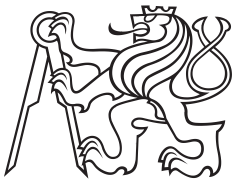
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Evaluation of recommendations for LLM prompt engineering

Boris Rakovan

Supervisor: doc. Mgr.Viliam Lisý, MSc., Ph.D.
Field of study: Open Informatics
Subfield: Data Science
May 2024

Acknowledgements

I would like to thank my supervisor, doc. Mgr.Viliam Lisý, MSc., Ph.D., for his help, guidance and invaluable advice during my work on this thesis.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

During the preparation of this work, I used ChatGPT[1] tool to interact with the GPT-4 language model in order to fix mistakes and reformulate existing texts. After using this tool, I reviewed and edited the content as needed and take full responsibility for the content of this work.

In Prague, 20. 5. 2024

[1] https://chatgpt.com/

Abstract

Prompt engineering is a crucial skill for the efficient use of large language models (LLMs). Despite many online resources offering recommendations on prompt engineering, these guidelines are rarely rigorously evaluated or compared against alternatives. This thesis proposes a rigorous methodology for evaluating different prompt engineering recommendations and tests four prevalent recommendations found in online literature across three different evaluation tasks from the natural language processing domain. Experiments are conducted on five state-of-the-art LLMs (GPT-3.5, LLaMA 7B, LLaMA 13B, LLaMA 70B and Gemini 1.0 Pro). The contribution of this thesis is twofold: first, it tests the efficiency of existing prompt engineering recommendations; second, it introduces a comprehensive framework for prompt evaluation that is easily extensible to various recommendations, evaluation tasks, and LLM models. We show that none of the four recommendations consistently impacts the result quality, indicating LLM users do not need to strictly adhere to these guidelines when designing prompts.

Keywords: natural language processing, large language models, prompt engineering, text classification

Supervisor: doc. Mgr.Viliam Lisý, MSc., Ph.D.
Prague
Karlovo náměstí 13
Faculty of Computer Science

Abstrakt

Prompt engineering je klíčovou dovedností pro efektivní využití velkých jazykových modelů (LLM). Navzdory mnoha online zdrojům nabízejícím doporučení ohledně prompt engineeringu jsou tyto doporučení zřídka důkladně otestovány nebo srovnány s alternativami. Tato práce navrhuje důkladnou metodiku pro otestování různých prompt engineering doporučení a testuje čtyři různá doporučení nalezená v online literatuře napříč třemi různými evaluačními úlohami z oblasti zpracování přirozeného jazyka. Experimenty jsou prováděny na pěti state-of-the-art jazykových modelech (GPT-3.5, LLaMA 7B, LLaMA 13B, LLaMA 70B a Gemini 1.0 Pro). Přínos této práce je dvojí: za prvé, testuje efektivitu stávajících doporučení pro prompt engineering; za druhé, představuje komplexní framework pro testování promptů, který je snadno rozšiřitelný na různá doporučení, evaluační úlohy a modely. Ukáže se, že žádné ze čtyř doporučení konzistentně neovlivňuje kvalitu výsledků, což naznačuje, že uživatelé LLM nemusí při navrhování promptů striktně dodržovat tyto pokyny.

Klíčová slova: zpracování přirozeného jazyka, velké jazykové modely, prompt inženýrství, klasifikace textu

Překlad názvu: Vyhodnocení doporučení pro psaní dotazů na velké jazykové modely

Contents

Figures

Tables

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Rakovan Boris** Personal ID number: **483500**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Data Science**

II. Master’s thesis details

Master’s thesis title in English:

Evaluation of recommendations for LLM prompt engineering

Master’s thesis title in Czech:

Vyhodnocení doporučení pro psaní dotazů na velké jazykové modely

Guidelines:

Prompt engineering is an essential skill for maximizing the effectiveness of large language models (LLMs). Despite a lot of online resources offering guidance on prompt engineering, these recommendations are often not rigorously evaluated or compared to alternatives. This thesis aims to critically assess various prompt engineering techniques and methodologies, compare their effectiveness, and establish a framework for their systematic evaluation.
In this work, the student will:
(1) Conduct a thorough review of existing academic and non-academic resources that offer recommendations for prompt engineering
(2) Develop a rigorous methodology to test and evaluate at least three distinct prompt engineering recommendations on appropriate benchmarks
(3) Test the recommendations on suitable LLMs, accessible either through an API or as an open-source project
(4) Analyze the results to determine the effectiveness of each recommendation as well as compare different prompt engineering techniques

Bibliography / sources:

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H. and Neubig, G., 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys, 55(9), pp.1-35. https://dl.acm.org/doi/full/10.1145/3560815
White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J. and Schmidt, D.C., 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. https://arxiv.org/pdf/2302.11382.pdf
Min, S., Lyu, X., Holtzman, A., Artetxe, M., Lewis, M., Hajishirzi, H. and Zettlemoyer, L., 2022. Rethinking the role of demonstrations: What makes in-context learning work? https://arxiv.org/pdf/2202.12837.pdf
Qin, C., Zhang, A., Zhang, Z., Chen, J., Yasunaga, M. and Yang, D., 2023. Is ChatGPT a general-purpose natural language processing task solver? https://arxiv.org/pdf/2302.06476.pdf
OpenAI Prompt Engineering Guide. OpenAI, 2023

Name and workplace of master's thesis supervisor:

doc. Mgr. Viliam Lisý, MSc., Ph.D. Artificial Intelligence Center FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **23.01.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

doc. Mgr. Viliam Lisý, MSc., Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Chapter 1

Introduction

In the most recent years, we have witnessed significant advances in the field of natural language processing (NLP) and artificial intelligence (AI). These developments have culminated in the creation of large language models (LLMs), which are powerful deep learning architectures capable of achieving human-like performance across a wide range of tasks. One of the key skills in harnessing the full potential of LLMs is prompt engineering. This technique involves crafting textual inputs to maximize the effectiveness of these models. Despite the abundance of online resources offering guidance on prompt engineering, these recommendations often lack rigorous evaluation and are rarely compared against alternative approaches. The lack of systematic evaluation makes it challenging for researchers and practitioners to determine the most effective prompt engineering strategies for their specific use-cases.

In this work, we propose a comprehensive methodology for the evaluation of diverse prompt engineering recommendations sourced from various channels, mainly online blogs and articles, and demonstrate this methodology by evaluating four different prompt engineering recommendations. We will assess the effectiveness of these recommendations on a set of evaluation tasks from the text classification and multiple-choice question answering domains. The evaluation will be conducted on five state-of-the-art LLMs, chosen to represent a diverse set of models with varying sizes and capabilities: GPT-3.5, LLaMA 7B, LLaMA 13B, LLaMA 70B, and Gemini 1.0 Pro.

1.1 Thesis outline

In Chapter 2, we introduce the fundamental theoretical concepts that will be built upon in the following chapters, including an overview of the transformer architecture, large language models, and prompt engineering. Chapter 3 will present the evaluation methodology developed for this work, including the selection of prompt engineering recommendations, evaluation tasks, models, and model hyperparameters. Next, in Chapter 4, we will detail the technical implementation of the evaluation framework, focusing on integration with the model APIs, the individual steps in the experiment execution and evaluation, and the statistical methods used during the result analysis. Chapter 5 will present the results of the conducted experiments, analyze the findings, and discuss the implications of these results for the field of prompt engineering. Finally, in Chapter 6, we will summarize the key findings of this work, discuss the limitations of the study, and outline the possible directions for future research on the topic.

Chapter 2

Theoretical foundations

This chapter provides an overview of the theoretical foundations that underpin the subsequent experimental work. It covers essential concepts related to transformer architecture, large language models, prompt engineering, and a selection of prompt engineering recommendations.

We expect the readers to be familiar with the basics of neural networks, specifically feed-forward neural networks and recurrent neural networks, and their training. It is also assumed that readers possess a fundamental understanding of key concepts in the domain of natural language processing. For a more comprehensive overview of these topics, we recommend consulting the books *Deep Learning* by Goodfellow et al. [1] and *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* by Jurafsky et al. [2]. These concepts, while foundational to the field, are not directly relevant to the core subject of this thesis, so their detailed explanation is omitted in this chapter.

2.1 Transformer architecture

This section provides an overview of the transformer architecture, which serves as the foundation for large language models.

2.1.1 Pre-Transformer architectures (RNNs, LSTMs)

Before the advent of transformer architectures, Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks were prevalent in NLP. However, these architectures had significant limitations, particularly in handling long-range dependencies and computational inefficiency due to their sequential processing nature.

Transformers were introduced in 2017 by Vaswani et al. [3] as a solution to these shortcomings. Unlike their predecessors, transformers process data in parallel, significantly improving computational efficiency. This architecture's ability to handle long sequences more effectively without losing contextual information marked a substantial advancement over RNNs and LSTMs.

2.1.2 Architecture and training

A key part of the transformer architecture is the self-attention mechanism. This mechanism allows the model to weigh the significance of different parts of the input data, providing a more nuanced understanding of the context. Self-attention contributes significantly to the ability of transformers to generate more contextually relevant and coherent responses.

The training process for transformer architectures involves multiple layers of self-attention and feed-forward neural networks. These models are typically trained on vast datasets, enabling them to learn a wide array of language patterns and nuances. The training process is resource-intensive but results in models that exhibit a remarkable understanding of language and context.

2.2 Large language models

Large language models are a class of deep learning models that are primarily based on the transformer architecture. These models are distinguished by their immense scale, both in terms of the size of their training datasets and the complexity of their architecture. The transformer architectures enables LLMs to process and generate human-like text by learning a conditional probability distribution over sequences of words. This means that LLMs can

predict the likelihood of a word occurring in a text given the preceding and following words, allowing for the generation of coherent and contextually relevant language outputs.

A **prompt** in the context of LLMs is essentially an input text provided to the model, which serves as a starting sequence or context for the model to generate text. The prompt acts as an initial set of conditions for the probability distribution, guiding the model on how to continue the text. By adjusting the prompt, users can influence the direction and nature of the generated text, making prompts a crucial aspect of interacting with LLMs.

LLMs stand out primarily for their ability to understand and generate language with a high degree of fluency and context awareness, thanks to their ability to process extensive sequences of text. LLMs operate by processing text input and predicting subsequent text based on the learned conditional probability distribution. The model evaluates the context provided by the prompt and any preceding text to predict the most likely next word or sequence of words. Sampling methods, such as temperature-based sampling, are used to control the predictability and diversity of the output. A lower temperature results in more predictable and conservative text generation, while a higher temperature encourages creativity and variability in responses. This process enables LLMs to produce responses that are not only relevant to the given context but also varied and nuanced.

The trajectory of LLM development points toward continued scaling and the integration of multimodal capabilities. The performance of these models tends to improve with size, following the scaling law, though with diminishing returns [4]. Additionally, the exploration of models that can understand and generate not just text but also images, audio, and other data types, represents an exciting frontier in AI research.

