Large language models as an interface to interact with API tools in natural language
Abstract

In this research project, we aim to explore the use of Large Language Models (LLMs) as an interface to interact with API tools in natural language. Bubeck et al. [1] shed some light on how LLMs could be used to interact with API tools. Since then, new versions of LLMs have been launched and the question of how reliable a LLM can be in this task remains unanswered. The main goal of our thesis is to investigate the designs of the available system prompts for LLMs, identify the best-performing prompts, and evaluate the reliability of different LLMs when using the best-identified prompts. We will employ a multiple-stage controlled experiment: A literature review where we reveal the available system prompts used in the scientific community and open-source projects; then, using $F_1$-score as a metric we will analyse the precision and recall of the system prompts aiming to select the best-performing system prompts in interacting with API tools; and in a latter stage, we compare a selection of LLMs with the best-performing prompts identified earlier.

From these experiences, we realize that AI-generated system prompts perform better than the current prompts used in open-source and literature with GPT-4, zero-shot prompts have better performance in this specific task with GPT-4 and that a good system prompt in one model does not generalize well into other models.

Keywords: Large language model (LLM), Natural Language Processing (NLP), GPT-4, Llama-2, Palm, Application Programming Interface (API).
Acknowledgements

We extend our deepest gratitude to our supervisor, Diego Pérez Palacin, for his unwavering support throughout the completion of this thesis. We also extend heartfelt thanks to our families for their endless love and encouragement, without which we could not have achieved this milestone.
# Contents

1 **Introduction** ........................................................... 1  
  1.1 Background ....................................................... 1  
  1.2 Related work ..................................................... 2  
  1.3 Problem formulation ............................................. 3  
  1.4 Motivation ......................................................... 3  
  1.5 Scope/Limitation .................................................. 4  
  1.6 Target group ..................................................... 5  

2 **Method** ................................................................. 6  
  2.1 Research Project ................................................ 6  
  2.2 Research methods ................................................ 6  
    2.2.1 Literature Review ......................................... 6  
    2.2.2 Evaluation and tests of different prompt designs ....... 7  
    2.2.3 Comparison of different Large Language Models ...... 8  
  2.3 Reliability and Validity .......................................... 8  
    2.3.1 Testing the positive class ................................. 9  
    2.3.2 Testing the negative class ................................. 9  
    2.3.3 Final thoughts about reliability and validity .......... 9  
  2.4 Ethical considerations ........................................... 10  

3 **Theoretical background** ............................................ 11  
  3.1 Large Language Model (LLMs) .................................. 11  
    3.1.1 Importance and Types of LLMs: ............................. 11  
    3.1.2 Detailed Overview of GPT-3 and GPT-4: .................. 12  
    3.1.3 LLMs underlying Theories: Transformer Architecture, Attention Mechanism, and Unsupervised Learning ............. 12  
    3.1.4 Zero-shot, Few-shot Prompts, and Fine-Tuning: .......... 12  
    3.1.5 Usage and Applications of LLMs in Various Fields: ...... 13  
  3.2 Prompt Design ..................................................... 13  
    3.2.1 Prompt Design Methodologies ............................... 13  
    3.2.2 Practical Examples ......................................... 13  
    3.2.3 Strengths and Limitations of Various Prompt Designs .... 14  
  3.3 Application Programming Interface (API) Tools ............... 15  
    3.3.1 Importance and Usage in Software Development ......... 15  
    3.3.2 Overview of APIs ........................................... 15  
    3.3.3 Interaction of APIs with LLMs ............................. 15  
  3.4 Tool Learning ..................................................... 16  
    3.4.1 Different Tool Learning Methodologies .................... 16  
    3.4.2 Strengths and Limitations of Various Tool Learning ...... 16  
  3.5 Performance Evaluation Criteria ................................ 17  
    3.5.1 Comprehending Binary Classification and the Confusion Matrix ........ 17  
    3.5.2 Precision, Recall and their Importance in Evaluation ... 17  
    3.5.3 F-Score: balancing Precision and Recall ................ 18  
    3.5.4 Reliability Measures in Research Experiments .......... 19  
    3.5.5 Linking Evaluation Metrics to Large Language Models ...... 19  
  3.6 Research Studies and Research Gap Identification ............ 19
4 Research project – Implementation

4.1 Literature Review .................................................. 20
4.2 Expansion of prompt designs using GPT-4 .......................... 20
4.3 API commands standardization ...................................... 21
4.4 Testing scenarios for user prompt ................................. 22
4.5 Experimentation Environment ..................................... 22

5 Results

5.1 Literature Review and prompt discovery .......................... 23
5.2 Testing prompt designs .............................................. 24
5.3 Testing Large Language Models models .......................... 29

6 Analysis ..................................................................... 33

7 Discussion ................................................................... 39

8 Conclusions and Future Work ........................................ 41

References ..................................................................... 42

A Plain text API list with input arguments A
B AutoGPT system prompt A
C Prompt 1 generated by GPT-4 D
D Prompt 2 generated by GPT-4 E
E Prompt 3 generated by GPT-4 F
F Prompt 4 generated by GPT-4 G
G Prompt 5 generated by GPT-4 H
H Prompt 6 generated by GPT-4 I
I Prompt 7 generated by GPT-4 J
J Prompt 8 generated by GPT-4 K
K Prompt 9 generated by GPT-4 M
L Prompt 10 generated by GPT-4 N
M Prompt 11 generated by GPT-4 N
N Prompt 12 generated by GPT-4 P
O Prompt 13 generated by GPT-4 Q
P Prompt 14 generated by GPT-4 R
Q Prompt 15 generated by GPT-4 T
<table>
<thead>
<tr>
<th>Prompt</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Prompt 16 generated by GPT-4</td>
<td>U</td>
</tr>
<tr>
<td>S</td>
<td>Prompt 17 generated by GPT-4</td>
<td>W</td>
</tr>
<tr>
<td>T</td>
<td>Prompt 2 generated by GPT-4 with examples (one-shot prompt)</td>
<td>Y</td>
</tr>
<tr>
<td>U</td>
<td>Prompt 5 generated by GPT-4 with examples (one-shot prompt)</td>
<td>Y</td>
</tr>
</tbody>
</table>
1 Introduction

This study is a 15 HEC undergraduate thesis project in computer science investigating the potential of large language models (LLMs) as an interface for interacting with API tools in natural language. The project aims to see whether LLMs, including GPT4, Llama-2 and PaLM, can effectively be integrated with API tools (e.g., Google calendar, email or message and search engines) and evaluate their performance in satisfying user interactions. The study delves into the capabilities of LLM, prompt selection, and the potential challenges associated with LLMs for interacting with APIs as an interface.

1.1 Background

The rapid advancement of natural language processing (NLP) technologies, particularly LLMs like OpenAI’s GPT-3 [2] and Google’s BERT [3], has revolutionised various domains, such as conversational AI, content generation, and information retrieval. LLMs have shown remarkable capabilities in understanding user inputs and generating appropriate responses in a conversational setting [4].

In the study, we explore various studies that examine LLMs and their ability to interact with APIs and tools; Bubeck et al. [1] examined GPT-4’s capabilities and limitations, while Parisi et al. [5] presented Tool Augmented Language Models (TALM) that merge LLMs with non-differentiable tools. Lazaridou et al. [6] proposed a method for conditioning LLMs on web information to enhance open-domain question-answering performance. Lastly, Schick et al. [7] introduced Toolformer, which uses external tools via simple APIs to improve zero-shot performance, albeit with some limitations. However, designing effective prompts for LLMs to interact with API tools in natural language remains an open challenge [8].

Prompt programming, the process of designing effective prompts for LLMs, has emerged as a crucial aspect of harnessing the full potential of these models. A prompt is a set of instructions or text given to an LLM to produce a desired output [9] [10]. A system prompt is an initial pre-written text or message that is used to start a conversation and establish the context. An example of a system prompt can be seen in Listing 1. A user prompt or user request in text form provides more specific information about their intention and what they want the LLM to generate [11], as seen in Listing 2.

You are an helpful chatbot that answers user queries using the following API commands:

1. **CalculateDistance** (from_location : str, to_location : str, unit : str)
2. **GetWeather** (city : str)
3. **GetNews** (topic : str)

Instructions:
1. Identify the user’s intention and the required API command.
2. Ask questions to gather all necessary input arguments.
3. Format the input arguments according to the given format.
4. Respond with the selected API command and input arguments in one line.

Listing 1: Example of system prompt

Listing 2: Example of user prompt

Techniques like few-shot learning [12] further enhance the model’s ability to generate API requests and process responses. Effectiveness of such integrations can be evaluated by assessing the model’s accuracy in interpreting user queries, the quality of generated API calls, and user satisfaction with the system’s responses. While challenges include handling ambiguous queries, ensuring security and privacy, and managing computational resources, natural language interfaces for APIs can lead to more accessible and user-friendly applications [2]. Selecting effective prompts is crucial to the performance of LLMs. Criteria for evaluating and selecting the best prompts include gathering all the data needed from the user and structuring the output so it can be parsed by the wrapper function/middleware and the ability to elicit meaningful responses [2]. Adherence to these criteria can optimise the model’s performance and produce desired results. In this controlled experiment, the study will investigate available prompts for LLMs and evaluate their effectiveness based on the aforementioned criteria. By determining the best prompt, the research aims to compare the reliability of LLMs when using the selected prompt, shedding light on the potential of LLMs as an interface for interacting with API tools in natural language.

1.2 Related work

Bubeck et al. [1] examined the capabilities of an early version of GPT-4 across various domains. Their findings demonstrate the impressive performance of the model but also acknowledge its limitations. The study highlights the potential of LLMs as an interface for interacting with APIs, providing a foundation for further investigation into this area.

Parisi et al. [5] introduced Tool Augmented Language Models (TALM), which combine LLMs with non-differentiable tools to enhance their capabilities. According to the study, TALM performs knowledge-heavy question answering (QA) and reasoning math tasks well. It outperforms non-augmented LLMs on MathQA [13] with significant accuracy. This suggests TALM enhances reasoning capabilities.

Lazaridou et al. [6] proposed a few-shot prompting approach to condition LLMs on external information from the web, leading to improved performance in open-domain question-answering tasks. Their approach demonstrates the potential of using external information sources to augment the abilities of LLMs and facilitate more effective interactions with APIs and tools. Schick et al. [7] introduced the Toolformer, a language model that learns to use external tools via simple APIs to improve zero-shot performance. While the Toolformer shows promising results on API interaction, it has limitations, such as sensitivity to input wording and an inability to use tools interactively or in a chain. These limitations indicate areas for further research and development in the LLMs and API interaction field.

Regarding evaluation techniques, researchers have primarily focused on the accuracy of LLMs in understanding user intent and their ability to use external tools to expand their intrinsic abilities.
On April 17, 2023, Qin, Yujia, et al. [11] published a study that demonstrated the interaction of using LLMs with various APIs. The study involved experimenting with 17 representative tools, such as search engines, Wikipedia, professional domain tools, and online shopping tools. The results showed that current foundation models have the potential to effectively utilize these tools to enhance accuracy, efficiency, and automation in problem-solving. The study also discussed the concept of tool learning with foundation models, which involves combining specialized tools and foundation models to achieve better results. The researchers proposed a general tool learning framework that includes understanding user instructions, breaking down complex tasks into sub-tasks, dynamic planning, and selecting appropriate tools for each sub-task. The paper concludes by outlining several open problems that require further investigation for tool learning, such as bridging user intents with proper tools, better planning by leveraging the reasoning abilities of foundation models, training strategies for tool learning, and facilitating generalization for tool learning. The study hopes future research will integrate tools with foundation models to achieve even better results.

On the whole, the related work demonstrates the potential of LLMs in interacting with APIs and tools and highlights the need for further research to overcome existing limitations and explore more advanced techniques to evaluate LLM performance in this area. As LLMs evolve and new models are launched, there is a need to measure the performance of newly launched models like GPT-4, Llama-2 and Palm.

1.3 Problem formulation

The rapid development and publication of new Large Language Models (LLMs) have made it challenging to thoroughly document and evaluate their reliability. Consequently, this presents an opportunity to assess the performance of these LLMs when interacting with various API tools.

To effectively interact with APIs, LLMs may need to gather additional information from users by asking follow-up questions and ensuring all necessary data is available before proceeding. Furthermore, it demands a comprehensive understanding of the available API tools, their input arguments, and utilities by the LLM. Through comparing and analyzing different prompt designs, we aim to find an optimal solution to encode and instruct an LLM.

To address these challenges, our thesis will focus on the following research questions:

- What are the various available prompt designs that enable Large Language Models to effectively interact with API tools?
- Which prompt designs perform the best in terms of detecting user intent, selecting the appropriate tool, obtaining any necessary additional data from users, and structuring the final output?
- How reliable are different Large Language Models when utilizing the most effective prompt design identified through our research?

