LLMs. The project began by exploring basic prompt engineering concepts, like Zero-Shot prompting, which relies on the model's inherent ability to generate responses without any prior examples. Although Zero-Shot prompting provides a baseline for model performance, more sophisticated techniques like Chain of Thought and Least to Most were implemented to improve the model's reasoning capabilities.

Chain of Thought Prompting involved guiding the model to break down complex problems into a series of logical steps, promoting a more structured approach to reasoning. By encouraging the model to think "step by step," Chain of Thought prompting often led to more accurate answers, particularly for problems requiring multi-step solutions. Least to Most Prompting further refined the approach by explicitly prompting the model to identify and solve subproblems before tackling the overall question. This hierarchical problem-solving strategy helped in simplifying complex questions, making it easier for the model to arrive at the correct answers.

Throughout the project, several challenges were encountered that shaped the research trajectory. One of the primary challenges was computational, as the initial setup on a standard laptop proved insufficient for running inference using the Ollama server, which necessitated the use of a more powerful external AI server. This challenge was mitigated by setting up the server on a different machine using Docker, with Ollama tunneled through ngrok to allow for remote access and testing. This solution not only resolved the computational limitations but also highlighted the importance of having scalable infrastructure for AI research.

Another significant challenge was the discrepancy between the GSM8K dataset answers and the responses generated by the model. In some cases, the model, particularly when using advanced prompting techniques like Chain of Thought or Least to Most, provided answers that were more accurate than those in the dataset. This raised questions about the reliability of the dataset and underscored the need for rigorous evaluation methodologies

when benchmarking model performance.

Looking forward, there are several avenues for further research that could enhance the findings of this project. One promising direction is the exploration of more diverse datasets from various domains such as legal, medical, and technical fields. This would provide a broader evaluation of the prompt engineering techniques and their applicability across different types of reasoning tasks. Additionally, integrating Few-Shot Learning, where the model is provided with a few examples before generating responses, could further improve performance on complex tasks.

Another potential area of future research involves the automation of prompt optimization. By leveraging reinforcement learning or genetic algorithms, researchers could develop tools that automatically refine prompts to achieve better results, thus saving time and potentially discovering new, more effective prompting strategies. Cross-model comparisons would also be valuable, as they would reveal how different LLMs respond to the same prompting techniques, guiding the selection of the most suitable models for specific tasks.

In conclusion, this project has demonstrated the importance of prompt engineering in optimizing the performance of large language models on reasoning tasks. Through the implementation of various prompting techniques, we have shown that the way a question is posed to the model can dramatically affect the accuracy and quality of the responses generated. While computational challenges and dataset discrepancies presented obstacles, they also provided valuable learning opportunities that informed the development of more robust research methodologies. The findings from this project lay a solid foundation for future work, which could include expanding the range of datasets, integrating advanced learning techniques, automating prompt optimization, and conducting cross-model comparisons. As the field of AI continues to evolve, prompt engineering will remain a key area of research, with the potential to unlock even greater capabilities in LLMs and their applications.

References

[1] Chain of Thought Prompting : https://learnprompting.org/docs/intermediate/chainofthought.

[2] Le Hou Jason Wei Nathan Scales Xuezhi Wang Dale Schuurmans Claire Cui Olivier Bousquet Quoc Le Ed Chi Denny Zhou, Nathanael Schärli. Least-to-most prompting enables complex reasoning in large language models : arxiv.org/abs/2205.10625.

[3] Daudi Jjingo Joyce Nakatumba-Nabende Ggaliwango Marvin, Nakayiza Hellen. Prompt engineering in large language models. In *Data Intelligence and Cognitive Informatics*, pages 387–402, 2024.

[4] Prompt engineering for generative ai: https://developers.google.com/machine-learning/resources/prompt-eng.

[5] Welcome llama3 - meta's new open llm : https://huggingface.co/blog/llama3.

[6] Dale Schuurmans Maarten Bosma-Brian Ichter Fei Xia Ed Chi Quoc Le Denny Zhou Jason Wei/, Xuezhi Wang. Chain-of-thought prompting elicits reasoning in large language models : arxiv:2201.11903.

[7] Ruixiang Tang Xiaotian Han-Qizhang Feng Haoming Jiang Bing Yin Xia Hu Jingfeng Yang, Hongye Jin. Harnessing the power of llms in practice: A survey on chatgpt and beyond : arxiv:2304.13712.2023.

[8] https://about.fb.com/news/2024/06/releasing-new-ai-research-models-to-accelerate-innovation-at-scale/.

[9] Overview of prompts : https://learn.microsoft.com/en-us/ai-builder/prompts-overview.

[10] Sriparna Saha Vinija Jain Samrat Mondal Aman Chadha Pranab Sahoo, Ayush Kumar Singh. A systematic survey of prompt engineering in large language models: Techniques and applications : arxiv:2402.07927.

[11] Prompt engineering guide : https://www.promptingguide.ai/.

[12] Prompting techniques : https://www.promptingguide.ai/techniques.

[13] What are large language models: https://www.ibm.com/topics/large-language-models.

[14] What Are Zero-Shot Prompting and Few-Shot Prompting : https://machinelearningmastery.com/what-are-zero-shot-prompting-and-few-shot-prompting/.

[15] Zero-Shot Prompting : https://www.promptingguide.ai/techniques/zeroshot.

[16] Arkaitz Zubiaga. Natural language processing in the era of large language models. *Frontiers in Artificial intelligence*, 2024.

9 Appendix

9.1 How to use my code

To access my code please use the git repository link " https://github.com/VishalSurti/Prompt-Engineering-in-Large-Language-Models". You will find a README.md file
with all the information to run the code and get the results.