2.2.1 Training process

The training process of LLMs generally involves several distinct stages, though the specific steps can vary depending on the model:

1. **Pre-training phase:** The first phase involves training the model on a large and diverse corpus of text data in an unsupervised manner. This process uses a transformer-based language modeling approach to learn a wide range of language patterns, grammar, and contextual understanding

from unlabeled text data. The goal is to develop a strong foundational understanding of natural language. After this stage, the model can handle various tasks such as language modeling, summarization, translation, and sentiment analysis. The resulting models are often referred to as *base models*, *foundation models*, or *completion models*. During pre-training, different variants of stochastic gradient descent, such as Adam [5], are used to update the model weights.

2. **Fine-tuning phase:** In this phase, models are often tailored to specific downstream tasks or domains, such as conversation or chat. This involves training the model in a supervised fashion on a more specialized dataset crafted by humans to be relevant to the desired application. This further optimization of the model parameters enhances its performance in those specific use-cases. Often, additional post-training techniques like Reinforcement Learning from Human Feedback (RLHF) [6] are used. RLHF involves training the model with reinforcement learning where human feedback is used to reward desirable outputs. This process involves the following stages:

   a. **Reward model training**: Human evaluators rate model outputs based on quality. These ratings are used to train a reward model that scores outputs.

   b. **Proximal policy optimization (PPO)**: The scores from the reward model are used for further fine-tuning through reinforcement learning, specifically using the PPO algorithm to maximize the reward.

   This further enhances the model's ability to produce accurate, appropriate, and contextually relevant responses. However, not all models use RLHF; some might use other post-training techniques or none at all. It is important to note that there are many other LLMs developed by various organizations, each with their unique features and training methodologies.

The effectiveness of an LLM in understanding and generating language highly depends on the quality and diversity of its training dataset. Models are generally trained on a mixture of publicly available text data and proprietary sources, aiming to cover a wide range of languages, topics, and styles. However, the representation of less common languages and dialects can be a challenge, leading to potential biases or limitations in the model's performance in those contexts. Continuous efforts are made by researchers to improve the inclusivity and representativeness of training data in LLM development.

2.2.2 Selection of LLMs

In the rapidly evolving landscape of LLMs, numerous models stand out for their groundbreaking capabilities, scalability, and innovative training methodologies. This section focuses on a selection of several LLMs developed by leading organizations in the field, including OpenAI's GPT series, Meta AI's LLaMA, and Google's Gemini model family. The primary reason for focusing on these models is their relevance to the experiments conducted in this thesis. These models were chosen to span a diverse selection of sizes, capabilities, and training approaches, to ensure a comprehensive exploration of the state-of-the-art in generative AI in the experimental part of this work.

1. **GPT models:** The models from GPT series, notably GPT-3.5 and GPT-4, developed by the scientists from OpenAI organization, have demonstrated significant advancements in understanding and generating complex language constructs [7, 8]. The models from this series feature an increasing number of parameters, which correlates with enhanced reasoning capabilities and a reduction in errors like hallucinations. The training process for these models includes extensive pre-training on diverse datasets, followed by supervised fine-tuning and, in the case of conversational models, further refinement using techniques like RLHF [8].

2. **LLaMA models:** LLaMA is another language model developed by Meta AI (formerly known as Facebook AI). It showcases capabilities akin to the GPT series but with distinct variations in training methodologies and applications [9]. Similarly to GPT, LLaMA models are also pre-trained on a diverse and large text corpus using unsupervised learning. For conversational LLaMA models like LLaMA-2-chat, an additional supervised fine-tuning step is performed on dialogue-specific datasets. Unlike the GPT models, the models from LLaMA family have been made partially open-source, allowing for broader research and experimentation. They are available in various sizes, ranging from 7B to 175B parameters, which makes them suitable for different computational needs and use-cases. The smaller models are generally faster and require less computing power, making them more accessible for experimentation.

3. **Gemini models:** The Gemini model family was announced in 2023 as Google's competitive response to OpenAI's GPT-4. The Gemini models, including the notable Gemini 1.0 Pro, have been designed to offer advanced capabilities in understanding different forms of data. Specifically, Gemini Pro is a multimodal model that can can accept text, image, and video inputs and generate text output. The models from Gemini family achieved state-of-the-art performance across a wide range of multimodal benchmarks [10]. The exact details of the training process

for Gemini models are not known, but similarly to the previous models, they are expected to involve large-scale pre-training and fine-tuning on specialized datasets.

2.2.3 Applications of LLMs

Large language models have numerous applications across various fields, leveraging their capability to understand, generate, and interact with human language. The key use-cases include:

1. **Text classification:** LLMs can categorize text into predefined classes, helping in tasks like sentiment analysis or medical report classification. For instance, in sentiment analysis, LLMs can classify customer reviews as positive, negative, or neutral, helping businesses gauge customer satisfaction.

2. **Summarization:** LLMs can distill long texts into concise summaries. This is particularly useful in contexts where the practitioners need to quickly grasp the essence of large volumes of information. For example, researchers can benefit from automatic summarization of academic papers, enabling them to efficiently review literature and identify relevant studies.

3. **Question answering and chatbots:** One of the most impactful applications of LLMs is the development of chatbots and question-answering systems. In organizations, chatbots can assist employees by providing quick answers to common questions, facilitating knowledge sharing and information access. They can also enhance the customer service by offering instant responses to customer inquiries, troubleshooting issues, or providing product information. These use-cases are often coupled with the implementation of retrieval-augmented generation (RAG) pipelines [11]. In RAG pipelines, relevant context is dynamically retrieved from internal documentation or knowledge bases based on the user's query and included as extra context in the input prompt to the model.

4. **Content generation:** LLMs can efficiently generate various types of text, such as articles, marketing materials, and code. Their ability to produce coherent and contextually relevant text makes them very valuable for creative but also technical writing tasks. Additionally, LLMs are effective in proofreading and editing existing content, helping to correct grammatical errors, improve style, and enhance overall readability. This is particularly useful for non-native English speakers who seek to refine their written communication. For example, in this master thesis, GPT-4

was used via the ChatGPT[1] interface to fix mistakes and reformulate existing texts, ensuring fluency and accuracy in English. It is crucial to always carefully inspect the outputs produced by LLMs, as they can sometimes generate hallucinations — statements that appear true but are factually incorrect. Moreover, LLMs often tend to produce generic text that lacks substance. Therefore, proactive checking and strict inspection of the generated output, an approach taken also while writing this thesis, is essential to maintain the quality and reliability of the content.

As research and development continue, we expect the set of applications to expand, with LLMs playing an increasingly important role in various industries and domains of our daily lives.

2.3 Prompt engineering

Prompt engineering refers to the practice of designing and optimizing input prompts to elicit the most suitable responses from LLMs. This field has evolved significantly in recent years, mirroring the advancements in LLMs themselves. It was shown that the quality and structure of the input prompt can significantly influence the quality of the generated result [12].

There are two main types of prompts:

- **System prompt:** This is the initial setup or context provided to the model, often designed by the developers or system designers. It sets the stage for the model's operation and can influence its general behavior and output style. As an example, in a customer service chatbot application, a system prompt might be, "As a customer service assistant, your goal is to provide helpful and accurate information in a friendly manner." This prompt sets a tone and behavior expectation for the model, guiding its responses to align with the intended customer service objectives. It is important to note that some models, like Gemini Pro, do not support system prompts, and the user must provide the entire context for the model in the user prompt.
- **User prompt:** On the other hand, user prompts are those provided by the end-users. These prompts are more dynamic and can vary greatly,

[1] https://chatgpt.com/

requiring the LLM to adapt and respond accurately to a wide range of requests and queries. As an example, in a customer service chatbot application, the user might ask a customer service chatbot, "Can you help me track my order that was supposed to arrive today?" This prompt is a typical inquiry that customers might direct towards a customer service assistant, prompting the model to generate a response that guides the user on how to track their order or provide an update on the order status.

Effective prompt engineering is crucial for maximizing the quality of the outputs generated by LLMs. As of now, a plethora of online resources offer guidance on this practice. However, the quality and effectiveness of these recommendations vary, which creates a need for a thorough evaluation to establish best practices in prompt engineering.

2.3.1 Prompt template syntax

In this thesis, prompt templates are presented in Jinja2 format, which is also the format used for defining the prompts in the experiments. Jinja2 is a templating and rendering engine used for writing and managing parametrized textual templates, and it allows for the inclusion of variables, control structures, and other dynamic content.

The syntax for these templates follows the standard Jinja2 conventions, such as using double curly braces `{{ }}` for variables and control structures like `{% %}` for loops and conditionals.

Additionally, to enhance readability and indicate where lines are broken due to the width of the document page, a red arrow symbol (↪) is used.

Here is an example of a simple prompt template that showcases these features:

```
This is some very long text that instructs the model to
    ↪ perform a specific task. The task is to choose the
    ↪ correct answer to the question from the provided
    ↪ choices.

Question: {{ question }}

{% for label, text in choices.items() %}
```

```
{{ label }}: {{ text }}
{% endfor %}
```

This template includes two variables, `{{ question }}` and `{{ choices }}`, to display the question and a loop to list the possible choices dynamically. The choices are provided as a dictionary mapping labels (A, B, C, D) to text of the choice, and we use the `{% for %}` loop to iterate over the dictionary items and display the labels and corresponding text for each choice.

2.3.2 Existing prompt engineering recommendations

This section delves into various existing recommendations for prompt engineering and provides illustrative use-cases associated with each type of reframing technique. While certain techniques like chain-of-thought prompting introduced by Wei et al. [13], and few-shot prompting introduced by Brown et al. [7], have already gained scientific validation and demonstrated their effectiveness, this section focuses on a different set of recommendations. We will be mostly concerned with recommendations that were popularized through different internet articles and blogs and since adopted by many practitioners, but lack rigorous scientific evaluation. It is important to note that the recommendations below are most commonly given in the context of working with models from the GPT family.