1.4 Motivation

The utilization of natural language in API interaction holds potential in various areas such as accessibility, efficiency, and business process optimization, which could redefine software use for both general users and businesses alike.
By simplifying the technical process of interacting with APIs, technology becomes increasingly accessible to all users regardless of their technical background, reflecting the democratizing potential of user-friendly design. This process is further enriched by natural language processing, which augments the efficiency of API interaction by reducing the need to navigate through multiple commands or menus.

Additionally, the interaction becomes more engaging as it deviates from a stereotypically mechanical process to a more organic form of communication, contributing to a more comfortable and enjoyable user experience. On the accessibility side, this natural language approach can be easily converted into voice, using text-to-speech software and on the other hand using voice-to-text to query the software which opens new horizons for impaired people to be able to interact with software that was not accessible before.

From a business perspective, creating user interfaces that utilise natural language can enhance customer appeal, making the product or service offering more attractive. It can be used to automatize some processes, for example, answering questions about a product or service, it can make the actual interaction with a service or product could be made entirely by chatting or talking with the software in natural language and much more. This can reduce the amount of human resources needed to provide customer support in a company, effectively saving money, time and resources.

LLMs show an impressive ability to understand the intention of the user, generate a custom answer to his queries or solve specific problems [14]. They already have a general knowledge of the world [15], but, on their own, they lack the specific knowledge domain of a company’s knowledge base, they are unaware of the current time and date [16], they do not perform well with maths calculations [17], they are unable to access current events [18]. The ability to use external APIs (public or private) would increase the power of the LLM to retrieve data and its ability to interact with other systems. By testing current approaches to present external API tools to an LLM we could discover the best way to create a prompt to the LLM, and indirectly decrease radically the amount of time and resources needed to develop a useful chatbot. This thesis aims to provide experimental results from different LLM systems that can help chatbot and systems designers to choose a combination of system prompt and LLM to interact with APIs. We will try to understand if LLMs can reliably do the work of selecting a tool, gathering all the data needed, and if they are able to generalise and use any API toolset provided without any fine-tuning.

1.5 **Scope/Limitation**

This paper aims to investigate how LLMs can be used as an interface to interact with API tools in natural language. In this research, we will experiment with different prompt designs: different ways to structure each API command or function, its inputs and instructions to encode the data from the user to be presentable to an API endpoint. The thesis will focus on testing the most recent and popular LLMs, including GPT4, Palm, and Llama-2, and will compare side-by-side different system prompts and designs. Multiple test cases will be created with different user queries to determine if all API commands are used according to each test case. Due to the time limitations for developing the project, we have accepted the following limitations in the results:

- All the user queries will be simple and directed to one API command at a time. We will not test situations where the LLMs need data from the reply of a previous API command to be incorporated as an input argument on the next API command. This way we decrease the complexity of the query as well as minimize possible parallel interpretations.
Each experiment stops when the model returns the API function name and input data structured in the instructed way. We will not test the API call itself or the way the model would reply to the user after getting the data from the API.

The limited time and resources make us select a subset of LLMs to test in our controlled experiments. This may not reflect the full range of real-world scenarios where LLMs are used to interact with API tools.

1.6 Target group

This research paper aims to be a guide to system designers that might need to understand the differences in performance and limitations of each prompt design, specifically when using LLMs as an interface between the user and API tools or commands. Additionally, our research can be used as a base for further study and experimentation by other researchers.
2 Method

In this section, we will describe our research methodology to address research questions and bridge knowledge gaps.

2.1 Research Project

In this research project, we aim to explore using LLMs as an interface to interact with API tools in natural language. The main goal is to investigate the designs of the available prompts for LLMs, identify the best-performing prompts, and evaluate the reliability of different LLMs when using the selected prompt. We will employ a multiple-stage controlled experiment using F-score as a metric for analyzing the accuracy of a prompt.

First, in order to answer our research questions we will analyse the literature and open-source projects in this area to select the prompts to test.

After, in the second stage, we will compare and evaluate the effectiveness of those prompt designs in instructing the LLM on the API commands/tools it has available, stating the number and quality of the input arguments for each tool, and how it should present the data after gathering it from the user.

Then, in the third step, after selecting the best-performing prompts from the previous step, we will use them to compare the performance of the selected LLMs side-by-side in order to find the best-performing LLM from a selection of models.

2.2 Research methods

For the first stage of the project, we will conduct a literature review of the available literature and publicly available open-source projects to gather different system prompt designs for our experiments.

In the second step of the multi-stage controlled experience, we will use the GPT-4 large language model to evaluate the best system prompt designs for the task of interacting with API tools. This will be done by creating different test cases that will ask for the large language model to use each API command it has available. These results will be classified manually into true/false positives and true/false negatives, which will be used to calculate the $F_1$ score (this metric is explained in detail in section 3.5.3) for each prompt design. For this part of the study, the independent variables are the prompt designs.

After the selection of the best prompt designs, we will evaluate the performance of different LLMs. The independent variables in this case will be the LLMs and the system prompts used will remain the same across all the LLMs.

2.2.1 Literature Review

In the literature review part we will use the search engines OneSearch from Linnaeus University Library and Google Scholar. Additionally, we used the github.com search feature to search popular and publicly available open-source projects to gather different prompt designs for our experiments. The prompt designs must be considerably different from each other to be considered in our study.

In the case that we cannot find enough system prompts for this task. We can further expand the prompt designs by generating additional system prompts using GPT-4.

All the prompts will be standardized with a selection of API commands before continuing to the next step.
2.2.2 Evaluation and tests of different prompt designs

As mentioned before, the first stage is to find the best prompt design from our selection. In order to accomplish this, we will generate test cases that will explore the possible variations for a user query. Each API command will have a correspondent list of user prompts that are expected to trigger them. As time and money are limited, we will test each API command with 20 different user prompts.

In some situations, further information is needed from a user in order to use an API command. For example, when an API command requires more information (input arguments) to run, the LLM must ask the user for this missing information. For any additional interactions (if needed) we will supply the LLM with a possible user answer on the fly. All these interactions will be recorded for future evaluation.

We will ensure to test a scenario multiple times because of the intrinsic randomness of a large language model answer. The same combination of system prompt and user queries can generate different outputs. As such, each combination of user prompt and system prompt will be tested at least 3 times to verify the similarity of the results. This will also increase the number of results which will translate into more significant data.

We will also test scenarios where there is no available API command, for that we will use a user prompt that will ask for something that there is no API command available and the LLM is expected to answer “I do not know”. In order to keep the balance between positive and negative prompts, we will generate the same amount of negative user prompts as the positive ones. As we do have 10 API commands with 20 user prompts for each, we will generate a similar amount of user prompts where there is no available API command. These scenarios will be also tested 3 times.

When testing the positive class (where there is an API command available for the user request), we will evaluate the reply using the following criteria:

1. if it successfully detects the intent of the user and selects the right API tool;
2. if it asks for all the data needed to use the API tool selected
3. if it outputs the gathered data in the format instructed in the system prompt.

A full example of a successful experience can be found in Listing 3 (True positive).

System prompt:
You are a helpful chatbot named Jason and you have the following tools available to fulfill the user solicitation:
SEND_EMAIL(email_recipient, subject, message)
CREATECALENDAR_EVENT(date, time, event_name)
GETCALENDAR_EVENTS(date)
CREATEREMINDER(date, time, text)
GETCURRENT_DATE_AND_TIME()
GOOGLE_SEARCH(search_query)
Whenever you get all the data needed from the user you must use the name of the tools with the respective input arguments as a reply to the user. You must ask for all the data needed to use a tool. If there is no available tool to answer the user you should say “I don’t know”.

User: Create an event for lunch with John early next week.
When testing the positive class, in the scenarios where an API command is available, to be considered a true positive, the LLM result must follow the criteria in full. Any other reply, for example, if it selects the wrong tool/command, the inputs are not well structured or hallucinates by not using an API command and replying directly to the user, is considered a False Positive. If it says "I don’t know" should be classified as a False Negative.

In turn, when testing the negative class, the LLM should answer "I don’t know" (or something similar) when there is no appropriate API command available, which will be classified as True Negative. Any situation in which the LLM hallucinates and tries to answer the user when there is no available command, should be regarded as a false positive.

With these data, we can calculate the $F_1$ score for each different prompt and compare the sensibility and sensitivity of the LLM with each different prompt.

From these values, we will calculate the $F_1$ score to be able to compare the accuracy of each prompt side-by-side.

### 2.2.3 Comparison of different Large Language Models

After the selection of the best prompt design to interact with API tools, we will compare the same system prompt with different large language models.

The selection of LLM must include proprietary, commercial models as well as free, open-source models.

GPT-4 is one of the most advanced generative large language models from OpenAI and it outperforms GPT-3 and ChatGPT 3.5 [19].

Llama-2 model [20] is another alternative to GPT-4, with 70 billion parameters. It’s an open-source model developed by Meta and launched on the 18th of July 2023.

Palm 2 [21] is a 340 billion parameter transformer-based large language model developed by Google AI and launched on the 10th of May 2023.

We will test and compare these models by applying the best three system prompts from the previous step and using the same test scenarios. Then, we will calculate the $F_1$ score that will be used to compare the different LLMs in this task.

### 2.3 Reliability and Validity

The validity of our study depends on the classification of the results of each test case. They will be classified manually into true/false positives and true/false negatives. In order to decrease the ambiguity we have created criteria to access and classify each test case. Also, each test case will be created in a way that the user prompt is unequivocal and it will be specifically designed for the LLM to select one specific API tool/command. For example, to test the use of the SEND_EMAIL API command we will test a scenario where the user asks the LLM to send an e-mail specifically to someone. All the tests will be divided into a positive class, where there is an API command or tool available to answer the user and a negative class, where there is no available command and the expected result is "I don’t know".
2.3.1 Testing the positive class

To ensure a true positive, we will use the following criteria:

- Detection of the intention of the user by selecting the right API tool/command;
- Gathering all the data needed to use that tool/command, by asking follow-up questions to the user;
- Providing all the data and the command to use the API tool in the format specified in the prompt (for example, using JSON)

To be considered a true positive, an experiment must successfully complete all three steps. A false negative can happen when the LLM answers "I do not know" even if it has an API command that would allow it to answer the user.

2.3.2 Testing the negative class

The negative class is when the LLM does not have any tool available to answer the user and the expected result is "I don’t know".

False positive can happen when the LLM does not have an API command in their list of available tools, and instead of saying "I do not know" it hallucinates. For example, it can try to call an API command that does not exist, it can answer the user immediately without using an API command, the input data for the command is not well structured when it selects the wrong API command or any other intermediary situation that is not the "I do not know" answer.

On the other hand, a true negative happens when the LLM answers "I do not know" (or some similar answer) when in fact it does not have any API command available for that request.

Then, in order to be able to compare the multiple results we standardized them using $F_1$ score. This way we can encode the sensitivity and sensibility of each prompt design and be able to compare their results side by side.

2.3.3 Final thoughts about reliability and validity

Defining a criterion provides some structure to the way we classify each test and reduces the risk of reliability problems when reproducing the results of our study. It also increases the validity of our study as we define what we consider to be a true positive and a true negative.

All the prompt designs that we will test will have the same list of API tools. The independent variables that will be tested are, at first, the design of the system prompt and later the LLM used. With different system prompts, we will test different ways to list the available API commands/tools, input arguments and output structure of the data. For each test, the words and their sequence used to simulate a user query are important to achieve similar results. The intrinsic mechanism of an LLM is sensible to changes in the user prompt as well. This way, we advise anyone who tries to reproduce our study to use the same dataset used for the tests.

In the step to find the best performing LLM for this task, there is the risk of having another LLM that could perform better using a different prompt. Also, because of our limited selection of LLMs we will never know if there is another LLM available in the market or open-source communities that would perform better. There is also the risk that the best-performing prompts with GPT-4 are not the best-performing prompts with
other models. As the time and budget are limited we decided to keep the structure of the experimentation this way, but further investigation might be needed in the future.

2.4 Ethical considerations

In our research project, we will be conducting a multiple-stage controlled experiment to explore the use of Large Language Models (LLMs) as an interface to interact with API tools in natural language. While this project primarily focuses on the technical aspects of LLMs and prompt design, there are still ethical considerations that we must take into account.

1. Sampling/Bias: In the selection of prompts and LLMs for our study, we must ensure that our choices are representative of the broader population of available models and prompt designs. This will help minimize biases that may skew the results of our study. To achieve this, we will survey the literature and open-source projects extensively to identify a diverse range of prompts and LLMs for evaluation.

2. Data Privacy: As our study involves the use of LLMs, there is a possibility that the models might generate sensitive or private information, potentially compromising the privacy of individuals. To mitigate this risk, we will only use publicly available API tools and avoid any data sets containing personal or sensitive information. For the API commands that might gather private information, we will recreate the reply of the API using dummy data. Additionally, we will ensure that the selected LLMs are not trained on any confidential or sensitive data.