1. **Instruction context separation:** A prevalent recommendation is the use of delimiters to distinctly separate different parts of the input [14, 15, 16]. This technique involves using specific symbols like triple hash `###` or triple quotes `"""` to delineate the instruction from the context of the prompt. The underlying rationale is to provide a clear structure and organization to the prompt, potentially helping the model to better understand and respond to the prompt. This recommendation is most commonly given in the context of working with models from the GPT family.

   a. **Without separation:**

```
Please write a summary of the article, focusing on
    ↪ the main points and conclusions.

This is an example article.
```

b. **With triple hash (###):**

```
Please write a summary of the article, focusing on
    ↪ the main points and conclusions.

###

This is an example article.
```

c. **With triple tuotes ("""):**

```
Please write a summary of the article, focusing on
    ↪ the main points and conclusions.

"""

This is an example article.

"""
```

2. **Positive vs. negative formulation:** Another commonly cited tip is the preference for positive over negative formulations in prompt construction [14, 16, 17, 18, 19]. This approach suggests that instead of stating what the model should not do, it is more effective to clearly state what it should do. This recommendation stems from the belief that positive instructions are more direct and easier for the model to interpret and execute.

a. **Positive formulation:** This version positively states what the model should do.

```
Please write a summary of this article. Make sure to
    ↪ focus solely on the main points and conclusions
    ↪  of the article.

This is an example article.
```

b. **Negative formulation:** This prompt specifies what the model should not do.

```
Please write a summary of this article. Make sure to
    ↪ not include unnecessary details or diverge from
    ↪  the article's main points.

This is an example article.
```

3. **Instruction itemization:** This recommendation advises to turn long paragraphs with instruction into lists, converting individual sentences into bulleted statements [19, 18]. It is believed to help in presenting the instructions in a clear, structured manner, enabling the model to follow each requirement with equal attention. One rationale for applying this recommendation is the belief that LLMs cannot effectively follow long paragraphs stating multiple requirements due to first instruction bias (paying more attention to the beginning of the paragraph than the rest of it) [19].

   a. **Paragraph instruction:** Instructions are provided in a single paragraph.

```
Write a summary of the article by identifying the
    ↪ main points, highlighting the conclusions, and
    ↪ ensuring brevity and clarity.

This is an example article.
```

   b. **Bulleted instructions:** Instructions are itemized into bulleted points.

```
Write a summary of the article, following these
    ↪ guidelines:

- Identify the main points
- Highlight the conclusions
- Ensure brevity and clarity of the summary

This is an example article.
```

4. **Language correctness:** This recommendation explores the impact of grammatical and stylistic correctness on the response quality of a LLM. Incorrect grammar, inappropriate tense usage, incorrect punctuation, and awkward phrasing are believed to potentially degrade the quality of the model's responses. This perspective is supported by various online articles that mention the sensitivity of LLMs to the linguistic quality of the input [17, 20, 21].

   a. **Correct text:**

```
Select the correct answer by choosing the appropriate
    ↪  label. Each question has a single correct
    ↪ answer. The possible labels are, for example, A
    ↪ , B, C, D, E. Your response should contain only
    ↪  the letter corresponding to the correct answer
    ↪ .
```

```
This is an example question with some choices.
```

b. **Text with errors:**

```
select the corrected answer by choosing the
    ↪ appropiate label. every questions has a single
    ↪ correct answer. The possibly labels are for
    ↪ example A, B, C, D, E. your response should
    ↪ contains only the letter correspond to the
    ↪ correct answer.

This is an example question with some choices.
```

- **Errors:**
  - "select" instead of "Select" at the beginning.
  - "your" should be capitalized as it starts a new sentence.
  - "appropiate" instead of "appropriate".
  - "every questions" instead of "Each question", introducing a grammatical number disagreement.
  - "contains" instead of "contain", which introduces a verb agreement error.
  - "correspond" instead of "corresponding", which is a verb form error.
  - Omitted comma after "are" in the list of labels, which is a punctuation error.
  - Use of "every" which is less formal and less precise compared to "Each" in instructional content.

The degree of reframing (the amount of change applied to the original prompt) varies across the different recommendations, some requiring only minor algorithmic-like adjustments to the prompt (e.g., instruction context separation) while others require a complete reformulation of all instructions (e.g., positive vs. negative formulation). It's important to note that while these recommendations are widely discussed and applied within various online communities, their impact has not been systematically studied or validated in scientific research. The subsequent sections of this thesis will critically evaluate these practices, exploring their effectiveness in practical applications with LLMs.

Chapter 3

Evaluation methodology

In this chapter, we present different aspects of the methodology used to evaluate the impact of prompt engineering recommendations on the performance of large language models. This includes both the key factors that influenced our choices in designing the experiments, and the specific decisions made regarding the selection of prompt engineering recommendations, benchmarks, models, model hyperparameters, and more.

As already mentioned in the previous chapters, this work focuses on recommendations that were popularized through different internet articles and blogs, but lack rigorous scientific evaluation. Moreover, we will be testing relatively simple recommendations that involve only lightweight changes to the original prompt, and where the prompt reframing is quite algorithmic. Our focus will therefore be the evaluation of the four prompt engineering recommendations described in detail in Section 2.3.2. These four recommendations are:

1. **Instruction context separation:** This recommendation involves using delimiters like `###` or triple quotes `"""` to separate the instruction from the context.
2. **Instruction itemization:** Advises the conversion of instructions from long paragraphs into bulleted lists.
3. **Positive vs. negative formulation:** Suggests the use of positive formulations in prompt construction over negative ones.
4. **Language correctness:** Emphasizes the importance of using correct grammar, punctuation, and spelling in prompts.

The selected recommendations serve as the basis for our experiments, and drive the rest of the experiment design decisions.

3.1 Evaluation tasks

The first step in our experimental setup is to identify the suitable task domains and select the benchmarks that will be used to evaluate the impact of the prompt engineering recommendations.

When selecting a task domain and the specific datasets for evaluating LLMs in the context of prompt engineering, we considered several criteria:

1. **Textual input and output:** The core functionality of LLMs revolves around their ability to process and generate text. Thus, the selected benchmarks must inherently support textual inputs and outputs. This requirement automatically excludes tasks like image classification, audio processing, or any other domain that relies on processing non-textual data.
2. **Task difficulty:** While selecting benchmarks, it's crucial to balance the complexity of the tasks. Given our emphasis on examining the impact of subtle, algorithmic variations in prompts, we will opt for tasks that are relatively simple. The rationale behind this choice is that it ensures that the tasks allow for minor prompt modifications to have a distinguishable impact. Our aim is to prevent small variations in prompt engineering from being overshadowed by the complexity of the task.
3. **Evaluation metric:** Identifying an appropriate metric is key to evaluating and comparing the outcomes of our experiments. In this work, we prioritize metrics that are easily computable and directly attributable to the changes in prompt design. One example of such metric would be accuracy - a straightforward, computable metric suitable for many text classification tasks. On the other hand, more sophisticated evaluation metrics, such as using LLMs as judges (LLM Eval) [22] or human evaluation, are less favorable in this context due to their inherent variability and the complexity involved in their computation.
4. **Dataset size:** The dataset should be large enough to provide a representative sample for statistically robust evaluation.
5. **Dataset availability:** The dataset should be publicly available and well-documented to ensure reproducibility.

In this work, we decided to focus predominantly on different tasks from the text classification domain, given their relative simplicity and ease of evaluation compared to other NLP tasks, such as text generation. Specifically, we will be working with sentiment analysis and multiple-choice question answering tasks. These tasks involve assigning predefined labels to text, which will allow us to use clear and easily interpretable evaluation metrics like accuracy.

3.1.1 Selected datasets

The three benchmark datasets selected for our study are:

1. **Twitter US Airline Sentiment Analysis**[1] **(TWITTER):** This dataset is a collection of tweets about major U.S. airlines from February 2015, where contributors classified the sentiment of tweets as positive, negative, and neutral. The dataset is publicly available on Kaggle as both a CSV file and an SQLite database. We selected this classification task for its simplicity and its popularity within the NLP community.

2. **AI2 Reasoning Challenge**[2] **(ARC):** Unlike the first dataset focused on sentiment analysis, the ARC dataset developed by Clark et al.[23] offers a different challenge in the form of multiple-choice science questions. The questions are derived from grade-school level exams, and the ARC dataset is partitioned into easy and hard questions. In our experiments, we will be working only with the questions from the latter category. The ARC dataset has been previously used to benchmark models such as GPT-4 and Llama 2, making it a suitable choice for our evaluation [24, 9].

3. **Cosmos QA**[3] **(COSMOS):** Cosmos QA is a dataset developed by Huang et al. [25] and consisting of 35.6K multiple-choice questions that require commonsense-based reading comprehension. It is distinguished by its focus on *reading between the lines* across a diverse range of everyday topics. Cosmos QA has been leveraged in the evaluation of LLMs and the exploration of various prompt engineering techniques in past research [19, 26], which again highlights its relevance in the LLM evaluation domain.

[1] https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment
[2] https://allenai.org/data/arc
[3] https://wilburone.github.io/cosmos/

3.2 Prompt design

After selecting a specific prompt engineering recommendation for testing and the evaluation benchmark, the next step is to design different prompt variations for the LLM. To simplify the complexity of the prompt design for the experiments, we will exclusively focus on modifications to the user prompt. The rationale behind this is that the system prompt is often static or preset in many real-world applications. Especially in web interface interactions with chat models, end-users typically do not have access to modify the system prompt. Therefore, our decision to focus on user prompt modifications should not significantly diminish the relevance or applicability of our results. Moreover, we use the same generic system prompt across all experiments, and add all task-specific instructions to the user prompt. This decision aims to reduce the impact of the system prompt on the final results, ensuring that any observed differences are due to the user prompt modifications.

When designing user prompt variations, we start with a *baseline* prompt and refine it through several iterations. During this process, we evaluate the model's performance using the baseline prompt and a chosen evaluation metric, noting common error patterns in the outputs. This helps us make minor, targeted adjustments to the prompt, adding specific instructions to address these errors. For example, in a classification task, if we observe consistent misclassifications, we tweak the baseline prompt to improve accuracy in these cases. We ensure the baseline prompt remains relatively simple to avoid skewing the experiment's results, and to allow the modifications in the *studied* prompts variations to be more pronounced.

Starting from this baseline, we then develop one or more *studied* prompts that include modifications reflecting the tested recommendation. These prompts are constructed to explore different aspects of the recommendation. For instance, in the case of the *instruction context separation* recommendation, we might design prompts that separate the instructions from the context in various ways, such as using triple hashes (`###`) or enclosing the context within triple quotes (`"""`).

It is worth noting that we use a zero-shot learning approach, where no examples are provided to the model within the prompt. This approach ensures that any observed effects are due to prompt modifications and the results are not skewed by the use of other prompt engineering techniques.

Additionally, it's important to mention that the user prompt often includes specifications regarding the desired output format. For instance, in text

classification tasks, the model is directed to produce a single class label. This instruction simplifies response parsing and further emphasizes the influence of prompt modifications on the model's output.

The user prompts that were designed for each experiment are displayed in full detail in the following Chapter 5.

3.3 Model selection

The choice of LLMs is another critical aspect of our experimental setup. The models selected for evaluation should be representative of the current state-of-the-art in the field of generative AI. The aim to test the prompt engineering recommendations across a diverse set of models, including models of different sizes, of different architectures and developed by different companies.

Naturally, the models must also be accessible and available for use in our experiments. Currently, there are two main ways of interacting with large language models:

1. **Deploying an open-source model locally or on a remote server:** This is the most flexible option that offers more customization, but it requires a significant amount of computational resources, specialized hardware, and it is more complicated to set up. Moreover, it is not always possible to use this option, as some models, like GPT-3.5 and GPT-4, are not available open-source.
2. **Accessing the model via a third-party API:** This option is more straightforward and requires less computational resources, but it is less flexible and more expensive. There are several LLM providers that host deployed LLMs and offer interacting with them via an API. These include OpenAI, Anyscale, and all major cloud providers (Google Could Platform, Amazon Web Services and Microsoft Azure).