3. Transparency and Reproducibility: To uphold the ethical standards of research, we will provide a clear and transparent account of our research methodology, data sources, and analysis techniques. This transparency will ensure that other researchers can reproduce our study and verify the validity of our findings.

4. Fairness and Inclusivity: As LLMs are trained on diverse data sources, they may sometimes exhibit biases or discriminatory behaviour, which could inadvertently be transferred to the API tools they interact with. We will thoroughly evaluate the performance of the selected LLMs, keeping an eye out for any signs of biased behaviour and report them to model creators or repository maintainers.

5. Risk of Harm: We must carefully consider any potential risks of harm that might arise from the use of these technologies, for that we will continuously assess the potential impact of our research on users and take appropriate measures to minimize any unintended negative consequences.

6. Participation and Consent: As our research project does not involve human participants, the issue of consent might not be directly applicable. However, we will ensure that we have the necessary permissions to access and utilize the API tools and datasets involved in our study.

In conclusion, we will make every effort to address these ethical considerations throughout the research process, ensuring that our project adheres to the highest standards of scientific integrity and promotes the responsible development and use of LLMs and API tools.
3 Theoretical background

This chapter aims to lay a solid foundation for the thesis’s main argument by exploring and explaining the underlying theories and concepts pertinent to the area of investigation. This exploration encompasses a wide range of topics, from the evolution of large language models (LLMs) to their interaction with Application Programming Interface (API) tools and the critical aspect of user intent detection in this interaction.

The chapter will also delve into the mechanics of prompt design, examining its various forms and their effectiveness in facilitating the communication between LLMs and APIs. Subsequently, we will assess the reliability of different LLMs as they employ the most efficient prompt design, further expanding our understanding of the topic.

We will enhance our comprehension of the field by analysing relevant research studies. This includes the works of Bubeck et al. [1], Parisi et al. [5], and Lazaridou et al. [6]. Each study contributes unique insights into LLMs’ capabilities, potential applications, and limitations, particularly in interfacing with APIs.

Bubeck et al. [1] provided an insightful exploration of GPT-4’s capabilities, underlining its impressive performance and inherent limitations. Their work furnishes us with a preliminary understanding of the potential of LLMs as interfaces for API interactions.

The study by Parisi et al. [5] introduced us to Tool Augmented Language Models (TALM), a significant leap in the field, demonstrating how combining LLMs with non-differentiable tools could enhance their capabilities. This research’s findings have substantial implications on how LLMs can be leveraged for complex tasks.

Lastly, Lazaridou et al. [6] took a novel approach by proposing few-shot prompting that utilises external information from the web to improve LLMs’ performance in open-domain question-answering tasks. This innovative method offers a new perspective on enhancing LLMs’ abilities and optimising their interaction with APIs.

As we journey through this chapter, the theories and research discussed will equip us with a comprehensive understanding of the subject, ultimately highlighting the research gaps and identifying the areas requiring further investigation.

3.1 Large Language Model (LLMs)

LLMs are machine learning technology that helps produce written text resembling human language. Early models of NLP used methods such as Bag-of-Words (BoW) and n-grams to generate text, but these methods often struggled with comprehending context and producing content that is truly rich and meaningful. However, recent advancements in the field have led to the development of better models like recurrent neural networks (RNNs), long short-term memory (LSTM), and transformer models such as Generative Pre-trained Transformer (GPT), Bidirectional Encoder Representations from Transformers (BERT) [4], and Transformer-XL. These advancements have greatly improved the quality of the generated text.

3.1.1 Importance and Types of LLMs:

LLMs are incredibly powerful because they can generate contextually rich and coherent text, understand and reply to prompts in a meaningful way, and perform tasks across various domains without the need for explicit task-specific training data. This is particularly significant because it opens up the potential for more natural and intuitive human-computer interactions. When it comes to language models (LLMs), there are a few different categories to consider based on their application.
1. Continuing the text LLMs: Models like GPT-3 and GPT-4 fall under this category. These models generate responses by "continuing the text" given to them. They are not explicitly designed to maintain a conversational state or dialogue history but can be coaxed to do so by formatting the input prompt to include conversation history.

2. Dialogue Optimized LLMs: Some models are specifically trained for dialogues, such as LaMDA [22] by Google and DialoGPT [23] by Microsoft. These models are optimised for maintaining a conversational state over multiple turns of a conversation.

3.1.2 Detailed Overview of GPT-3 and GPT-4:
OpenAI has developed two advanced versions of the Generative Pretrained Transformer series, GPT-3.5 and GPT-4. These versions are an improvement over their predecessors, as they possess greater scale and generalisation capabilities. Using a transformer-based architecture, these models can effectively handle long-range dependencies in text and diverse linguistic phenomena. Their training involves unsupervised learning, where they learn to predict the next word in a sentence by analysing a large collection of internet text. Hence, LLMs are capable of generating text that closely resembles human language by predicting the likelihood of a particular word based on the words that come before it [2]. It is particularly known for its "few-shot learning" ability, where it can generate accurate and reliable responses after being given a small number of task examples in the prompt [8][12].

3.1.3 LLMs underlying Theories: Transformer Architecture, Attention Mechanism, and Unsupervised Learning
The transformer architecture is a type of model architecture used in NLP that handles sequential data using self-attention mechanisms [24]. Unlike previous models that processed words sequentially, the transformer processes all words in parallel, making it more efficient.

When generating output, the attention mechanism helps the model associated with specific parts of the input texts by assigning varying weights to different words. As a result, contextually accurate responses can be generated.

The process of training models involves utilizing unsupervised learning. In this approach, the models are exposed to vast amounts of text data, allowing them to predict the next word in a given sentence. As a result, the models can generate text that closely resembles human writing style and is coherent in terms of theme and language.

3.1.4 Zero-shot, Few-shot Prompts, and Fine-Tuning:
Zero-shot and few-shot prompts are methods that a model uses to handle new tasks that it was not explicitly trained for. Zero-shot learning is when the model is given a task without any prior experience, while few-shot learning is when the model has a few examples of the task to guide its output.

Fine-tuning refers to the process of training a pre-existing model on specific tasks to improve its ability to produce more precise and dependable responses for those tasks [3]. For example, if LLMs are fine-tuned using a medical dataset, they could generate medically accurate text more effectively. Nevertheless, the model’s capacity to generalise
might be affected as it may become too focused on the particular task it was fine-tuned for, leading to overfitting [25].

3.1.5 Usage and Applications of LLMs in Various Fields:

LLMs have many uses in various fields. They are used in fields such as customer service for answering queries [2, 26], content generation for generating articles or reports [27, 28], personal assistants for providing contextual responses [29], and helping medical professionals diagnose, symptom check, and medical research, leading to better-informed decisions [30]. LLMs are incredibly versatile, offering contextual responses and educational support in a wide variety of subjects [31]. They can even assist with programming tasks [2], language translation [24], and even generate creative content like poetry or music [2].

3.2 Prompt Design

Prompt design is a critical aspect of utilising LLMs, acting as a medium to bridge the gap between human users and these advanced AI systems. Prompts’ fundamental role is guiding the LLMs to produce desired, coherent, and accurate responses to tasks or queries. They form the basis of the user’s initial instructions that stimulate the learned knowledge of pre-trained language models during their application stage [2].

Recently, the emergence of ‘prompt design’ and ‘prompt engineering’ as a focused field attests to its growing significance in utilising Natural Language Processing [3]. This subfield is concerned with crafting precise instructions that an AI model, such as an LLM, uses to generate specific responses.

3.2.1 Prompt Design Methodologies

Several methodologies are considered when designing prompts. These include:

1. Prompt Chaining: This technique leverages the outputs generated by a model to form subsequent prompts. This chain of dialogue creates a more coherent, engaging, and creative interaction with the model, leading to superior text generation [3].

2. Prompt Tuning: This methodology adapts and optimises prompts for specific tasks or domains. This involves using smaller, specialised datasets to enhance the accuracy and diversity of generated texts while reducing the noise and bias present in the public dataset [3].

3. Prompt Testing: This method involves comparing and measuring the quality of prompts using various metrics and criteria, both human and automated. It includes rating, ranking, feedback, analysis, and testing to ensure the generated texts align with the intended outcome [3].

3.2.2 Practical Examples

A zero-shot system prompt can be seen in Listing 4. Were the LLM is given a prompt without any examples of the expected reply.
In this case, the language model will determine how to form the correct API call using
the given prompt without prior examples. Zero-shot prompts can be less effective, as they
do not provide prior examples to guide the model’s response.

Listing 5 presents a few-shot prompt design example.

As shown in Listing 5, with the help of a few-shot prompt, the language model can
use the previous examples to help it figure out the correct API call when asked about
the weather in Tokyo. Few-shot prompts often yield more reliable results than zero-shot
prompts, as they guide the model on the correct output. However, they also require more
effort to set up, as they necessitate the creation of accurate and representative examples.

3.2.3 Strengths and Limitations of Various Prompt Designs

The strength of prompt designs lies in their ability to extract the latent task-specific knowl-
edge embedded in language models by using natural language as a medium. This makes
complex tasks accessible to individuals without specialised technical knowledge [3]. The
SK framework, in particular, enables the crafting of complex, configurable, and testable
chains of prompts that can produce more precise outputs [3].

However, there are limitations to consider. While prompt chaining can enhance di-
ologue coherence and creativity, it might also amplify biases in the model’s responses.
Prompt tuning, though effective for specific tasks, may be less robust when faced with a
broader range of tasks. Additionally, the effectiveness of prompts can be highly sensitive
to the exact wording or phrasing used, potentially leading to inconsistent results [3].

Prompt design is a dynamic field that will continue to evolve alongside advancements
in LLMs and NLP. It is crucial to continuously refine these strategies while acknowledging
potential risks and ethical considerations to ensure the responsible application of AI.
3.3 Application Programming Interface (API) Tools

APIs are essential building blocks in software development that facilitate seamless communication among different computer programs and systems. They function by providing a simplified and abstracted layer that conceals the intricate internal workings of these programs, thereby making it easier for developers to integrate and link disparate software systems. This leads to the development of more robust and scalable applications [32]. Developers can interact with the functionalities of a system without fully understanding its complexities, thanks to this abstraction layer. APIs can be customized to meet the specific needs of particular systems or developed as a standard, promoting seamless communication and compatibility across different systems [33].

3.3.1 Importance and Usage in Software Development

The principles of API design, primarily abstraction and consistency, have profound implications in software development. Abstraction enables developers to focus on more critical aspects of their software, providing a consistent user experience despite internal changes in the system. Consistency across different parts of an API ensures developers can comprehend and utilise new functionalities by adhering to familiar patterns and conventions [34]. APIs, thus, allow developers to concentrate on their application’s core logic rather than becoming entangled in the minutiae of the system they are interacting with.

The advent of Web APIs has revolutionised software development by endorsing the widespread use of microservices architecture. This architectural style offers increased flexibility, scalability, and resilience, enabling more robust and adaptable applications. Consequently, APIs have become indispensable in modern software development, empowering developers to develop more versatile and powerful applications.

3.3.2 Overview of APIs

Various types of APIs are designed for specific purposes. One of the most commonly used forms is Web APIs, which facilitate smooth communication between distinct applications or systems through the Internet. These APIs employ specific protocols such as REST or SOAP to enable efficient exchange of data and requests [32]. Meanwhile, Remote APIs facilitate communication between applications or systems operating on different machines in a distributed environment, typically using Remote Procedure calls (RPC) or Remote Method Invocation (RMI) protocols [35].

3.3.3 Interaction of APIs with LLMs

While LLMs like GPT-3 and GPT-4 have an extensive knowledge base, they are currently limited to the scope of their training dataset. They lack domain-specific knowledge unique to a company’s database and cannot interact with APIs on their own [15][16][17][18]. However, with suitable adjustments and enhancements, LLMs can interact with APIs to retrieve data and increase their ability to interact with other systems. For instance, if a user asks an LLM-powered chatbot for the current weather, the LLM alone cannot fulfill the request because it does not have real-time data. However, suppose the LLM is programmed to interact with a weather API. In that case, it can formulate a suitable API request, receive the current weather data response, and present it to the user in an understandable format. This interaction demonstrates how APIs can significantly extend...
the capabilities of LLMs, enabling them to provide real-time data and access to external functionalities.

3.4 Tool Learning

Tool learning represents a critical dimension in the evolution of LLMs. Drawing from the analogy of how children learn to use tools, this approach focuses on training foundation models such as GPT-4 to comprehend and proficiently utilise APIs and other tools [36]. LLMs must possess the knowledge and proficiency to utilise modern API tools efficiently in order to effectively complete tasks in a natural language environment, owing to their intricate and versatile nature.

LLMs serve as the cognitive backdrop against which tool learning plays out, mirroring how human cognition aids in learning and applying tools. Just as humans utilise tools to simplify and enhance task performance, LLMs are expected to use API tools to execute tasks more efficiently and accurately.

3.4.1 Different Tool Learning Methodologies

Tool learning in the context of LLMs is primarily realised through prompting methods, which can be bifurcated into two main types: zero-shot and few-shot prompting [2].

1. Zero-shot Prompting: This methodology describes API functionalities, input/output formats, and parameters to the model without providing any examples of the task at hand. Therefore, the model is expected to interpret and execute instructions based solely on the API definitions. This method’s success heavily depends on the complexity of the API and the LLM’s capability to grasp and execute the instructions. For instance, zero-shot prompting might instruct an LLM to interact with Hugging Face’s API to generate a sentiment analysis of a given text.

2. Few-shot Prompting: The model is provided with a few examples or demonstrations along with the task instruction. The LLM learns to mimic human behaviour by using tools based on these demonstrations. For example, when interacting with Langchain’s API to translate a text, the model is given a few examples of English phrases and their corresponding translations in another language, guiding it to perform the task accurately.

3. One-shot Prompting: The model is provided with just one example of how it should answer the user along with the system prompt.

Alternative methods include an intermediate stage of tool selection or fine-tuning [11].

- Tool selection involves retrieving a more relevant set of tools for a specific task, reducing the input length and potentially improving model performance.

- Fine-tuning optimises models using concrete tool-use examples and human experiences, which can reduce input length and speed up model inference.

3.4.2 Strengths and Limitations of Various Tool Learning

Prompting, especially few-shot prompting, has shown promise in teaching LLMs about tools with minimal human effort, offering flexibility to adjust to tool changes or functionalities [19]. Despite its strengths, prompting faces challenges such as dependency on
model capabilities and limitations on input context length, which can impede its effectiveness for more complex or extensive toolsets.

Alternative solutions like tool selection and fine-tuning come with their own sets of strengths and limitations. Although efficient in reducing input length and improving performance, tool selection may not always be the most suitable tool for a given task. On the other hand, fine-tuning can alleviate the need for tool definitions in the input and speed up model inference. However, it requires extensive human annotations or demonstrations and the need to retrain models when tools are updated [11].

As this field continues to evolve, the balance between these strengths and limitations and the continual improvement in tool learning methodologies will play a pivotal role in the effective application of LLMs in interacting with API tools.

3.5 Performance Evaluation Criteria

A selection of appropriate evaluation criteria is crucial for accurately assessing the performance of LLMs. These metrics provide quantitative measures that help identify a model’s strengths and weaknesses in terms of prediction accuracy. In our study, which aims to explore the potential of LLMs in conjunction with API tools, the selection of evaluation metrics is particularly significant.

Through a comprehensive examination of critical performance metrics such as accuracy, precision, and recall, models’ predictive prowess can be revealed. These criteria will aid in measuring and fine-tuning the models based on their interaction with APIs, as well as in identifying the most promising prompt designs that align with optimal model performance. Therefore, it is essential to carefully consider and utilize appropriate evaluation criteria in determining the reliability and overall performance of LLMs.

3.5.1 Comprehending Binary Classification and the Confusion Matrix

A model’s effectiveness cannot be evaluated by its overall accuracy but rather requires the use of a confusion matrix. This tool allows a more comprehensive assessment of a model’s data classification efficiency by analyzing its True Positives, True Negatives, False Positives, and False Negatives. Through careful examination of these components, a focused evaluation of a model’s classification accuracy can be made. The confusion matrix surpasses a mere overall accuracy measurement and it can be used to deconstruct a model’s performance.

<table>
<thead>
<tr>
<th></th>
<th>Predicted: Positive</th>
<th>Predicted: Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual: Positive</td>
<td>True Positive (TP)</td>
<td>False Negative (FN)</td>
</tr>
<tr>
<td>Actual: Negative</td>
<td>False Positive (FP)</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>

3.5.2 Precision, Recall and their Importance in Evaluation

Precision and recall are fundamental metrics that can be accurately assessed using the confusion matrix, which is a crucial tool. Measuring the accuracy of a model in predicting positive results is done through the metric of Precision. Consequently, it evaluates the precision of anticipated affirmative results [37]. Organisations can effectively analyse the identification process and identify areas that require improvement through a meticulous examination of precision levels. Formally, Precision is defined as shown in Equation 1 [38]:

\[
\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}
\]
When analyzing data, it is crucial to prioritize recall as it gauges the model’s ability to identify true positive cases. Recall determines how effectively the model detects all the genuinely positive instances in the dataset [37]. It is formally defined as in Equation 2 [38]:

\[ R = \frac{TP}{TP + FN} \]  

(2)

When utilizing APIs, it is of utmost importance to exercise prudence in order to avoid expensive errors. The potential exists for misidentification of real events or the neglect of genuine occurrences. It is essential, therefore, to give priority to precision and recall. Precision plays a critical role in evading false predictions, while recall is vital in the avoidance of any missed accurate predictions. The achievement of API management efficiency is dependent upon achieving a balance between these metrics.

### 3.5.3 F-Score: balancing Precision and Recall

The F-Score is a quantitative metric utilized to gauge the efficacy of a model based on a specific dataset [39]. This metric is particularly applicable when examining binary classification systems that categorize instances as either ‘positive’ or ‘negative’. The F-Score measures both precision and recall in a model. It’s important to note that the highest possible value for the F-Score is 1, representing the best possible precision and recall, while the lowest possible value is 0. F-Score is defined as shown in Equation 3 [39] [38]:

\[ F - \text{Score}, F_{\beta} = \frac{(1 + \beta^2) \cdot P \cdot R}{\beta^2 \cdot P + R} \]  

(3)

Using the value of \( \beta > 1 \) in the F-Score formula signifies favouring recall over precision, applicable in situations where FN is more detrimental than FP. Conversely, favouring precision over recall involves a \( \beta < 1 \) and is used when the negative impact of FP is more significant than that of FN.

In a special case when \( \beta = 1 \), also called F1 Score [39]. This score is a type of F-score where recall and precision have equal importance in the harmonic mean. F1 Score is formally defined as depicted in Equation 4 [39] [38]:

\[ F1 - \text{score} = 2 \cdot \frac{(P \cdot R)}{(P + R)} \]  

(4)

The assessment of a model’s effectiveness is significantly enhanced by considering the F1 score, which serves as an indicator of its ability to maintain a balance between false positives and false negatives. This metric assumes greater significance when the consequences of both types of errors are similar. In the context of interactions between LLMs and API tools, precision and recall must be evaluated in tandem to ensure optimal balance. Therefore, the F1 score is a dependable and efficacious measure of overall performance.
3.5.4 Reliability Measures in Research Experiments

Reliability measurements are necessary when evaluating the performance of LLMs. These measurements establish the stability and consistency of model outputs over time. A high-reliability measurement indicates a more reliable model, which is crucial for the practical applicability of these models.

3.5.5 Linking Evaluation Metrics to Large Language Models

When dealing with LLMs interacting with API tools, three scores are relevant. In cases where incorrect API calls are just as detrimental as missed API calls or where both cases have an equal impact, the $F_1$ score should be utilized. However, if incorrect API calls are deemed worse (perhaps due to their use of unnecessary resources), an F-score with $\beta < 1$ would be a prudent choice. Conversely, if missed API calls are considered worse (potentially resulting in incomplete user command execution), then a balanced F-score with $\beta > 1$ should be employed. The selection of the appropriate scoring methodology for our research will be further expounded upon in subsequent chapters. It is vital to select a scoring methodology that suits the specific circumstances of the study at hand to ensure accurate and reliable results. The chosen scoring methodology should be clearly explained and justified in any subsequent publications on the research findings. The careful consideration of these scoring methodologies is paramount to the success of the study and the validity of its results.

3.6 Research Studies and Research Gap Identification

Bubeck et al. [1] examined the performance of an early version of GPT-4 across domains, highlighting its limitations and potential for further investigation. Parisi et al. [5] introduced Tool Augmented Language Models (TALM) and showed its superiority over non-augmented models on MathQA. Lazaridou et al. [6] proposed a few-shot prompting approach to condition LLMs on external information, improving performance in open-domain question-answering tasks. Few-shot learning can improve a model’s ability to generate API requests and process responses. Schick et al. [7] introduced the Toolformer, which learns to use external tools via simple APIs to improve performance but has limitations. To evaluate this integration, consider accuracy, quality of API calls, and user satisfaction. Challenges include ambiguous queries, security, privacy, and resource management. However, natural language APIs can lead to more user-friendly apps. Effective prompts are crucial for LLM performance, with criteria such as gathering necessary user data and eliciting meaningful responses.
4 Research project – Implementation

This section will explain how we set up and implement our experiment, focusing on generating standardised API prompts, user prompt scenarios, and the experimentation environment.

4.1 Literature Review

As mentioned in the methodology section, the first step is to gather different prompt designs from the literature and open-source projects. We conducted a literature review of the available literature using OneSearch from Linnaeus University Library and Google Scholar. We used logical operators to refine our search results: 'Artificial General Intelligence' OR 'General Language Model' AND 'Tool Learning' AND 'API tool'.

We adopted various selection criteria in our process, encompassing the inclusion of our predetermined keywords within the document’s title, abstract, or metadata keywords. To guarantee relevance and currentness, we only considered documents published within the time frame from 2019 to 2023. The nature of the documents had to align with the categories of academic research, conference proceedings, review articles, reports, or dissertations. All documents had to be articulated in English and were required to contain meticulously detailed designs of prompts. The exclusion criteria were the following: If a paper were a duplication of other studies already identified in our search results. Documents that were not entirely accessible via Linnaeus University’s subscription services. We excluded any study that did not clearly outline its methodology. Papers lacking sufficient documentation of the prompts used.

As most of the recent contributions in this field using LLMs are in the open-source community, we have decided also to search in github.com for the most popular LLMs projects searching for "Large language models", "LLM" and "GPT" and ordering the results by best match and stars. We have found some interesting projects like LangChainJS and Auto-GPT that use LLMs to interact with API tools and task planning.

4.2 Expansion of prompt designs using GPT-4

To further expand the tested system prompts we used GPT-4 to generate new system prompts to instruct a LMM to interact with a list of API commands.

The idea to use GPT-4 to formulate some prompt designs resides in the way that LLMs use tokenization of words to match and predict the next generated token. Using the model to generate the prompts, we may see some residues of its own latent space of how it represents API functions or commands. In this way, we may leverage some tokens and patterns to test if they perform better and give suggestions on how to improve prompt design. For the AI-generated prompts using GPT-4, we have used the instruction stated in listing 6 as input to the model.
Write a system prompt for GPT-4 in which you list 20 different external API commands that GPT-4 can use to answer user requests. Make sure to include a way for GPT-4 to know the input arguments for those commands. Make sure that those commands can be easily parsed by another program that will do the API request. Make sure that the command occupies only one line. After detecting the user's intention, the prompt must instruct GPT-4 to ask all the questions needed to gather all the data to use a specific tool. When GPT-4 have all the data, it must answer with the command selected and all its input arguments from the data gather from the user.

Listing 6: Prompt used in GPT-4 model to create prompts with the intention to interact with API tools or commands

The prompt shown in Listing 6, will be run multiple times until results start to be repeated, and there are no new features in the suggested prompts.

4.3 API commands standardization

For all the tested prompts we will test the LLM with the same generic API tools/commands:

- Send e-mail: send an e-mail with a subject, recipient e-mail and text message;
- Read calendar events for a specific date: get the calendar events from the user for a specific date;
- Create an event in the user calendar: create an event with a name, date and time;
- Create a reminder: with a name, date and time;
- Get current date and time;
- Get weather for a date and a place using the name of the city and country;
- Get the news for a topic and date;
- Convert text-to-speech: using some text;
- Current stock price of a company: using its stock symbol;
- Generate image: from a prompt;

In order to achieve that we need to standardize all the prompts to use the same API commands.

The subjective choice of the API commands for this project was driven by the popularity of some of the APIs and the goal of using the large language model as an AI assistant. For the first four commands, we consider that the user is authenticated. For the calendar commands, we consider that the user only has one calendar. Both situations are outside the scope of our experiments.
4.4 Testing scenarios for user prompt

To test the system prompts, we need to create user prompts that will mimic a user interaction and will test all and each of the API commands listed in the system prompt. For each API command available in the system prompt, we need to generate the multiple ways a user can ask a question. To achieve that, we started with more open instructions where we just signal the API command that we want to use, for example, "I want to send an e-mail" and we slowly include more information in the user prompts, for example, "I want to send an e-mail to jonh@example.com, with Tomorrow’s meeting in the subject and The meeting is cancelled in the e-mail message", where we include all the information needed as input arguments for the "send_email" API command. This way we can cover most of the scenarios and also test the ability of the LLM to ask follow-up questions when there is not enough information to execute an API command (like in the first example). For each API command, we generated on average 20 user prompts that will cover most of the scenarios and it is a balance between coverage and time and cost to run the models.