In our experimental part, we opted for the latter category (API-available models), because of the convenience of setup and model availability. The use of models via third-party provider APIs typically requires registration on the provider's website and obtaining the access credentials, typically in the form of API keys (e.g., OpenAI, Anyscale) or service account credentials (e.g., Google Cloud).

This decision also introduces several potential limitations on the extent of our experimentation, notably:

- **Rate limits:** API providers almost always impose rate limits on the number of requests that can be made or number of tokens that can be consumed within a given time frame. This constraint motivates careful planning and implementation of our communication with these APIs to ensure robust and reliable experiment execution, even when facing rate limits. The following chapter will dive in more detail into the strategies for managing these rate limits effectively, allowing for the extensive and reliable execution of experiments despite this restriction.
- **Latency:** Another important limitation is caused by the nature of communication with these third-party APIs. Evaluating each of the samples requires a network round-trip, which often takes hundreds of milliseconds or even seconds, depending on the model and the load on the provider's servers.
- **Cost:** The use of third-party APIs can be costly, as all requests are billed based on the number of tokens consumed. The costs are typically computed per thousand tokens, and can vary significantly between different providers and LLMs.

Given that in order to obtain statistically significant results we need to execute a relatively high number of LLM calls, we have to be mindful of these limitations. As a consequence, we will generally work only with a smaller subset of the original dataset. The strategy for selecting this subset is described in Section 3.4.

It is important to note that in the scope of this work, we will work exclusively with models tuned for chat completion tasks, rather than foundational completion models. This decision is motivated by the facts that chat models are more commonly used in real-world applications, are more representative of the current state-of-the-art in the generative AI domain, and are more often available via third-party APIs.

3.3.1 Selected LLMs

The LLMs we will evaluate in our experiments are as follows:

1. **GPT-3.5 Turbo (gpt-3.5-turbo-0613)**
2. **LLAMA models:**
    a. LLAMA 7B (meta-llama/Llama-2-7b-chat-hf)
    b. LLAMA 13B (meta-llama/Llama-2-13b-chat-hf)
    c. LLAMA 70B (meta-llama/Llama-2-70b-chat-hf)
3. **Gemini Pro (gemini-1.0-pro)**

For more information on these models and their architecture, please refer to Section 2.2.2. It is important to mention that we needed to implement a robust and resilient communication layer on top of these platforms. This not only accommodates the diverse communication requirements of each provider but also addresses potential issues related to API rate limits and network communication, and provides a unified interface for our evaluation framework. The implementation of this communication layer is elaborated upon later in Chapter 4.

3.3.2 Hyperparameter selection

Choosing the right parameters for the models is key to getting reliable results. For our experiments, we work with two main hyperparameters: temperature and max tokens.

1. **Temperature**: We set this to 0 for all of our experiments, as we want the model's answers to be consistent rather than random. While a higher temperature can make the model's responses more varied and creative, which could be useful for some tasks, our experiments, like figuring out the most likely class label in text classification, need precise and predictable outcomes.
2. **Max tokens**: In general, we set the max tokens parameter to a relatively low value consistent with our tasks' nature. For text classification tasks, where the model is generally instructed to output a single label, we could in theory set this to 1. However, we've observed that some models, specifically LLAMA 7B and LLAMA 13B, sometimes do not strictly adhere to instructions regarding output format, opting instead to generate complete sentences. Therefore, we allow for a longer token output to ensure enough buffer for the generated output to include the class label. We then use robust parsing algorithms to extract the predicted classification label from the model's response.

3.4 Execution and evaluation

The execution phase involves running all the LLMs on a subset of the benchmark dataset using all the designed prompt variations. This subset is determined by random sampling from the original dataset, with a fixed random seed to ensure the reproducibility of the experiments. The responses generated by the models are then compared against the ground truth, using the benchmark-defined metric. Given our focus on text classification tasks, the evaluation metric we will be dealing with is accuracy. The exact process of LLM request execution and result evaluation is described in more technical detail in Section 4.2.

The implemented evaluation framework also incorporates the computation of confidence intervals and statistical significance of the results. This approach not only guides our understanding of the reliability of the results but also helps in selecting an optimal sample size for our experiments. Specifically, the number of samples in individual experiments is chosen in an iterative way. First, we run the experiment with a smaller number of samples (e.g., 3000), and obtain preliminary results. We inspect the results to determine if there is any observable trend or significant difference between the prompt variations. If there appears to be any signal in the data, we proceed with a larger sample size (e.g., 6000), determined by the confidence intervals and chosen so that the results are statistically significant. This iterative approach ensures that we do not waste computational and financial resources on experiments that do not have potential to yield meaningful results. In some cases, like for the ARC dataset described in Section 3.1.1, the number of samples is constrained by the dataset size ($\pm$2500 samples), therefore we always use the entire dataset for evaluation. In other cases, like for the Gemini Pro model, we need to limit the number of samples due to the high latency and very strict rate limits imposed by the API provider. The exact details of the statistical methodology used to compute confidence intervals and determine statistical significance will be described in Section 4.6 of the following chapter.

Chapter 4

Implementation

In this chapter, we provide a detailed overview of the technical implementation of our evaluation framework. This includes the specific technologies and libraries used, details on the integration with LLM API providers, and different technical aspects of the evaluation framework implementation such as experiment tracking and caching.

4.1 Languages and tools

The implementation of our evaluation framework is entirely done in Python, version 3.11. Python's popularity in the machine learning space, as well as the extensive availability of specialized libraries, makes it an ideal choice for this work. We utilized several technologies and libraries to facilitate our experiments:

- **OpenAI API:** This API is used to interact with the ChatGPT family of models. To interact with the OpenAI API, we utilized the OpenAI Python SDK, which provides a convenient interface for making requests to the API.
- **Anyscale Endpoints:** A managed service that allows interactions with LLaMA models. Its API is compatible with the OpenAI Python SDK, enabling us to interact with both APIs using the same code.

- **Google Cloud Vertex AI:** A suite of machine learning tools that provides access to a wide range of models, including the LLM models available on Google Cloud. To interact with Vertex AI service we used the Google Cloud SDK - a Python library that simplifies the interaction with different Google Cloud services.
- **Pydantic:** A data validation and parsing library that simplifies the handling of complex data structures, including their validation, serialization, deserialization and more.
- **Jinja2:** A templating and rendering engine utilized for writing and managing parametrized textual templates. We use Jinja2 to provide a flexible and clear way to define prompts templates that can adapt to different inputs.
- **Tenacity:** A Python library for retrying operations with configurable backoff and retry strategies. This library is essential for handling transient network issues and rate limits when interacting with LLM APIs.
- **Huggingface datasets:** A library that provides easy access to a wide range of datasets for natural language processing tasks. This library was chosen to download and preprocess some of the datasets used in our experiments.

4.2 Experiment execution

To handle the high volume of requests to LLM APIs efficiently, we designed an optimized execution and evaluation strategy that uses smart concurrency and batching to optimize the throughput and minimize the execution time.

Initially, a subset of the input dataset of given size is selected through random sampling. A sampling strategy with a fixed random seed ensures reproducibility of the experiments by generating a consistent set of samples. Using the python asyncio library, we evaluate the samples for different models concurrently by sending requests to the different LLM APIs in parallel. For example, if the experiment involves evaluating GPT-3.5 and LLAMA 7B models, the framework will send requests to both APIs concurrently, allowing the models to process the samples simultaneously, as the rate limits imposed by the two providers are independent of each other. Without this optimization, the execution time would be significantly longer, as the request for different models would be evaluated sequentially. Furthermore, for each model, the evaluation of the input samples for the model is carried out by introducing another level of concurrency. The samples are split to batches, taking into

account how strict are the rate limits of the particular API provider on the number of tokens and requests that can be processed in a given time frame. The samples in each batch are then sent to the model for processing in an asynchronous fashion, again using the asyncio library. For example, the rate limit on the requests per second of the GPT-3.5 is much less strict than that of the Gemini Pro model, so the framework will fire 10 concurrent requests to GPT-3.5 and only 5 concurrent requests to Gemini Pro at the same time. This asynchronous execution strategy allows us to speed up the network-bound communication, which is often a bottleneck in applications of this nature.

After each batch of samples is processed, and we receive the model's responses for that batch, the responses are collected and evaluated against the ground truth labels provided in the dataset. The evaluation framework categorizes the results into correct, incorrect and invalid categories. Invalid responses are those where the model's output does not match the expected format, while correct and incorrect responses are determined by comparing the model's predictions to the ground truth labels.

The aggregated results for each model and prompt combination are then analyzed to compute the final accuracy for the given model and prompt variation. In this step, we also calculate the confidence intervals and statistical significance to gain insights into the reliability of the results. The results along with various other metadata are then stored in a structured json format for further analysis and comparison.

4.3 LLM provider API integration

Our experimental framework is designed to interact with multiple platforms providing access to the LLMs listed in Section 3.3.1.

1. **GPT-3.5 Turbo**: This model is accessed via the OpenAI API[1]. To interact with this API, an API key is created, and the OpenAI Python SDK is used for communication.

2. **LLAMA models:** The models from LLAMA family are accessed via Anyscale Endpoints[2], a managed service that allows interactions with multiple different open-source LLMs.

[1] https://platform.openai.com/docs/overview
[2] https://www.anyscale.com/

3. **Gemini Pro**: This model is accessed through Google Cloud's Vertex AI[3] service. Setting up this interaction requires creating a Google Cloud account, enabling the Vertex AI API, and generating IAM service account credentials for our Python evaluation framework script.

For each of the LLM providers, we implemented a separate class that encapsulates the logic for interacting with the respective API. This design ensures that the codebase is modular and extensible, allowing for easy integration of new LLM providers in the future. Moreover, each of the implementations contains custom error handling and retry mechanisms tailored to the specific rate limits and error codes of the respective API. By utilizing an exponential backoff strategy for specific errors, such as network issues or rate limit breaches, we ensure maximum throughput under given rate limits for our experiments.

4.4 Caching strategy

The motivation behind implementing a smart caching mechanism is clear when considering the scale and potential costs of our experiments. For instance, an experiment involving 1000 samples, 3 prompt variations, and 5 different models would require 15,000 external API calls. Such an undertaking, despite optimizations, can take hours to execute and incur significant costs ranging up to tens of dollars.

The risk of losing progress due to unexpected errors, especially towards the end of an experiment run, calls for a more scalable approach. To mitigate these risks, we've implemented a sophisticated caching mechanism. This strategy ensures that results from previously executed LLM calls can be retrieved from cache, eliminating the need to repeat identical calls in subsequent experiment runs. Such a system is not only cost-effective but also greatly reduces the time required for experiment runs, ensuring progress is not lost due to transient failures or manual interventions. This implementation uses the Python *pickle* library for serialization and deserialization, storing function results on a local file system cache based on unique keys generated using a hash function ran on the function names and arguments. This not only accelerated development by reusing results from previous identical experiment runs but also reduced costs by eliminating the need for repeated calls.

[3] https://cloud.google.com/vertex-ai

4.5 Experiment definition and tracking

All the experiments were conducted in a local development environment. A key part of our evaluation framework is the detailed mechanism for defining and tracking experiments. The individual experiment definitions are stored as YAML files, each with a unique name. The specific file with the definition of the experiment is loaded at the startup of the experiment script, determined by a parameter passed by the user. This approach ensures reproducibility and simplifies the process of adding new experiments to the framework in a straightforward and declarative manner.

An example of such an experiment definition file is given below:

```
name: "Instruction itemization ARC multiple choice QA"
experiment_type: "INSTRUCTION_ITEMIZATION"
task_type: "ARC_MULTIPLE_CHOICE_QA"
models:
  - "meta-llama/Llama-2-7b-chat-hf"
  - "meta-llama/Llama-2-13b-chat-hf"
  - "meta-llama/Llama-2-70b-chat-hf"
  - "gpt-3.5-turbo-0613"
  - "gemini-1.0-pro"
prompts:
  system: "system.jinja2"
  user:
    - name: "Baseline"
      file: "ii_arc_baseline.jinja2"
    - name: "Itemized"
      file: "ii_arc_itemized.jinja2"
```

This YAML structure allows to specify all the experiment parameters, including the experiment name, type, involved models, and the files containing Jinja templates for the system and user prompts.

For tracking and analysis purposes, detailed metadata about each experiment run is stored on the local filesystem in json format when the experiment is completed. The experiment file contains information about the experiment's name, type, evaluation task, models used, prompts employed, and the individual result samples obtained from the LLMs and used for manual evaluation or error inspection. These metadata also include important metrics

such as the number of correct, incorrect, and invalid responses, total samples evaluated, accuracy, the percentage of invalid responses, and a confidence interval for the experiment's metric values. Recommended sample sizes for future experiments, calculated based on desired margins of error, are also recorded, allowing for informed decision-making for subsequent runs. Lastly, this file includes information about the total cost of the experiment, which is computed based on the number of tokens consumed by the LLMs and the respective token prices.

4.6 Statistical evaluation methodology

Because evaluating different prompt variations on LLMs is relatively expensive and time-consuming, we cannot run experiments on arbitrary numbers of samples and have to be more strategic when selecting the sample size. To establish a rigorous approach for determining the minimum number of samples necessary for our experiments, we employ statistical methods to compute confidence intervals for our accuracy metric, based on a preset confidence level of 95%.

The computation of the confidence interval involves determining the sample proportion (accuracy) and its standard error. The standard error calculation is specifically tailored for Bernoulli trials, making it suitable for binary outcomes like correct or incorrect classifications. The confidence interval is then derived using the sample proportion, standard error, and a z-score corresponding to the desired confidence level. This interval provides a range within which the true accuracy metric of the model is likely to lie with 95% confidence.

Moreover, the framework calculates the required sample size for an experiment, aiming for a specific margin of error, such as 1%. This computation uses the estimated proportion of success (initial accuracy), the desired margin of error, and the confidence level to determine the smallest sample size needed to achieve results within the specified accuracy range. This confidence interval is crucial for understanding the variability of the accuracy and ensuring that our experimental findings are reliable within a specific range of precision. Mathematically, the confidence interval is computed as follows:

$$CI = \hat{p} \pm Z_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

where $\hat{p}$ is the sample proportion, $Z_{\alpha/2}$ is the Z-score corresponding to the desired confidence level, and n is the sample size.

The formula for calculating the initial sample size is:

$$n = \frac{Z_{\alpha/2}^2 \hat{p}(1-\hat{p})}{E^2}$$

where E represents the desired margin of error, and the other symbols have the same meanings as in the confidence interval formula.

By implementing and using these statistical formulas, we ensure that any results obtained from our experiments are statistically significant and reliable. It is important to note that the methodology described above is only one of possible approaches to computation of confidence intervals and there are other, more advanced mathematical methods, that could yield even more precise results.

Chapter 5

Experiments

This chapter presents the experimental work carried out to evaluate the effectiveness of different prompt engineering techniques as outlined in the previous chapter. A single experiment in this study involves assessing multiple prompt variations across a set of LLMs from Section 3.3 and a specific evaluation task. The prompt variations are constructed based on the chosen prompt engineering recommendation and the evaluation task at hand, as described in Section 3.2.

In total, 12 experiments were conducted to comprehensively test the four recommendations listed in Chapter 3. Each of the four recommendations is tested on three different benchmarks listed in Section 3.1. Each main section below is devoted to a specific prompt engineering recommendation. Each subsection is then named after the dataset used in the experiment, and it includes the used user prompt templates and a table summarizing the results of the experiment.

The prompt templates use the Jinja2 syntax and contain different placeholders that are replaced with actual content of the input sample in each experiment. These template also contain different control structures, such as loops, that allow for the generation of more complex prompts based on the input sample. The experiment results for all models are depicted in a table that displays the accuracies achieved by each model with different prompt variations, and the confidence intervals for these accuracies given the confidence level of 95%. Each table then also includes information about the number of samples used in the experiment for individual models.

It is important to note that we had to constraint our experiments with the Gemini Pro model to a smaller number of samples due to the limitations of the model's API, such as very high response times and very strict rate limits on the number of requests per minute. As an example, a single experiment with only the Gemini Pro model on the Twitter dataset with three prompt modifications and 500 samples took over 12 hours to complete. To this end, we only evaluated the Gemini Pro model on the Twitter dataset with number of samples ranging from 500 to 1000, depending on the experiment and the preliminary results obtained. As a result, we would only be able to detect very large differences in performance for this model given the statistical power of the experiments.

As mentioned in Chapter 3, we developed a generic system prompt that was identical across all experiments:

```
Your task is to analyze the provided text and produce a
    ↪ response based on the instructions.
```

For models that do not support system prompt (Gemini Pro), we concatenated the content of the system prompt with the user prompt and used the only the resulting user prompt to generate the model's response.

All the experiments in this chapter were ran between 11th and 20th of May 2024 using the model APIs and versions detailed in Section 3.3.

5.1 Instruction context separation

This section describes the experiments conducted to assess the impact of instruction context separation on model performance. We focus on the use of two most commonly recommended ways of separation to distinguish the instruction from the context within the prompt.

5.1.1 TWITTER

Prompt templates

1. **Baseline:**

```
Classify the tweet below based on its sentiment into one
    ↪ of the three following classes: Positive, Negative
    ↪ or Neutral.
You must output only a single word that is the class
    ↪ label assigned to the tweet.

{{ sample_text }}
```

2. **Hash separator:**

```
Classify the tweet below based on its sentiment into one
    ↪ of the three following classes: Positive, Negative
    ↪ or Neutral.
You must output only a single word that is the class
    ↪ label assigned to the tweet.

###

{{ sample_text }}
```

3. **Triple quote separator:**

```
Classify the tweet below into one of the three following
    ↪ classes: Positive, Negative or Neutral.
You must output only a single word that is the class
    ↪ label assigned to the tweet.

"""

{{ sample_text }}

"""
```

Experimental results

Model	Samples	Baseline	Hash separator	Triple quote separator
LLAMA_7B	3000	63.1% ± 1.5%	64.4% ± 1.5%	63.7% ± 1.5%
LLAMA_13B	3000	77.4% ± 1.3%	78.0% ± 1.3%	76.0% ± 1.3%
LLAMA_70B	3000	78.9% ± 1.3%	78.9% ± 1.3%	78.9% ± 1.3%
GPT_35	3000	82.5% ± 1.2%	82.5% ± 1.2%	83.5% ± 1.2%
GEMINI_PRO	800	82.9% ± 2.6%	81.4% ± 2.7%	80.8% ± 2.7%

Table 5.1: Accuracies of LLM models for different prompt variations for the Instruction Context Separation experiment on the TWITTER dataset.

5.1.2 ARC

Prompt templates

1. **Baseline:**

```
Select the correct answer to the question by choosing the
    ↪ appropriate label (e.g., A, B, C, D, E).

Question: {{ input.question }}

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

2. **Hash separator:**

```
Select the correct answer to the question by choosing the
    ↪ appropriate label (e.g., A, B, C, D, E).

###

Question: {{ input.question }}

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

3. **Triple quote separator:**

```
Select the correct answer to the question by choosing the
    ↪  appropriate label (e.g., A, B, C, D, E).

"""

Question: {{ input.question }}

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}

"""
```

Experimental results

Model	Samples	Baseline	Hash separator	Triple quote separator
LLAMA_7B	2500	55.3% ± 1.9%	55.0% ± 2.0%	54.0% ± 2.0%
LLAMA_13B	2500	63.6% ± 1.9%	64.6% ± 1.9%	64.3% ± 1.9%
LLAMA_70B	2500	77.5% ± 1.6%	77.2% ± 1.6%	77.6% ± 1.6%
GPT_35	2500	82.6% ± 1.5%	81.5% ± 1.5%	80.5% ± 1.6%

Table 5.2: Accuracies of LLM models for different prompt variations for the Instruction Context Separation experiment on the ARC dataset.

5.1.3 COSMOS

Prompt templates

1. **Baseline:**

```
Select the correct answer to the question based on the
    ↪ information provided in the context by choosing the
    ↪  appropriate label (A, B, C, D).

Context: {{ input.context }}

Question: {{ input.question }}

Options:
```

```
{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

2. **Hash separator:**

```
Select the correct answer to the question based on the
    ↪ information provided in the context by choosing the
    ↪  appropriate label (A, B, C, D).

###

Context: {{ input.context }}

Question: {{ input.question }}

Options:

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

3. **Triple quote separator:**

```
Select the correct answer to the question based on the
    ↪ information provided in the context by choosing the
    ↪  appropriate label (A, B, C, D).

"""

Context: {{ input.context }}

Question: {{ input.question }}

Options:

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}

"""
```

Experimental results

Model	Samples	Baseline	Hash separator	Triple quote separator
LLAMA_7B	3000	63.6% ± 1.7%	63.6% ± 1.7%	60.8% ± 1.7%
LLAMA_13B	3000	68.2% ± 1.7%	68.3% ± 1.7%	67.1% ± 1.7%
LLAMA_70B	3000	81.3% ± 1.4%	81.8% ± 1.4%	81.2% ± 1.4%
GPT_35	3000	77.3% ± 1.5%	77.7% ± 1.5%	77.2% ± 1.5%

Table 5.3: Accuracies of LLM models for different prompt variations in the Instruction Context Separation experiment on the COSMOS dataset.

Results analysis

The results of the experiments evaluating the impact of the instruction context separation recommendation on different datasets show us several insights. Across all models and datasets, there were no statistically significant differences in performance between the baseline and either of the prompt variations (hash separator or triple quote separator). The confidence intervals overlap, which indicates that any observed changes in accuracy are within the margins of error. This suggests that instruction context separation recommendation, as implemented in the prompt templates, does not significantly affect the performance of any of the models.