4.5 Experimentation Environment

To conduct the experiments we have created a CLI (Command line interface) program in NodeJS, see our Github repository ¹. This program will facilitate the process of manually testing all the system prompts and user queries to use an API command. It allows us to select the model to test, the system prompt and which user test scenario to test. The system prompts are saved in the folder /system_prompts in txt format and the user prompts for each test scenario are saved in the /user_prompts folder. Each test scenario file is made to test a specific API command, and inside it, we have one user prompt per line. For each test scenario, the CLI program will select a user prompt from the file to start the interaction with the LLM. In the situation that the LLM asks for more information, we will provide that information on-the-fly and evaluate the model response.

At the end of one experiment, we press ENTER to signal the program that the current experiment ended, and it asks us for the score to give to that experience from 0 to 3 (as defined in the criteria in Methodology 2.2.2). We then move on to the next combination of system prompt and user prompt.

All the interactions are recorded in a text file and CSV file for posterior analysis.

To speed up the implementation, we have used the hugging face inference endpoint for the LLama-2 model from Meta (more specifically the model upstage/Llama-2-70b-instruct-v2 [40]). For the PaLM model we used Vertex AI from Google cloud with the PaLM Bison chat model [41]. For the GPT-4 model, we used the OpenAI API with the gpt-4-0613 model [42]. All these solutions in the cloud do not require us to have specialized equipment to run these models and facilitate the testing and solution implementation using these huge models which require machines with a lot of memory and GPUs. However, they incur costs, which we will talk about in the discussion section.

¹https://github.com/bmmseven/llm-api-experiences
5 Results

This section presents the experiment results of each stage, including a literature review, testing the best prompt design, and evaluating the performance of the best prompt design across various LLMs.

5.1 Literature Review and prompt discovery

As mentioned in the implementation section, we conducted a literature review of the available literature using various search engines, which included Google Scholar and Linnaeus University’s OneSearch. Our sources included peer-reviewed articles, reports, and theses.

After the initial search by combining keywords using the logical operator, ‘AND’ and ‘OR’: ‘Artificial General Intelligence’ or ‘General Language Model’ and ‘Tool Learning’ and ‘API tool’, we got 17,300 results. To improve the accuracy of our search results, we carefully choose the most relevant literature and used inclusion and exclusion criteria mentioned before to select our papers.

From the literature review, the only article that meet all the requirements was “Sparks of Artificial General Intelligence: Early Experiments with GPT-4” [1].

For that reason, we expanded the search for publicly available Github projects. While searching Github.com we used the following keywords: "LLM" and "GPT" sorting the results by the number of Github stars, to find the state-of-the-art open-source projects. There were 81.4k results for "GPT" and 12.7k results for "LLM". We have selected the first results: LangchainAI and autoGPT projects for further exploration. The autoGPT project matched our requirements as its goal is to automatize GPT-4 including task planning and use of API commands to interact with a computer or the internet and the system prompt was easily extractable from the code. LangchainAI, as a framework to create AI agents, was used to speed up the implementation to test the system prompts with the use case scenarios.

In summary, from the literature and GitHub public repositories we have selected the following prompt designs to be used in the controlled experiment:

- A system prompt inspired by the paper “Sparks of Artificial General Intelligence: Early Experiments with GPT-4” [1] where the API tools and input arguments are defined in plain text with its description;

- A system prompt inspired by AutoGPT [43], with command label, name and arguments for each command or tool;

Additionally, to expand the number of prompts to test, we have used GPT-4 from OpenAI to generate new system prompts for this task.

Using the prompt from Listing 6, we run the query multiple times and stopped after 17 times, as the results started to be repeated and there were no new features in the suggested prompts. See Appendices for all the generated prompts C D E F G H I J K L M N O P Q R S.

From these 17 prompts, we randomly select three prompts that we will use for the study. The selected prompts are: D (AIgenerated2_zeroshot), G (AIgenerated5_zeroshot), and O (AIgenerated13_zeroshot). The prompts AIgenerated2_zeroshot and AIgenerated5_zeroshot were used to create one-shot prompts T (AIgenerated2_oneshot) and U (AIgenerated5_oneshot), by adding some examples of successful interactions with the user inside the system prompt. In theory, one-shot prompts make it easier for the
LLM to understand the structure of the final output and how to integrate the user’s answers into the final output. These prompts are the AIgenerated5_oneshot (prompt U) and AIgenerated2_oneshot (prompt T).

In summary, we will test the following system prompts:

- A system prompt inspired by the paper “Sparks of Artificial General Intelligence: Early Experiments with GPT-4” [1] where the API tools and input arguments are defined in plain text with its description (referenced as "Sparksofagi prompt");

- A system prompt inspired by AutoGPT [43], with command label, name and arguments for each command or tool (referenced as "AutoGPT_zeroshot prompt");

- AIgenerated5_zeroshot (G);
- AIgenerated13_zeroshot (O);
- AIgenerated5_oneshot (U);
- AIgenerated2_oneshot (T);
- AIgenerated2_zeroshot (D);

5.2 Testing prompt designs

As defined in the methodology, we will test multiple-use scenarios for each system prompt. Each use scenario will have user queries that must trigger a specific API command, which will be identified by the API command we expect to use. Additionally, there will be a use scenario with no available API commands, triggering an "I do not know" answer (NOT AVAILABLE use scenario). To find the best prompt design, we will run all the use scenarios in GPT-4, so the only variable changing is the system prompt.

Below are the results for each prompt design and use scenario (API command), where we classify and count the number of True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN). Additionally, we have calculated the precision, recall and $F_1$-Score for each use scenario. At the end of each table, there is the cumulative sum of all the use scenarios for a specific design prompt and its respective $F_1$ score that will be used later for comparison of design prompts.

In Table 5.1 we show the results for the system prompt design based on the AutoGPT project. The evaluation encompasses a sum of 970 tests executed across several API functionalities. Cumulatively, the findings reveal a TP rate of 52.99%, a TN rate of 29.28%, an FP rate of 16.80%, and a minimal FN rate of 0.93%. The compiled data affirms an average $F_1$ score of 0.86, reflecting the aggregated performance metrics across the APIs.
<table>
<thead>
<tr>
<th>APIs</th>
<th>TN</th>
<th>FN</th>
<th>FP</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERT_TO_SPEECH</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>54</td>
<td>0.90</td>
<td>1.00</td>
<td>0.95</td>
</tr>
<tr>
<td>CREATE_CALEDAR</td>
<td>0</td>
<td>0</td>
<td>22</td>
<td>38</td>
<td>0.63</td>
<td>1.00</td>
<td>0.78</td>
</tr>
<tr>
<td>CREATE_REMINDER</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>53</td>
<td>0.88</td>
<td>1.00</td>
<td>0.94</td>
</tr>
<tr>
<td>GENERATE_IMAGE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_CALEDAR</td>
<td>17</td>
<td>0</td>
<td>15</td>
<td>28</td>
<td>0.65</td>
<td>1.00</td>
<td>0.79</td>
</tr>
<tr>
<td>GET_CURRENT_DATE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_CURRENT_STOCK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>59</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_NEWS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_WEATHER</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>58</td>
<td>0.94</td>
<td>1.00</td>
<td>0.97</td>
</tr>
<tr>
<td>NOT_AVAILABLE</td>
<td>296</td>
<td>0</td>
<td>55</td>
<td>7</td>
<td>0.11</td>
<td>1.00</td>
<td>0.20</td>
</tr>
<tr>
<td>SEND_EMAIL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Cumulative</strong></td>
<td>313</td>
<td>0</td>
<td>109</td>
<td>537</td>
<td>0.83</td>
<td>1.00</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Table 5.1: using the AutoGPT_zeroshot prompt designs

Table 5.2 illustrates the outcomes derived from the 'Sparksofagi' prompt design, applied to assess GPT-4’s capability in discerning diverse API commands. As delineated in the table, a total of 959 tests were undertaken, encompassing various API functionalities. Aggregated findings highlight a TP rate of 56.0%, a TN rate of 32.64%, an FP rate of 11.37%, and a negligible FN rate. Collating these metrics, the APIs demonstrated a cumulative F1 score of 0.91, underscoring the proficiency of the implemented design in API detection tasks.

<table>
<thead>
<tr>
<th>APIs</th>
<th>TN</th>
<th>FN</th>
<th>FP</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERT_TO_SPEECH</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>54</td>
<td>0.90</td>
<td>1.00</td>
<td>0.95</td>
</tr>
<tr>
<td>CREATE_CALEDAR</td>
<td>0</td>
<td>0</td>
<td>22</td>
<td>38</td>
<td>0.63</td>
<td>1.00</td>
<td>0.78</td>
</tr>
<tr>
<td>CREATE_REMINDER</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>53</td>
<td>0.88</td>
<td>1.00</td>
<td>0.94</td>
</tr>
<tr>
<td>GENERATE_IMAGE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_CALEDAR</td>
<td>17</td>
<td>0</td>
<td>15</td>
<td>28</td>
<td>0.65</td>
<td>1.00</td>
<td>0.79</td>
</tr>
<tr>
<td>GET_CURRENT_DATE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_CURRENT_STOCK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>59</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_NEWS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_WEATHER</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>58</td>
<td>0.94</td>
<td>1.00</td>
<td>0.97</td>
</tr>
<tr>
<td>NOT_AVAILABLE</td>
<td>296</td>
<td>0</td>
<td>55</td>
<td>7</td>
<td>0.11</td>
<td>1.00</td>
<td>0.20</td>
</tr>
<tr>
<td>SEND_EMAIL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Cumulative</strong></td>
<td>313</td>
<td>0</td>
<td>109</td>
<td>537</td>
<td>0.83</td>
<td>1.00</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Table 5.2: using the Sparksofagi prompt designs

Table 5.3 showcases the results of experiments using the ’AI_generated_2_oneshot’ prompt design, gauging GPT-4’s efficacy in discerning various API commands. The empirical data elucidates that a total of 960 tests were conducted across diverse API functions. Cumulatively, the findings depict a TP rate of 53.96%, a TN rate of 34.69%, an FP rate of 9.06%, and an FN rate of 2.29%. Collating these statistics, the APIs presented an average $F_1$ score of 0.90, signifying the proficiency of the tested design in the API recognition task.

25
<table>
<thead>
<tr>
<th>APIs</th>
<th>TN</th>
<th>FN</th>
<th>FP</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERT_TO_SPEECH</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>58</td>
<td>0.97</td>
<td>1.00</td>
<td>0.98</td>
</tr>
<tr>
<td>CREATECALENDAR</td>
<td>0</td>
<td>11</td>
<td>19</td>
<td>30</td>
<td>0.61</td>
<td>0.73</td>
<td>0.67</td>
</tr>
<tr>
<td>CREATE_REMINDER</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>57</td>
<td>0.98</td>
<td>0.97</td>
<td>0.97</td>
</tr>
<tr>
<td>GENERATE_IMAGE</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>40</td>
<td>0.77</td>
<td>0.87</td>
<td>0.82</td>
</tr>
<tr>
<td>GETCALENDAR</td>
<td>11</td>
<td>2</td>
<td>11</td>
<td>36</td>
<td>0.77</td>
<td>0.95</td>
<td>0.85</td>
</tr>
<tr>
<td>GET_CURRENT_DATE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>59</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_CURRENT_STOCK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_NEWS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_WEATHER</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>61</td>
<td>0.97</td>
<td>1.00</td>
<td>0.98</td>
</tr>
<tr>
<td>NOT_AVAILABLE</td>
<td>320</td>
<td>0</td>
<td>38</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SEND_EMAIL</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>57</td>
<td>0.97</td>
<td>0.98</td>
<td>0.97</td>
</tr>
<tr>
<td><strong>Cumulative</strong></td>
<td>333</td>
<td>22</td>
<td>87</td>
<td>518</td>
<td>0.86</td>
<td>0.96</td>
<td>0.90</td>
</tr>
</tbody>
</table>

Table 5.3: using the AIgen2_oneshot prompt designs

In Table 5.4, we present results derived from the 'AIgenerated_5_oneshot' prompt design, testing GPT-4's proficiency in discerning various API commands. The dataset encompasses 861 tests, examining a plethora of API functionalities. Cumulatively, the APIs displayed a TP rate of 54.01%, a TN rate of 31.34%, an FP rate of 13.36%, and an FN rate of 1.28%. Taking these figures into account, a commendable mean F₁ score of 0.88 is achieved.

<table>
<thead>
<tr>
<th>APIs</th>
<th>TN</th>
<th>FN</th>
<th>FP</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERT_TO_SPEECH</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>58</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>CREATECALENDAR</td>
<td>0</td>
<td>0</td>
<td>37</td>
<td>23</td>
<td>0.38</td>
<td>1.00</td>
<td>0.55</td>
</tr>
<tr>
<td>CREATE_REMINDER</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>59</td>
<td>0.98</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td>GENERATE_IMAGE</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>52</td>
<td>1.00</td>
<td>0.96</td>
<td>0.98</td>
</tr>
<tr>
<td>GETCALENDAR</td>
<td>24</td>
<td>6</td>
<td>10</td>
<td>20</td>
<td>0.67</td>
<td>0.77</td>
<td>0.71</td>
</tr>
<tr>
<td>GET_CURRENT_DATE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_CURRENT_STOCK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_NEWS</td>
<td>0</td>
<td>2</td>
<td>43</td>
<td>15</td>
<td>0.26</td>
<td>0.88</td>
<td>0.40</td>
</tr>
<tr>
<td>GET_WEATHER</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>57</td>
<td>0.92</td>
<td>1.00</td>
<td>0.96</td>
</tr>
<tr>
<td>NOT_AVAILABLE</td>
<td>240</td>
<td>0</td>
<td>18</td>
<td>1</td>
<td>0.05</td>
<td>1.00</td>
<td>0.10</td>
</tr>
<tr>
<td>SEND_EMAIL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Cumulative</strong></td>
<td>270</td>
<td>11</td>
<td>115</td>
<td>465</td>
<td>0.80</td>
<td>0.98</td>
<td>0.88</td>
</tr>
</tbody>
</table>

Table 5.4: using the AIgen5_oneshot prompt designs

Table 5.5 shows the results for the system prompt design adapted from the 'AIgenerated5_zeroshot' prompt design on GPT-4’s performance to discern different API commands. As the table elucidates, the dataset encompasses a total of 960 tests distributed across various API functions. Combined, the results reveal a TP rate of 51.77%, a TN rate of 39.58%, an FP rate of 5.94%, and an FN rate of 2.71%. Consolidating these findings, the examined APIs demonstrated an average F₁ score of 0.92, accentuating the robustness of the implemented design in the API recognition task.
<table>
<thead>
<tr>
<th>APIs</th>
<th>TN</th>
<th>FN</th>
<th>FP</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERT_TO_SPEECH</td>
<td>0</td>
<td>1</td>
<td>58</td>
<td>58</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>CREATE_CALENDAR</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>27</td>
<td>0.45</td>
<td>1.00</td>
<td>0.62</td>
</tr>
<tr>
<td>CREATE_REMINDER</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>59</td>
<td>0.98</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td>GENERATE_IMAGE</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>49</td>
<td>1.00</td>
<td>0.89</td>
<td>0.94</td>
</tr>
<tr>
<td>GET_CALENDAR</td>
<td>25</td>
<td>7</td>
<td>4</td>
<td>24</td>
<td>0.86</td>
<td>0.77</td>
<td>0.81</td>
</tr>
<tr>
<td>GET_CURRENT_DATE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_CURRENT_STOCK</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>59</td>
<td>1.00</td>
<td>0.98</td>
<td>0.99</td>
</tr>
<tr>
<td>GET_NEWS</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>43</td>
<td>0.88</td>
<td>0.81</td>
<td>0.84</td>
</tr>
<tr>
<td>GET_WEATHER</td>
<td>2</td>
<td>3</td>
<td>58</td>
<td>5</td>
<td>0.95</td>
<td>1.00</td>
<td>0.98</td>
</tr>
<tr>
<td>NOT_AVAILABLE</td>
<td>347</td>
<td>1</td>
<td>9</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SEND_EMAIL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Cumulative</strong></td>
<td>380</td>
<td>26</td>
<td>57</td>
<td>497</td>
<td>0.90</td>
<td>0.95</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Table 5.5: using the AIgen5_zeroshot prompt designs

In Table 5.6, the experimental results derived from the 'AI_generated_13_zeroshot' prompt design applied to GPT-4 LLMs are showcased. Across the 862 tests across various API functions, the observed results are: TP rate of 59.86%, TN rate of 31.32%, FP rate of 6.61%, and an FN rate of 2.20%. The cumulative analysis highlights a commendable average F1 score of 0.93, elucidating the consistent performance and reliability of APIs under this specific prompt structure.

<table>
<thead>
<tr>
<th>APIs</th>
<th>TN</th>
<th>FN</th>
<th>FP</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERT_TO_SPEECH</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>CREATE_CALENDAR</td>
<td>0</td>
<td>0</td>
<td>36</td>
<td>24</td>
<td>0.40</td>
<td>1.00</td>
<td>0.57</td>
</tr>
<tr>
<td>CREATE_REMINDER</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GENERATE_IMAGE</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>57</td>
<td>1.00</td>
<td>0.97</td>
<td>0.98</td>
</tr>
<tr>
<td>GET_CALENDAR</td>
<td>15</td>
<td>7</td>
<td>3</td>
<td>35</td>
<td>0.92</td>
<td>0.83</td>
<td>0.88</td>
</tr>
<tr>
<td>GET_CURRENT_DATE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_CURRENT_STOCK</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>59</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_NEWS</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>40</td>
<td>1.00</td>
<td>0.80</td>
<td>0.89</td>
</tr>
<tr>
<td>GET_WEATHER</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>61</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>NOT_AVAILABLE</td>
<td>242</td>
<td>1</td>
<td>18</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SEND_EMAIL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Cumulative</strong></td>
<td>270</td>
<td>19</td>
<td>57</td>
<td>516</td>
<td>0.90</td>
<td>0.96</td>
<td>0.93</td>
</tr>
</tbody>
</table>

Table 5.6: using the AIgen13_zeroshot prompt designs
Table 5.7 summarizes the performance of the ‘AI_generated_2_zeroshot’ prompt design on GPT-4 Large Language Models executed across several API functionalities. A total of 960 assessments were conducted across multiple API tasks, yielding a True Positive (TP) rate of 55.0%, a True Negative (TN) rate of 31.88%, a False Positive (FP) rate of 9.38%, and a False Negative (FN) rate of 3.75%. The cumulative F1 score is reported at 0.89, encapsulating the overall efficacy of the model in accurately identifying different API functionalities.

<table>
<thead>
<tr>
<th>APIs</th>
<th>TN</th>
<th>FN</th>
<th>FP</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERT_TO_SPEECH</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>57</td>
<td>0.95</td>
<td>1.00</td>
<td>0.97</td>
</tr>
<tr>
<td>CREATECALENDAR</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>30</td>
<td>0.50</td>
<td>1.00</td>
<td>0.67</td>
</tr>
<tr>
<td>CREATE_REMINDER</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>52</td>
<td>0.87</td>
<td>1.00</td>
<td>0.93</td>
</tr>
<tr>
<td>GENERATE_IMAGE</td>
<td>0</td>
<td>26</td>
<td>0</td>
<td>34</td>
<td>1.00</td>
<td>0.57</td>
<td>0.72</td>
</tr>
<tr>
<td>GETCALENDAR</td>
<td>6</td>
<td>8</td>
<td>40</td>
<td>6</td>
<td>0.87</td>
<td>0.83</td>
<td>0.85</td>
</tr>
<tr>
<td>GET_CURRENT_DATE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_CURRENT_STOCK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_NEWS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GET_WEATHER</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>57</td>
<td>0.93</td>
<td>0.98</td>
<td>0.96</td>
</tr>
<tr>
<td>NOTAVAILABLE</td>
<td>308</td>
<td>1</td>
<td>45</td>
<td>3</td>
<td>0.06</td>
<td>0.75</td>
<td>0.12</td>
</tr>
<tr>
<td>SEND_EMAIL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Cumulative</td>
<td>315</td>
<td>36</td>
<td>96</td>
<td>513</td>
<td>0.84</td>
<td>0.93</td>
<td>0.89</td>
</tr>
<tr>
<td>Cumulative</td>
<td>306</td>
<td>36</td>
<td>90</td>
<td>528</td>
<td>0.85</td>
<td>0.94</td>
<td>0.89</td>
</tr>
</tbody>
</table>

Table 5.7: using the AIgen2_zeroshot prompt designs

For detailed documentation, code, and results from our experiment, including dataset, calculations, analysis, and plots, visit our GitHub repository [44].
5.3 Testing Large Language Models

We selected the three best-performing prompts from GTP-4 experiences to test the different LLMs and re-run the same test use cases with the models LLama2 from Meta and Palm from Google.

Table 5.8 details outcomes from our experiment utilizing the AIgenerated13_zeroshot prompt design on the performance of Palm LLMs in discerning different API commands. The table indicates that a cumulative 956 tests were executed across various API interfaces. Cumulatively, the performance reveals a TP rate of 3.98%, a TN rate of 35.77%, an FP rate of 35.88%, and an FN rate of 24.37%. Summatively, the APIs produced an average $F_1$ score of 0.12, as represented in the data.

<table>
<thead>
<tr>
<th>APIs</th>
<th>TN</th>
<th>FN</th>
<th>FP</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERT_TO_SPEECH</td>
<td>0</td>
<td>45</td>
<td>11</td>
<td>4</td>
<td>0.27</td>
<td>0.08</td>
<td>0.12</td>
</tr>
<tr>
<td>CREATECALENDAR</td>
<td>0</td>
<td>3</td>
<td>49</td>
<td>8</td>
<td>0.14</td>
<td>0.73</td>
<td>0.24</td>
</tr>
<tr>
<td>CREATE_REMINDER</td>
<td>0</td>
<td>18</td>
<td>38</td>
<td>3</td>
<td>0.07</td>
<td>0.14</td>
<td>0.10</td>
</tr>
<tr>
<td>GENERATE_IMAGE</td>
<td>0</td>
<td>52</td>
<td>6</td>
<td>2</td>
<td>0.25</td>
<td>0.04</td>
<td>0.06</td>
</tr>
<tr>
<td>GETCALENDAR</td>
<td>0</td>
<td>53</td>
<td>6</td>
<td>1</td>
<td>0.14</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>GET_CURRENT_DATE</td>
<td>0</td>
<td>1</td>
<td>59</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GET_CURRENT_STOCK</td>
<td>0</td>
<td>5</td>
<td>55</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GET_EMAIL</td>
<td>0</td>
<td>30</td>
<td>30</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GET_WEATHER</td>
<td>0</td>
<td>3</td>
<td>54</td>
<td>1</td>
<td>0.02</td>
<td>0.25</td>
<td>0.03</td>
</tr>
<tr>
<td>NOT_AVAILABLE</td>
<td>342</td>
<td>3</td>
<td>9</td>
<td>5</td>
<td>0.36</td>
<td>0.62</td>
<td>0.45</td>
</tr>
<tr>
<td>SEND_EMAIL</td>
<td>0</td>
<td>20</td>
<td>26</td>
<td>14</td>
<td>0.35</td>
<td>0.41</td>
<td>0.38</td>
</tr>
<tr>
<td>Cumulative</td>
<td>342</td>
<td>233</td>
<td>343</td>
<td>38</td>
<td>0.10</td>
<td>0.14</td>
<td>0.12</td>
</tr>
</tbody>
</table>

Table 5.8: Palm model using AIgen13_zeroshot prompt designs

Table 5.9 delineates the performance evaluation of the Palm LLM leveraging the ‘AI_generated_5_zeroshot’ prompt design. Out of a cumulative 960 tests executed, the model manifested a TP rate of 38.75%, a TN rate of 34.17%, an FP rate of 14.17%, and an FN rate of 12.92%. Synthesizing the metrics of all tested APIs, the results engendered an average $F_1$ score of 0.74. This indicates the model’s proficiency and areas of potential refinement.

Table 5.9: Palm model using AIgen13_zeroshot prompt designs
Table 5.9: Palm model using Algen5_zeroshot prompt designs

Table 5.10 delineates the results obtained when employing the Palm LLM with the 'Sparksofagi' prompt design in discerning varied API commands. The experiment encompassed 958 tests in total. Cumulatively, the true positive (TP) rate stood at 2.5%, the true negative (TN) rate was 23.9%, false positives (FP) registered at 62.5%, and false negatives (FN) were 11.1%. Taking into account the performance of all the APIs, the $F_1$ score was calculated to be 0.06.
In Table 5.11, provides an analysis of the 'AI_generated_13_zeroshot' prompt design on Llama2 LLMs, to ascertain the capability of correctly identifying distinct API commands. The test comprises a total of 321 trials spanning multiple API functionalities. The aggregated results indicate a TP percentage of 6.85%, a TN percentage of 33.95%, an FP percentage of 50.47%, and an FN percentage of 8.72%. The cumulative performance across all APIs, as represented by the $F_1$ score, stands at 0.19.