5.2 Instruction itemization

This section describes the experiments conducted to evaluate the impact of instruction itemization on model performance. We created two prompt variations for each dataset, one baseline with instructions presented as a single paragraph and one studied variation with instructions itemized into a list.

5.2.1 TWITTER

Prompt templates

1. **Baseline:**

```
Given a tweet, your task is to classify its sentiment
    ↪ into one of three classes: Positive, Negative, or
    ↪ Neutral. Output only one of the following class
    ↪ labels that best fits the tweet's sentiment: "
    ↪ Positive", "Negative", "Neutral". Do not include
    ↪ any additional text or commentary in your response.
    ↪  For tweets containing offensive language, personal
    ↪  attacks, or inappropriate content, classify them
    ↪ based on the sentiment of the context. If a tweet
    ↪ does not contain clear sentiment or is primarily
    ↪ informational (e.g., about flight delays), classify
    ↪  it as "Neutral". Keep your response concise,
    ↪ focusing solely on providing the class label.

Tweet:

{{ sample_text }}
```

2. **Itemized:**

```
Instructions:

- Given a tweet, your task is to classify its sentiment
    ↪ into one of three classes.
- Output only one of the following class labels that best
    ↪  fits the tweet's sentiment: "Positive", "Negative
    ↪ ", "Neutral".
- Do not include any additional text or commentary in
    ↪ your response.
- For tweets containing offensive language, personal
    ↪ attacks, or inappropriate content, classify them
    ↪ based on the sentiment of the context.
- If a tweet does not contain clear sentiment or is
    ↪ primarily informational (e.g., about flight delays)
    ↪ , classify it as "Neutral".
- Keep your response concise, focusing solely on
    ↪ providing the class label.

Tweet:

{{ sample_text }}
```

Experimental results

Model	Samples	Baseline	Itemized
LLAMA_7B	5000	61.7% ± 1.4%	71.2% ± 1.3%
LLAMA_13B	5000	73.1% ± 1.2%	68.1% ± 1.3%
LLAMA_70B	5000	75.0% ± 1.2%	74.9% ± 1.2%
GPT_35	5000	80.0% ± 1.1%	77.0% ± 1.2%
GEMINI_PRO	500	73.9% ± 3.9%	73.9% ± 3.9%

Table 5.4: Accuracies of LLM models for different prompt variations in the Instruction Itemization experiment on the TWITTER dataset.

5.2.2 ARC

Prompt templates

1. **Baseline:**

```
Select the correct answer by choosing the appropriate
    ↪ label. Each question has a single correct answer.
    ↪ The possible labels are for example A, B, C, D, E.
    ↪ Your response should contain only the letter
    ↪ corresponding to the correct answer.

Question: {{ input.question }}

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

2. **Itemized:**

```
Instructions:

- Select the correct answer by choosing the appropriate
    ↪ label.
- Each question has a single correct answer.
- The possible labels are for example A, B, C, D, E.
- Your response should contain only the letter
    ↪ corresponding to the correct answer.
```

```
Question: {{ input.question }}

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

Experimental results

Model	Samples	Baseline	Itemized
LLAMA_7B	2500	55.5% ± 1.9%	51.8% ± 2.0%
LLAMA_13B	2500	62.8% ± 1.9%	61.6% ± 1.9%
LLAMA_70B	2500	76.2% ± 1.7%	73.2% ± 1.7%
GPT_35	2500	82.8% ± 1.5%	82.2% ± 1.5%

Table 5.5: Accuracies of LLM models for different prompt variations in the Instruction Itemization experiment on the ARC dataset.

5.2.3 COSMOS

Prompt templates

1. **Baseline:**

```
Select the correct answer by choosing the appropriate
    ↪ label. The selection should be based on the
    ↪ information provided in the context. Each question
    ↪ has a single correct answer. The possible labels
    ↪ are for example A, B, C, D, E. Your response should
    ↪  contain only the letter corresponding to the
    ↪ correct answer.

Context: {{ input.context }}

Question: {{ input.question }}

Options:

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
```

```
{% endfor %}
```

2. **Itemized:**

```
Instructions:

- Select the correct answer by choosing the appropriate
    ↪ label.
- The selection should be based on the information
    ↪ provided in the context.
- Each question has a single correct answer.
- The possible labels are for example A, B, C, D, E.
- Your response should contain only the letter
    ↪ corresponding to the correct answer.

Context: {{ input.context }}

Question: {{ input.question }}

Options:

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

Experimental results

Model	Samples	Baseline	Itemized
LLAMA_7B	5000	65.3% ± 1.3%	80.1% ± 1.1%
LLAMA_13B	5000	68.1% ± 1.3%	67.8% ± 1.3%
LLAMA_70B	5000	78.7% ± 1.1%	81.2% ± 1.1%
GPT_35	5000	76.5% ± 1.2%	77.6% ± 1.2%

Table 5.6: Accuracies of LLM models for different prompt variations in the Instruction Itemization experiment on the COSMOS dataset.

Results analysis

For the TWITTER dataset, LLAMA_7B showed a significant improvement in accuracy with itemized instructions (9.5%), while LLAMA_13B and GPT_35

showed a significant decrease (5%, 3%). The performance of LLAMA_70B and GEMINI_PRO remained roughly unchanged. In the ARC dataset, LLAMA_70B showed a significant decrease in accuracy with itemized instructions (2.9%), while the other models showed only minor changes that are not statistically significant. For the COSMOS dataset, LLAMA_7B and LLAMA_70B showed significant improvements in accuracy with itemized instructions (14.8%, 2.5%), while LLAMA_13B and GPT_35 showed no significant changes. This results indicates that itemization may not be uniformly beneficial for different tasks and models, sometimes enhancing performance and sometimes degrading it. All in all, the impact of instruction itemization appears to vary significantly across different models and datasets. This indicates that there is likely no clear, consistent pattern in how instruction itemization affects LLM performance. The observed, seemingly random differences in results and the overall inconsistency of the results could be caused by several other factors that will be discussed later in this chapter.

5.3 Positive vs. negative formulation

This section describes the experiments conducted to evaluate the impact of positive vs. negative formulation on model performance. We created two prompt variations for each dataset, one with a positive formulation and the other with a negative formulation.

5.3.1 TWITTER

Prompt templates

1. **Baseline:**

```
Given a tweet, your task is to classify its sentiment
    ↪ into one of three classes: Positive, Negative, or
    ↪ Neutral. Output only one of the following class
    ↪ labels that best fits the tweet's sentiment: "
    ↪ Positive", "Negative", "Neutral". For tweets
    ↪ containing offensive language, personal attacks, or
    ↪  inappropriate content, classify them based on the
    ↪ sentiment of the context instead of refusing to
    ↪ classify them. If a tweet does not contain clear
```

```
    ↪ sentiment or is primarily informational, classify
    ↪ it as "Neutral". Keep your response concise,
    ↪ focusing solely on providing the class label.

Tweet:

{{ sample_text }}
```

2. **Negative:**

```
Given a tweet, your task is to classify its sentiment
    ↪ into one of three classes: Positive, Negative, or
    ↪ Neutral. Do not output anything else other than the
    ↪  class labels that best fits the tweet's sentiment:
    ↪  "Positive", "Negative", "Neutral". Don't refuse to
    ↪  classify tweets that contain offensive language,
    ↪ personal attacks, or inappropriate content and
    ↪ classify them on the sentiment of the context. If a
    ↪  tweet does not contain clear sentiment or is
    ↪ primarily informational, don't classify it as "
    ↪ Positive" nor "Negative". Avoid outputting
    ↪ information other than the selected class label,
    ↪ keeping the response concise.

Tweet:

{{ sample_text }}
```

Experimental results

Model	Samples	Baseline	Negative
LLAMA_7B	3000	80.4% ± 1.4%	79.0% ± 1.5%
LLAMA_13B	3000	80.4% ± 1.4%	79.0% ± 1.5%
LLAMA_70B	3000	83.9% ± 1.3%	84.3% ± 1.3%
GPT_35	3000	77.0% ± 1.5%	77.8% ± 1.5%
GEMINI_PRO	500	74.9% ± 3.8%	75.8% ± 3.8%

Table 5.7: Accuracies of LLM models for different prompt variations in the Positive vs. Negative Formulation experiment on the TWITTER dataset.

5.3.2 ARC

Prompt templates

1. **Baseline:**

```
Select the correct answer to the question by choosing the
    ↪  appropriate label. Each question has a single
    ↪ correct answer. Your response should contain only
    ↪ the letter corresponding to the correct answer.

Question: {{ input.question }}

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

2. **Negative:**

```
Select the correct answer to the question by choosing the
    ↪  appropriate label. Remember, there should never be
    ↪  more than one correct answer. Your response should
    ↪  not contain anything other than the single letter
    ↪ that corresponds to the correct answer. Do not
    ↪ include any additional characters or explanations.

Question: {{ input.question }}

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

Experimental results

Model	Samples	Baseline	Negative
LLAMA_7B	2500	77.4% ± 1.6%	79.0% ± 1.6%
LLAMA_13B	2500	77.4% ± 1.6%	79.0% ± 1.6%
LLAMA_70B	2500	92.4% ± 1.0%	92.6% ± 1.0%
GPT_35	2500	82.8% ± 1.5%	81.2% ± 1.5%

Table 5.8: Accuracies of LLM models for different prompt variations in the Positive vs. Negative Formulation experiment on the ARC dataset.

5.3.3 COSMOS

Prompt templates

1. **Baseline:**

```
Select the correct answer to the question by choosing the
    ↪  appropriate label. The response should be based
    ↪ solely on the provided context. Each question has a
    ↪  single correct answer. Your response should
    ↪ contain only the letter corresponding to the
    ↪ correct answer.

Context: {{ input.context }}

Question: {{ input.question }}

Options:

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

2. **Negative:**

```
Select the correct answer to the question by choosing the
    ↪  appropriate label. The response should not be
    ↪ based on anything but the provided context.
    ↪ Remember, there should never be more than one
    ↪ correct answer. Your response should not contain
```

```
    ↪ anything other than the single letter that
    ↪ corresponds to the correct answer. Do not include
    ↪ any additional characters or explanations.

Context: {{ input.context }}

Question: {{ input.question }}

Options:

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

Experimental results

Model	Samples	Baseline	Negative
LLAMA_7B	3000	81.2% ± 1.4%	84.0% ± 1.3%
LLAMA_13B	3000	81.2% ± 1.4%	84.0% ± 1.3%
LLAMA_70B	3000	90.9% ± 1.0%	91.1% ± 1.0%
GPT_35	3000	76.9% ± 1.5%	76.6% ± 1.5%

Table 5.9: Accuracies of LLM models for different prompt variations in the Positive vs. Negative Formulation experiment on the COSMOS dataset.

Results analysis

In general, the differences in performance between the baseline and the negative prompt formulations are negligible and not statistically significant. This is true across all models and datasets. For example, the TWITTER dataset, LLAMA_7B and LLAMA_13B showed slight decreases in accuracy with the negative formulation, while LLAMA_70B, GPT_35, and GEMINI_PRO showed slight increases. On the other hand in the ARC dataset, LLAMA_7B and LLAMA_13B showed minor improvements with the negative formulation, while LLAMA_70B showed a minimal increase and GPT_35 showed a slight decrease. The confidence intervals for all these changes overlap, indicating that these differences are not statistically significant and could be attributed to a random variability. Overall, the results suggest that the impact of positive versus negative formulation is not consistent across different models and datasets.

5.4 Language correctness

This section describes the experiments conducted to evaluate the impact of language correctness on model performance. Two categories of errors were introduced to the prompts:

- **Grammatical errors:** These errors include incorrect verb forms, noun-number agreement mistakes, and other common grammatical issues that can occur in written English.
- **Idiomatic errors:** These are errors typically made by non-native speakers and include incorrect word order, misuse of prepositions, and other errors that affect the natural flow of English.

We created three prompt variations for each dataset, one without errors, one with grammatical errors, and the other with idiomatic errors.

5.4.1 TWITTER

Prompt templates

1. **Baseline:**

```
Given a tweet, your task is to classify its sentiment
    ↪ into one of three classes: Positive, Negative, or
    ↪ Neutral. For tweets containing offensive language,
    ↪ personal attacks, or inappropriate content,
    ↪ classify them based on the sentiment of the context
    ↪  instead of refusing to classify them. If a tweet
    ↪ does not contain clear sentiment or is primarily
    ↪ informational, classify it as "Neutral". Keep your
    ↪ response concise, providing only the selected class
    ↪  label.

Tweet:

{{ sample_text }}
```

2. **With grammatical errors:**

```
Given a tweet, your tasks are classyfy its sentiment into
    ↪  one of three classes: Positive, Negative, or
    ↪ Neutral. For tweet containing offensives languages,
    ↪  personal attacking, or inappropriates content,
    ↪ classifies them based on the sentiments of the
    ↪ contexts instead of refused to classify. If a tweet
    ↪  does not contained clear sentiments or is
    ↪ primarily informations, classifies it as "Neutral".
    ↪  Keep you response concise, providing only the
    ↪ selected class labels.

Tweet:

{{ sample_text }}
```

- Incorrect verb forms: “classyfy”, “classifies”, “contained”, “classifies”
- Noun-number agreement: “tasks”, “offensives languages”, “sentiments”, “informations”
- Miscellaneous: “Keep you response concise” should be “Keep your response concise”

3. **With idiomatic errors:**

```
Give tweet, task you is classify sentiment into three
    ↪ classes: Positive, Negative, or Neutral. Tweets
    ↪ with offensive language, attack personal, or
    ↪ content not appropriate, classify on sentiment from
    ↪  context instead deny to classify. If tweet not
    ↪ having clear sentiment or mainly for information,
    ↪ put as "Neutral". Keep response short, give only
    ↪ label class chosen.

Tweet:

{{ sample_text }}
```

- Incorrect word order: “Give tweet, task you is classify sentiment”, “give only label class chosen”
- Misuse of prepositions: “classify on sentiment”, “instead deny to classify”
- Inconsistent verb tenses and forms: “not having clear sentiment”, “put as Neutral”

Experimental results

Model	Samples	Baseline	With grammatical errors	With idiomatic errors
LLAMA_7B	5000	81.3% ± 1.1%	79.8% ± 1.1%	80.8% ± 1.1%
LLAMA_13B	5000	81.3% ± 1.1%	79.8% ± 1.1%	80.8% ± 1.1%
LLAMA_70B	5000	83.2% ± 1.0%	82.5% ± 1.1%	82.4% ± 1.1%
GPT_35	5000	75.1% ± 1.2%	69.6% ± 1.3%	66.1% ± 1.3%
GEMINI_PRO	1000	73.5% ± 2.8%	72.9% ± 2.8%	70.6% ± 2.8%

Table 5.10: Accuracies of LLM models for different prompt variations in the Language Correctness experiment on the TWITTER dataset.

5.4.2 ARC

Prompt templates

1. **Baseline:**

```
Select the correct answer by choosing the appropriate
    ↪ label. Each question has a single correct answer.
    ↪ The possible labels are, for example, A, B, C, D, E.
    ↪  Your response should contain only the letter
    ↪ corresponding to the correct answer.

Question: {{ input.question }}

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

2. **With grammatical errors:**

```
select the corrected answer by choosing the appropiate
    ↪ label. every questions has a single correct answer.
    ↪  The possibly labels are for example A, B, C, D, E.
    ↪  your response should contains only the letter
    ↪ correspond to the correct answer.

Question: {{ input.question }}

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

- Capitalization errors: “select” should be capitalized.
- Spelling errors: “appropiate” instead of “appropriate.”
- Singular/plural mismatches: “every questions has” should be “every question has.”
- Adjective/noun mismatches: “possibly labels” should be “possible labels.”
- Verb form errors: “contains” should be “contain,” “correspond” should be “corresponding.”

3. **With idiomatic errors:**

```
Choosing the correct answer by selecting label
    ↪ appropriate. Every question single correct answer
    ↪ has. Labels possible are, for example, A, B, C, D,
    ↪ E. Only letter corresponding to answer correct your
    ↪  response should contain.

Question: {{ input.question }}

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

- Inverted syntactic structure: “selecting label appropriate” should be “selecting the appropriate label.”
- Incorrect verb placement and form: “Every question single correct answer has” should be “Each question has a single correct answer.”
- Misuse of prepositions and word order: “Only letter corresponding to answer correct your response should contain” should be “Your response should only contain the letter that corresponds to the correct answer.”

Experimental results

Model	Samples	Baseline	With grammatical errors	With idiomatic errors
LLAMA_7B	2500	77.6% ± 1.6%	76.3% ± 1.7%	76.1% ± 1.7%
LLAMA_13B	2500	77.6% ± 1.6%	76.3% ± 1.7%	76.1% ± 1.7%
LLAMA_70B	2500	92.3% ± 1.0%	92.4% ± 1.0%	91.6% ± 1.1%
GPT_35	2500	82.8% ± 1.5%	81.9% ± 1.5%	82.3% ± 1.5%

Table 5.11: Accuracies of LLM models for different prompt variations in the Language Correctness experiment on the ARC dataset.

5.4.3 COSMOS

Prompt templates

1. **Baseline:**

```
Select the correct answer to the question based on the
    ↪ provided context by choosing the appropriate label.
    ↪  Each question has a single correct answer. The
    ↪ possible labels are A, B, C, D. Your response
    ↪ should contain only the letter corresponding to the
    ↪  correct answer.

Context: {{ input.context }}

Question: {{ input.question }}

Options:

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

2. **With grammatical errors:**

```
select the corrected answer to question based on provided
    ↪  context by choosing the appropiate label. every
    ↪ questions has a single correct answer. The possibly
    ↪  labels are A, B, C, D. your response should
    ↪ contains only the letter correspond to the correct
    ↪ answer.

Context: {{ input.context }}

Question: {{ input.question }}

Options:

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

3. **With idiomatic errors:**

```
Choosing correct answer to question based on provided
    ↪ context by selecting label appropriate. Every
    ↪ question single correct answer it has. Labels
    ↪ possible are A, B, C, D. Only letter corresponding
    ↪ to answer correct your response should contain.

Context: {{ input.context }}

Question: {{ input.question }}

Options:

{% for label, text in input.choices.items() %}
{{ label }}: {{ text }}
{% endfor %}
```

The errors in this section are similar to those in the ARC dataset and their descriptions are omitted.

Experimental results

Model	Samples	Baseline	With grammatical errors	With idiomatic errors
LLAMA_7B	5000	83.4% ± 1.0%	83.9% ± 1.0%	81.7% ± 1.1%
LLAMA_13B	5000	83.4% ± 1.0%	83.9% ± 1.0%	81.6% ± 1.1%
LLAMA_70B	5000	91.0% ± 0.8%	90.4% ± 0.8%	90.0% ± 0.8%
GPT_35	5000	77.4% ± 1.2%	77.0% ± 1.2%	76.5% ± 1.2%

Table 5.12: Accuracies of LLM models for different prompt variations in the Language Correctness experiment on the COSMOS dataset.

Results analysis

Across all models and datasets, the introduction of grammatical and idiomatic errors generally results in slight decreases in performance, but these changes are mostly not statistically significant.

For the TWITTER evaluation task, GPT_35 shows significant decreases in accuracy with both grammatical and idiomatic errors (5.5%, 9%), indicating

a potential sensitivity to language correctness. However, for other tasks, GPT_35 shows only minor changes that are not statistically significant. Similarly, GEMINI_PRO shows a slight decrease in performance with idiomatic errors (2.9%), but this decrease is not statistically significant due to the wide confidence interval caused by the smaller sample size for this model. Moreover, the LLAMA models also show only minor changes in performance.

Overall, the results suggest that the impact of language correctness is generally minor and inconsistent across different models and datasets.

5.5 Results discussion

The results across all models and datasets show that none of the four prompt engineering recommendations significantly and consistently impacts the quality of the results.

Specifically:

- **Instruction context separation**: No statistically significant differences in performance were observed between the baseline and the variations using hash separators or triple quotes.
- **Instruction itemization**: Mixed results were observed, with some models showing improvements and others showing decreases in performance. Overall, there was no clear, consistent pattern indicating a significant impact.
- **Positive vs. negative formulation**: Differences in performance between positive and negative formulations were negligible and not statistically significant.
- **Language correctness**: Introduction of grammatical and idiomatic errors generally resulted in slight decreases in performance, with a significant decrease for GPT_35 on the TWITTER dataset. However, other changes were mostly not statistically significant.

Despite the seemingly random behavior of the models, we conducted multiple *sanity check* experiments to ensure there are no bugs in the evaluation framework that could affect the results. The first sanity check experiment

was performed on the COSMOS dataset, where we constructed two prompt variations: a baseline variation identical to the baseline in Experiment 5.4.3, and a second variation where we intentionally left out the context part of the prompt, so that the model lacks the context necessary to answer the question. As expected, the performance of all the models dropped significantly to around 40%, which is less than half of the correct predictions for the baseline for all models. As a next sanity check, we took the baseline prompt from Experiment 5.1.1 and constructed a variation where the entire text from the instructional part of the prompt was reversed (characters written from end to beginning). As expected, the accuracy of all models on this task dropped to 0%. This means that we can be confident there are no bugs in the evaluation process and the seemingly random results might be caused by other factors, such as the ones described below.