<table>
<thead>
<tr>
<th>APIs</th>
<th>TN</th>
<th>FN</th>
<th>FP</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERT_TO_SPEECH</td>
<td>0</td>
<td>5</td>
<td>14</td>
<td>1</td>
<td>0.07</td>
<td>0.17</td>
<td>0.10</td>
</tr>
<tr>
<td>CREATE CALENDAR</td>
<td>0</td>
<td>1</td>
<td>19</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>CREATE_REMINDER</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GENERATE_IMAGE</td>
<td>0</td>
<td>4</td>
<td>16</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GET_CALENDAR</td>
<td>0</td>
<td>2</td>
<td>18</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GET_CURRENT_DATE</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>13</td>
<td>0.93</td>
<td>0.68</td>
<td>0.79</td>
</tr>
<tr>
<td>GET_CURRENT_STOCK</td>
<td>0</td>
<td>2</td>
<td>11</td>
<td>7</td>
<td>0.39</td>
<td>0.78</td>
<td>0.52</td>
</tr>
<tr>
<td>GET_NEWS</td>
<td>1</td>
<td>8</td>
<td>11</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GET_WEATHER</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>1</td>
<td>0.05</td>
<td>1.00</td>
<td>0.10</td>
</tr>
<tr>
<td>NOT_AVAILABLE</td>
<td>108</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SEND_EMAIL</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Cumulative</strong></td>
<td>109</td>
<td>28</td>
<td>162</td>
<td>22</td>
<td>0.12</td>
<td>0.44</td>
<td>0.19</td>
</tr>
</tbody>
</table>

Table 5.11: Llama2 model using AIgen13_zeroshot prompt designs

Table 5.12 presents the experiment’s results using the ’Sparksofagi’ prompt design to evaluate Llama2 LLMs in identifying different API commands. The table outlines the performance metrics, including True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) for each API, along with the cumulative scores. In total, 640 tests were conducted across multiple API functions. The cumulative results display a TP rate of 2.19%, a TN rate of 26.56%, an FP rate of 40.00%, and an FN rate of 31.25%. The average $F_1$ score for all APIs is 0.06.

<table>
<thead>
<tr>
<th>APIs</th>
<th>TN</th>
<th>FN</th>
<th>FP</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERT_TO_SPEECH</td>
<td>0</td>
<td>24</td>
<td>16</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>CREATE CALENDAR</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>CREATE_REMINDER</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GENERATE_IMAGE</td>
<td>0</td>
<td>30</td>
<td>10</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GET_CALENDAR</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GET_CURRENT_DATE</td>
<td>0</td>
<td>22</td>
<td>16</td>
<td>2</td>
<td>0.11</td>
<td>0.08</td>
<td>0.10</td>
</tr>
<tr>
<td>GET_CURRENT_STOCK</td>
<td>0</td>
<td>24</td>
<td>16</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GET_NEWS</td>
<td>0</td>
<td>38</td>
<td>2</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GET_WEATHER</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>NOT_AVAILABLE</td>
<td>170</td>
<td>2</td>
<td>56</td>
<td>12</td>
<td>0.18</td>
<td>0.86</td>
<td>0.29</td>
</tr>
<tr>
<td>SEND_EMAIL</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Cumulative</strong></td>
<td>170</td>
<td>200</td>
<td>256</td>
<td>14</td>
<td>0.05</td>
<td>0.07</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Table 5.12: Llama2 model using Sparksofagi prompt designs

Table 5.13 represents the performance of Llama2 in recognizing various APIs using the 'AI_generated_5_zeroshot' prompt design. The experiment involved a total of 225
tests across different API functions. The cumulative results indicate a TP rate of 2.22%, a TN rate of 4.00%, an FP rate of 84.89%, and an FN rate of 8.44%. The F1-Score, which measures the model’s accuracy considering precision and recall, yielded an average of 0.05 across all APIs. These results indicate the model’s effectiveness in API recognition under the experiment’s conditions.

<table>
<thead>
<tr>
<th>APIs</th>
<th>TN</th>
<th>FN</th>
<th>FP</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERT_TO_SPEECH</td>
<td>0</td>
<td>15</td>
<td>3</td>
<td>2</td>
<td>0.40</td>
<td>0.12</td>
<td>0.18</td>
</tr>
<tr>
<td>CREATECALENDAR</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>CREATEREMINDER</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GENERATE_IMAGE</td>
<td>0</td>
<td>1</td>
<td>19</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GETCALENDAR</td>
<td>2</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GETCURRENTDATE</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GETCURRENTSTOCK</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GETNEWS</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GETWEATHER</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>NOTAVAILABLE</td>
<td>7</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SEND_EMAIL</td>
<td>0</td>
<td>2</td>
<td>14</td>
<td>3</td>
<td>0.18</td>
<td>0.60</td>
<td>0.27</td>
</tr>
<tr>
<td><strong>Cumulative</strong></td>
<td>9</td>
<td>19</td>
<td>191</td>
<td>5</td>
<td>0.03</td>
<td>0.21</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 5.13: Llama2 model using Algen5_zeroshot prompt designs

For more information on our experiment results, including dataset, calculations, analysis, and plots, please visit our GitHub repository [44]. The repository contains detailed documentation and code to aid in understanding and replicating our work.
6 Analysis

From the results gathered to compare the prompt system designs with the GPT-4 model, we summarize all the data in the following Figure 6.1 (confusion matrix): Model performance: F1 score matrix for different APIs using various prompt designs.

The prompts AI generated prompt 13 (zero shot) got an F-score of 0.96, AI generated prompt 5 (zero shot) got 0.94, Sparks of AGI 0.93, AI generated prompt 2 (one shot) 0.92, AutoGPT based had 0.9 and AI generated prompt 5 (one shot) had 0.89.

From these results, we can see a consistent performance from the GPT-4 LLM model in different prompt designs, with the AI-generated prompt 13 obtaining the highest F-score of 0.96. This may indicate its superior ability to detect user intent and structuring final outputs effectively, as the F-score reflects the overall performance of a model in terms of precision and recall.

Similarly, the AI-generated prompt 5 in zero-shot also performed well with a 0.94 F score, and Spark of AGI also showcases a competitive performance with a score of 0.93.
These may point to their effective prompt design which was conducive to tool selection and data extraction.

On the other hand, AI-generated prompt 2 and 5 in one-shot settings, as well as the AutoGPT-based prompts had slightly lower F-scores, at 0.92, 0.89, and 0.9 respectively.

Figure 6.2 presents a heatmap comparing the F-score performance of three LLMs: GPT-4, Palm, and Llama-2. The comparison is conducted across the top three prompt designs and various APIs.

Figure 6.2: A Heatmap of F-scores for GPT-4, Palm, and Llama-2 using the best three prompts

To note as well, some TP when testing the negative class. There were some extraordinary situations in which the model managed to solve the problem in unexpected ways, by using follow-up questions to solve the problem even without having a specific API for that or by using another API to deliver the answer in a way more natural to the user. For example, in experiment 280, using the GTP-4 model with Sparksofagi prompt and when asked to show the user stock portfolio and even not having a specific API for that task, it asked the user for the list of companies that compose his portfolio in a follow-up question and then called the ‘getCurrentStockPrice’ API multiple times for each company, see figure 7. Another example was when creating a 5-minute breathing exercise that was not supposed to be able to solve. It generated the exercise and then used the text-to-speech API command to say it to the user, which indicates some understanding that usually breathing exercises are spoken to the user to guide him during the exercise, see figure 8.
EXPERIMENT 280 ====
SCORE: 3
SYSTEM PROMPT
You have the ability to use API commands to answer an user request. To do so, you would need access to the following API functions:

1. 'sendEmail(recipient, subject, message)' - this function sends an e-mail with a subject, recipient e-mail and text message.
2. 'getEvents(date)' - this function gets the calendar events from the user for a specific date.
3. 'createEvent(name, date, time)' - this function creates an event with a name, date and time.
4. 'createReminder(name, date, time)' - this function creates a reminder with a name, date and time.
5. 'getCurrentDateAndTime()' - this function returns the current date and time.
6. 'getWeather(city, country, date)' - this function returns the weather for a date, city and country.
7. 'getNews(country, topic)' - this function returns the top headlines for a country in a specific topic.
8. 'convertTextToSpeech(text, languageCode, gender)' - this function converts text-to-speech with text as input, destination language and gender.
9. 'getCurrentStockPrice(ticker)' - this function returns the current price for a publicly traded company.
10. 'generateImage(prompt)' - this function returns a generated image related to the inserted prompt.

You either call one of the APIs or call respond(message) to respond to the user.

---

USER: How’s my portfolio doing?
AI: To answer this question, I would need to know the tickers of the stocks in your portfolio. Could you please provide them?

USER: MSFT, GOOGL, AMD
AI: Sure, let's check the current stock prices for these companies.

API command: getCurrentStockPrice('MSFT')
API command: getCurrentStockPrice('GOOGL')
API command: getCurrentStockPrice('AMD')

Listing 7: Experiment 280 with GPT-4 model and Sparksofagi prompt
AI: Give me a 5-minute breathing exercise.
The user is asking for a 5-minute breathing exercise. I can provide a simple exercise that involves inhaling, holding the breath, and exhaling, each for a certain amount of time. This exercise can be repeated until the 5 minutes are up.

Breathing exercises can help reduce stress and promote relaxation. A common technique is the 4-7-8 method, where you inhale for 4 seconds, hold the breath for 7 seconds, and exhale for 8 seconds. This cycle can be repeated until the 5 minutes are up.

This is a simple and effective breathing exercise, but it might not be suitable for everyone. Some people might find it difficult to hold their breath for 7 seconds, or to exhale for 8 seconds. It's important to remind the user to do what feels comfortable for them.

Sure, here's a simple 5-minute breathing exercise for you. It's called the 4-7-8 method. Start by exhaling completely through your mouth. Then, inhale quietly through your nose for a count of 4. Hold your breath for a count of 7. Exhale completely through your mouth for a count of 8. This completes one breath. Now inhale again, and repeat the cycle three more times for a total of four breaths. Continue this exercise until the 5 minutes are up. Remember, it's important to do what feels comfortable for you. Don't force anything, and take breaks if you need to.

Sure, here's a simple 5-minute breathing exercise for you. It's called the 4-7-8 method. Start by exhaling completely through your mouth. Then, inhale quietly through your nose for a count of 4. Hold your breath for a count of 7. Exhale completely through your mouth for a count of 8. This completes one breath. Now inhale again, and repeat the cycle three more times for a total of four breaths. Continue this exercise until the 5 minutes are up. Remember, it’s important to do what feels comfortable for you. Don’t force anything, and take breaks if you need to.
comfortable for you. Don’t force anything, and take breaks if you need to.”

Listing 8: Experiment 260 with GPT-4 model and AutoGPT prompt

Regarding zero-shot vs one-shot prompts, the zero-shot had $F_1$ scores of 0.931 (AI_generated_13_zeroshot), 0.923 (AI_generated_5_zeroshot), 0.910 (sparksofagi), 0.886 (AI_generated_2_zeroshot) and 0.857 (autogpt_zeroshot), with an average of 0.9014. The one-shot prompts had $F_1$ scores of 0.905 (AI_generated_2_oneshot) and 0.881 (AI_generated_5_oneshot), with an average of 0.893.

In tables 5.8 (Palm model using AI_generated_13_zeroshot), 5.10 (Palm with Sparksofagi), 5.12 (Lamma2 with Sparksofagi), 5.13 (Lamma2 with AI_generated_5_zeroshot) when testing the positive class, they had very few TP and a considerable amount of FN which indicates a bad understanding of the task. The FP when testing the positive class, derives from the strict classification of TP. When the model misses an input argument, fails to print the API command in the right structure, calls a non-existent API, or prints a result without calling any API command we classify it as FP.
7 Discussion

Our results demonstrated that different prompt designs significantly impact the performance of Large Language Models (LLMs) when interacting with API tools. Our core research question aimed to identify which prompt designs were most effective in facilitating this interaction—our results clearly showed that the AI-generated prompt 13 (zero-shot) yielded the highest effectiveness, reflected in its top F-score of 0.931. AI-generated prompt 5 (zero-shot) and Sparksofagi prompt followed closely with F-score of 0.923 and 0.910 accordingly.

Our study contributed to the body of knowledge by objectively comparing several prompt designs with different API interactions using GPT-4.

Although our results suggested a high level of consistency in the GPT-4 model’s performance across different prompt designs and data structures, it is important to consider these findings within the study’s limitations. Given the rapidly advancing field of LLMs, it is likely that future studies involving more recent or advanced models may yield different results.

We understand that for the GPT-4 model, the zero-shot system prompts had better results than the one-shot prompts. This might result from the fact that the example given in the one-shot prompts in some cases are from a different subject than what the user is asking, and as such it increases the entropy and noise within the system prompt, making it more difficult for the LLM to understand the intention of the user.