The different accuracies observed for the same model and similar baseline prompts across datasets can be attributed to the experiments being run at different times with different random seeds, resulting in different subsets of samples from the original dataset. The impact of prompt engineering recommendations can differ greatly among various datasets, highlighting the importance of optimizing prompts for each dateset separately. This finding is consistent with the previous research that has shown the importance of dataset-specific prompt engineering [19]. Similarly, the differences in the effectiveness of prompt engineering recommendations across models could be attributed to different sensitivity of models to different types of prompts, indicating that prompt engineering should be tailored to the specific model being used.

As a practical consequence, our findings suggest that users of LLMs do not need to spend unnecessary time trying to strictly adhere to these guidelines when designing prompts. This can save time and allow for faster experimentation with different prompts and models.

Several factors could explain why no significant signal was found in the results:

- **Task complexity**: For more complicated tasks, such as multiple-choice question answering that involves context, questions, and answer options, the impact of minor, algorithmic-like changes to the prompt, might be less pronounced. The relative ratio of the number of tokens in the instruction part to the number of tokens in the context part is lower, possibly making these modifications less important for the model.
- **Differences in the nature of evaluation tasks**: The different nature of individual evaluation tasks and the varying abilities of models to

solve them effectively could also contribute to the inconsistency in the results. For instance, simpler tasks like sentiment classification might be more sensitive to prompt modifications than more complex tasks like multiple-choice question answering.

- **False generalization assumptions**: Many of the prompt engineering recommendations have been originally devised for foundational completion models like GPT-2 or GPT-3. However, they may not be as relevant for models like GPT-3.5, which have undergone additional training steps such as supervised fine-tuning and reinforcement learning from human feedback described in detail in Chapter 2. These additional training steps may possibly make the models more robust to minor prompt modifications, reducing the impact of these recommendations. Moreover, as already mentioned, the majority of the prompt engineering literature available online is focused on models from the GPT family, likely due to their public availability via ChatGPT, and the general popularity of the models. Very few online articles mention prompt engineering recommendations for other models, such as those from the LLAMA or Gemini series. It is thus questionable how well these recommendations generalize to other models.

Another speculation is that the results might be more pronounced for other text-based tasks such as code generation or summarization, where the quality of the result depends more on the actual instructions given to the model. However, these tasks are harder to robustly evaluate due to the lack of inherently computable metrics. In the evaluation tasks we focused on, especially multiple-choice question answering, it can often be obvious for the model what to do even without specifying detailed instructions, as there are not many possible ways to interpret the prompt that contains a context, question and a set of answer options. This lowers the value of the instructional part of the prompt which was the subject of our modifications, although some preliminary experimentation omitted from this chapter showed that the model still benefits from the instructions to some extent.

5.6 Other observations

Throughout the experiments, we encountered several additional observations that are worth noting. These observations provide insights into the behavior of the LLMs, the challenges encountered, and the broader implications for working with these models.

Firstly, it was challenging to make the models output the answers in the specified format (e.g., a single word corresponding to the class label). This difficulty likely arises from the conversational fine-tuning they underwent, which generally causes the models to output full sentences. Consequently, we had to develop more robust response parsing algorithms to extract the actual answer for comparison with the expected sample answer. Notably, we observed that GPT-3.5 is much more likely to follow the instructions regarding the expected answer format than other models, and the adherence of LLAMA models to these instructions increases with the model size.

Another limitation stems from the careful fine-tuning steps engineered by researchers to minimize the risk of producing harmful or inappropriate content. For example, in the language correctness experiment on the Cosmos dataset, the LLAMA 7B model generated the following response to a sample containing derogatory language:

> *"I cannot provide a response to that question as it contains harmful and derogatory language. The use of such language is not acceptable and promotes a negative and disrespectful attitude towards individuals. It is important to treat others with respect and dignity, and to communicate in a way that is respectful and appropriate. Instead, I suggest rephrasing the question in a respectful and appropriate manner."*

In general, we observed that the tendency of the model to refuse to solve a task because of inputs that could be deemed inappropriate decreased with the size of the model. This suggests that larger models are more robust in correctly perceiving the context in which the inappropriate language appears.

We also noticed a broader issue with the replication of results in the LLM research domain. Academic literature related to LLM benchmarking, particularly in the area of prompt engineering, sometimes suffers from low reproducibility. One contributing factor is the poor quality of accompanying codebases, which frequently suffer from disorganized code, a lack of experiment tracking tools, and poor software design skills. Additionally, the reported metrics in some literature are often not detailed enough and are aggregated across multiple experiments or datasets without clear descriptions of the computation methods. As an example, the work by Mishra et al. [19] illustrates most of the above problems.

Another finding is that benchmarking with LLMs is inherently challenging, not because of the complexity of the experiments themselves, but due to the

nature of communication with the models and the reliability of the third-party services. This underscores the importance of implementing robust communication mechanisms with retrying and caching. Without such logic in place, conducting experiments on this scale would be infeasible. Furthermore, the cost aspect stemming from the decision to interact with models via third-party APIs also cannot be ignored. Redoing the same LLM calls multiple times due to the failure of the benchmarking script caused by temporary network or rate limit errors would quickly add up to a significant cost. The total expenditure for all experiments conducted while working on this thesis was around $70, of which approximately $35 was spent on the final experiments described in this chapter. The latency and cost aspects were the most significant contributors to the inability to perform experiments on a larger scale, which might have helped uncover patterns that were not apparent in the experiments conducted.

In summary, these observations highlight the practical challenges and considerations when working with large language models.

Chapter 6

Conclusion

In this work, we developed a comprehensive evaluation framework to assess the effectiveness of different prompt engineering recommendations for large language models. Through a detailed survey of existing literature, we identified four prevalent prompt engineering recommendations: instruction context separation, instruction itemization, positive vs. negative formulation, and language correctness. We used these to conduct a series of 12 experiments, testing the performance of five different LLMs across three evaluation tasks within the sentiment analysis and multiple-choice question answering domains.

The results of our experiments consistently showed that none of the four prompt engineering recommendations significantly and consistently impacted the quality of the model outputs. These findings suggest that users of LLMs fine-tuned for chat completion may not need to adhere strictly to these prompt engineering guidelines, potentially saving time and effort in designing prompts. Our results also underscore the importance of considering specific tasks and models when applying prompt engineering techniques, as the impact of these techniques can vary. While the evaluated recommendations did not demonstrate significant benefits in our experiments, this does not rule out their potential usefulness in other contexts. Further research is necessary to explore these hypotheses and better understand the conditions under which prompt engineering can be most effective.

We hope that the empirically-driven techniques and findings presented in this work will pave the way towards more effective and practical prompting techniques in the future.

6.1 Future steps

At last, let us identify some future research directions.

Firstly, we could extend the evaluation scope to different prompt engineering recommendations that were not studied in this work, which focused only on four of the most commonly appearing recommendations. For instance, it would be interesting to investigate how the split between the system prompt and the user prompt affects model performance and adherence to instructions. Another interesting way of future research would be to study the effectiveness of framing prompts as questions versus statements. Some hypotheses suggest that models may respond differently to a question format (e.g., "What are the causes of . . . ?") compared to a declarative statement format (e.g., "Describe the causes of . . . "). Furthermore, we could explore combinations of different recommendations to discover any synergistic effects that might improve model performance.

Another compelling next step would be to conduct the experiments with larger volumes of data, potentially using entire datasets, to uncover patterns that were not apparent with only subsets of them. Next, we could extend the set of evaluated LLMs to include not only chat models but also foundational completion models like GPT-2 or GPT-3. This would help test the hypothesis that additional training steps, such as supervised fine-tuning, make the chat models more robust to minor prompt modifications. Lastly, we could run the experiments on more complex evaluation tasks, such as those from code generation or math reasoning domains, to assess the effectiveness of prompt engineering recommendations in these areas.

In spite of the remarkable progress in generative AI in recent years, there is still much work to be done for those of us who aspire to build *truly* intelligent systems that could be considered artificial general intelligence.

Appendix A

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[2] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 1st. USA: Prentice Hall PTR, 2000. ISBN: 0130950696.

[3] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[4] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG].

[5] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[6] Long Ouyang et al. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL].

[7] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[8] OpenAI. "Introducing ChatGPT". In: (2022). URL: https://openai.com/blog/chatgpt.

[9] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[10] Gemini Team et al. "Gemini: A Family of Highly Capable Multimodal Models". In: (2023). arXiv: 2312.11805 [cs.CL].

[11] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL].

[12] Xavier Amatriain. *Prompt Design and Engineering: Introduction and Advanced Methods*. 2024. arXiv: 2401.14423 [cs.SE].

[13] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL].

[14] Jessica Shieh. "Best practices for prompt engineering with OpenAI API". In: (2023). URL: https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-openai-api.

[15] Dair AI. "Prompt engineering guide". In: (2022). URL: https://github.com/dair-ai/Prompt-Engineering-Guide.

[16] Akash Takyar. "Bridging the AI-human communication gap: A guide to prompt engineering". In: (2023). URL: https://www.leewayhertz.com/prompt-engineering/.

[17] Xin Cheng. "Large language models and Prompt Engineering". In: (2023). URL: https://billtcheng2013.medium.com/large-language-models-and-prompt-engineering-1ffd381c10c5.

[18] Emiliano Viotti. "Prompt Engineering 101 - I: Unveiling Principles and Techniques of Effective Prompt Crafting". In: (2023). URL: https://hackernoon.com/prompt-engineering-101-i-unveiling-principles-and-techniques-of-effective-prompt-crafting.

[19] Swaroop Mishra et al. *Reframing Instructional Prompts to GPTk's Language*. 2022. arXiv: 2109.07830 [cs.CL].

[20] Mihail Eric. "A Complete Introduction to Prompt Engineering For Large Language Models". In: (2022). URL: https://www.mihaileric.com/posts/a-complete-introduction-to-prompt-engineering/.

[21] Ansdrew Cantino. "Prompt Engineering Tips and Tricks with GPT-3". In: (2021). URL: https://blog.andrewcantino.com/blog/2021/04/21/prompt-engineering-tips-and-tricks/.

[22] Lianmin Zheng et al. *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. 2023. arXiv: 2306.05685 [cs.CL].

[23] Peter Clark et al. "Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge". In: *ArXiv* abs/1803.05457 (2018). URL: https://api.semanticscholar.org/CorpusID:3922816.

[24] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL].

[25] Lifu Huang et al. *Cosmos QA: Machine Reading Comprehension with Contextual Commonsense Reasoning*. 2019. arXiv: 1909.00277 [cs.CL].

[26] Srijan Bansal et al. *Few-shot Unified Question Answering: Tuning Models or Prompts?* 2023. arXiv: 2305.14569 [cs.CL].

Appendix B

Attachments

Name	Description
experiments/	Source definitions of the conducted experiments.
src/	Source code of the evaluation framework.
tests/	Unit tests for the evaluation framework.
.example.env	Example environment file.
pyproject.toml	Poetry project configuration file.
README.md	Project README file.

Table B.1: Thesis source code repository structure.

This thesis has one attachment - a compressed archive that contains the source code of the evaluation framework used to conduct the experiments. Table B.1 lists the directories and files in the root directory of the archive along with their short description.