Regarding our third research question on the reliability of LLMs, we found that there were considerable variations in behaviour between LLMs. A good system prompt for one LLM can perform badly with another LLM. This situation might be related to the training data for each large language model and feedback given during training. The LLMs that had a low \( F_1 \) might not have been trained with data similar to the system prompts that we used, which in turn results in a bad performance for this specific task. This suggests that the search for the best system prompt design should be limited to the scope of one LLM.

Additionally, another way to improve the results could be by fine-tuning an open-source model (like Llama-2, for example) for this specific task or even specific APIs. This however requires much more human resources and it is not a plug-and-play solution.

Also, when selecting randomly the AI-generated prompts there was the risk that we might leave behind some system prompts that might have better results with other LLMs. This opens an opportunity to further test other LLMs with system prompts suggested in the appendix.

It is important to mention as well the costs of running our experiments. Using openAI API it cost us 105.87$ USD, and the inference endpoint from HuggingFace 54.60$ USD, with a total cost of 160.47$ USD. This has an impact on the number of times we run the experiments and we might have more significant data if we run the tests more times with more user prompts to test each test scenario.

As for future work, one potential avenue could be to explore the efficacy of the new GPT-4 model that is capable of executing tasks with functions defined by the developer. This model was launched on the 13th of July 2023 while we were writing this thesis and it has the potential to have better results than our findings.

Furthermore, the field of LLMs is developing rapidly, and as such, continual refinement and updating of this research would be useful—to ensure continued understanding of optimal prompt design and tool utilization amidst the evolution of LLMs. In that order, we have more suggestions for possible system prompts in the Appendix of this thesis that might be interesting to test using different LLMs.
Overall, this study represented a necessary and timely exploration of Large Language Models’ interaction with API tools. By better understanding and optimizing this interaction, the potential uses and benefits of LLMs can significantly grow, offering a compelling insight into the future of NLP and AI development.
The results from our thesis project shed light on the nuances of Large Language Models (LLMs) and their ability to interact with API tools. Our experiment encompassed a variety of prompt designs and compared their effectiveness in enabling these LLMs to work cohesively with different APIs.

All the system prompts tested with GPT-4 had a $F_1$ score higher than 0.85. This illustrates that LLMs can accurately interpret user intent, select the relevant API tool, acquire the necessary data, and structure coherent output. This specifically addresses the knowledge gap in comprehending how LLMs can be used to interact with APIs.

The AI-generated prompts remarkably stood out compared with prompts used in the open-source and scientific community, with prompt 13 in the zero-shot context scoring the highest F-score of 0.931.

Zero-shot prompts show a better performance than one-shot prompts with GPT-4 model. One possible reason is that the examples given could compete with the user prompt in some situations where the subject of the example is different than the subject of the user, for example, when the example shows how to use the 'sendEmail' API command but the user asks for a company stock price. This opens the opportunity for further investigation to understand if this is a behaviour specific to this model or a general behaviour with chat-based models. Few-shot prompts should be also tested in future investigations to see if they influence the results.

Upon reflection, while our project yielded significant results, the investigation of a broader range of LLMs and API tools could have potentially revealed even richer insights. We concentrated mainly on GPT-4, Palm and LLama 2 models. From this subset of LLM, it seems that the results are model-specific whereas a system prompt that performs well in GPT-4 does not translate into a good performance in the other LLMs tested. This opens the question of which system prompts will perform well for this task in other LLMs or if it is an intrinsic problem with the other LLMs that are not able to fulfil this task.

If we were to continue our project, we would like to further analyze more LLMs and system prompts and delve deeper into other factors, such as the specific language or format of prompts that may influence their effectiveness.

The valuable insight of this thesis is not only applicable to academia but is also extensive to industry and society. Many businesses use chatbots for customer service and often require interaction with several API tools. Our findings can enhance and expedite the development process in such settings, leading to more robust and reliable chatbot solutions while saving time and resources.

Certain observations from our study indicate promising avenues for future research, such as which other models could be used to interact with API tools, or if its possible to find a system prompt that will perform better in each LLM; Understanding why certain AI-generated prompts excel and others fail could be intriguing to follow up on. With our work, we also leave a list of possible interesting system prompts in the appendix (AI generated by GPT-4) that could be used in the future. We hope our findings spark further investigations in this fascinating field.
References


A  Plain text API list with input arguments

A system prompt inspired by the paper "Sparks of Artificial General Intelligence: Early Experiments with GPT-4" where the API tools and input arguments are defined in a list in plain text. For example:

<table>
<thead>
<tr>
<th>Function</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEND_EMAIL</td>
<td>email_recipient, subject, message</td>
</tr>
<tr>
<td>CREATECALENDAR_EVENT</td>
<td>date, time, event_name</td>
</tr>
<tr>
<td>GETCALENDAREVENTS</td>
<td>date</td>
</tr>
<tr>
<td>CREATE_REMINDER</td>
<td>date, time, text</td>
</tr>
<tr>
<td>GET_CURRENT_DATE_AND_TIME</td>
<td></td>
</tr>
<tr>
<td>GOOGLE_SEARCH</td>
<td>search_query</td>
</tr>
</tbody>
</table>

Listing 9: Plain text API list with input arguments

B  AutoGPT system prompt

A system prompt inspired by AutoGPT, with command label, name and arguments for each tool.

Tool format (numbered list):

```
COMMAND_LABEL: COMMAND_NAME, args: ARG_KEY: ARG_VALUE
```

Response format:

```
"command": {"name": "command name", "args": {"arg name": "value"}}
```
Complete example to use the API tools we have selected for this study:

Commands:
1. send an email: send_email, args: email_recipient: <email_address>, subject: <string>, message: <string>
2. create an event in user calendar: create_calendar_event, args: date: <YYYY-MM-DD>, time: <HH:mm>, event_name: <string>
3. get events for a specific date: get_calendar_events, args: date: <YYYY-MM-DD>
4. create a reminder for the user: create_reminder, args: date: <YYYY-MM-DD>, time: <HH:mm>, text: <string>
5. get current date and time: get_current_date_and_time, args: None
6. search the web for a specific query: google_search, args: search_query: <string>

You should only respond in JSON format as described below

Format:
```
{
   "thoughts": {
      "text": "thought",
      "reasoning": "reasoning",
      "plan": "- short bulleted\n- list that conveys\n- long-term plan",
      "criticism": "constructive self-criticism",
      "speak": "thoughts summary to say to user",
   },
   "command": {"name": "command name", "args": ["arg name": "value"]},
}
```

Ensure the response can be parsed by Python json.loads

Listing 10: System prompt inspired by AutoGPT

System prompt inspired by OpenAI plugins:
Tool format:

- API name in natural language
- API description in natural language
- Use of OpenAPI standard to instruct the available commands and input arguments

To use OpenAPI prompts we will divide the prompts in two:

1. A prompt with the list of API tools with their name and description, where we ask the LLM to select the API tool to answer the user request
2. Another prompt specific for integrating the OpenAPI specs into the prompt.

---

2 https://github.com/Significant-Gravitas/Auto-GPT/blob/ca5abff93fb5e170788415f23d2bf098a26dc4f6/autogpt/prompts/generator.py
An example of the second prompt is as follows:

```typescript
/* API for fetching Klarna product information */
type productsUsingGET = (_: {
  q: string,
  size?: number,
  min_price?: number,
  max_price?: number,
}) => any;
```

**USER_INSTRUCTIONS:** "whats the most expensive shirt?"

Your arguments must be plain json provided in a markdown block:

**ARGS:**
```
{valid json conforming to API_SCHEMA}
```

Example
-----

**ARGS:**
```
{"foo": "bar", "baz": {"qux": "quux"}}
```
The block must be no more than 1 line long, and all arguments must be valid JSON. All string arguments must be wrapped in double quotes.
You MUST strictly comply to the types indicated by the provided schema, including all required args.

If you don’t have sufficient information to call the function due to things like requiring specific uuid’s, you can reply with the following message:

Message: ```text
Concise response requesting the additional information that would make calling the function successful.
```
13. ‘API_JOKE(category)’
14. ‘API_TRAFFIC_CONDITIONS(location, time)’
15. ‘API_BOOK_RECOMMENDATION(genre, author)’
16. ‘API_HOROSCOPE(sign, date)’
17. ‘API_LOCAL_EVENTS(location, date, category)’
18. ‘API_REESTAURANT_RECOMMENDATION(location, cuisine, price_range)’
19. ‘API_QUOTE(category)’
20. ‘API_GEOCODE(address)’

For example, if the user wants to know the weather for a specific city, ask for the city and country code, then provide the command like this: ‘API_WEATHER(New York, US)’.

Listing 12: GPT-4 generated prompt 1

D Prompt 2 generated by GPT-4

Welcome to GPT-4! You can access a wide range of APIs to help answer your questions. Here are 20 external API commands you can use:

1. ‘API:Weather:Location=<city>,<country>’
2. ‘API:News:Topic=<topic>’
3. ‘API:Stock:Symbol=<stock_symbol>’
4. ‘API:Currency:From=<from_currency>,To=<to_currency>’
5. ‘API:Translation:Text=<text>,From=<from_lang>,To=<to_lang>’
6. ‘API:Movies:Title=<movie_title>’
7. ‘API:Books:Title=<book_title>’
8. ‘API:Restaurants:Location=<city>,<country>,Food=<food_type>’
9. ‘API:Distance:From=<from_location>,To=<to_location>’
10. ‘API:Define:Word=<word>’
11. ‘API:Recipe:Ingredients=<ingredient1>,<ingredient2>,...’
12. ‘API:Flight:From=<from_city>,To=<to_city>,Date=<date>’
13. ‘API:Sports:Team=<team_name>’
14. ‘API:Lyrics:Artist=<artist_name>,Song=<song_title>’
16. ‘API:Math:Equation=<equation>’
17. ‘API:Cryptocurrency:Symbol=<crypto_symbol>’
18. ‘API:Events:Location=<city>,<country>,Date=<date>’
19. ‘API:JobSearch:Position=<job_title>,Location=<city>,<country>’
20. ‘API:LanguageDetection:Text=<text>’
To use a specific tool, I need to gather all the required data. Please answer the questions below based on the API command you want to use. Once I have all the data, I will provide you with the selected command and its input arguments.

Listing 13: GPT-4 generated prompt 2

E Prompt 3 generated by GPT-4

Welcome to GPT-4! Here are 20 different external API commands that you can use to answer user requests. Please provide the necessary information when prompted.

1. weather_api(location, datetime) - Get weather information for a specific location and date/time.
2. stock_api(ticker_symbol) - Get stock information for a specific ticker symbol.
3. sports_api(sport, team) - Get information about a sports team or event.
4. news_api(topic, date) - Get news articles related to a specific topic and date.
5. language_api(text, target_lang) - Translate text into the target language.
6. restaurant_api(location, cuisine) - Find restaurants in a specific location and cuisine type.
7. movie_api(title, release_year) - Get information about a movie by title and release year.
9. flight_api(origin, destination, date) - Find flights between origin and destination on a specific date.
10. hotel_api(location, check_in, check_out) - Find hotels in a specific location between check-in and check-out dates.
11. currency_api(base_currency, target_currency, amount) - Convert an amount from one currency to another.
12. map_api(location, zoom_level) - Get a map image of a specific location at a zoom level.
13. recipe_api(ingredients, dish_type) - Find recipes based on ingredients and dish type.
14. job_api(position, location) - Find job listings for a specific position and location.
15. business_api(category, location) - Find local businesses in a specific category and location.
16. fact_api(subject) - Get a random fact about a specific subject.
17. definition_api(word) - Get the definition of a specific word.
18. synonym_api(word) - Get synonyms for a specific word.
19. product_api(product_name, brand) - Get information about a product by name and brand.
20. event_api(event_name, location, date) - Get information about an event by name, location, and date.

To use a specific tool, simply type the command number followed by the necessary information separated by commas. For example, to get the weather in New York City on December 25th, type "1, New York City, December 25th". GPT-4 will then ask you any additional questions needed to gather all the data for the specific tool. Once all the data has been gathered, GPT-4 will respond with the command selected and all its input arguments.

Listing 14: GPT-4 generated prompt 3

F Prompt 4 generated by GPT-4

GPT-4, you have access to the following external API commands to help answer user requests. To use a specific command, gather all necessary input arguments by asking the user questions, then provide the selected command and its input arguments in a single line.

1. ‘GET_WEATHER|location|date’
2. ‘FIND_RECIPES|ingredients|cuisineType’
3. ‘TRANSLATE_TEXT|text|targetLanguage’
4. ‘GET_STOCK_PRICE|symbol|date’
5. ‘FIND_FLIGHTS|origin|destination|departureDate|returnDate’
6. ‘GET_NEWS|topic|language|sortBy’
7. ‘CONVERT_CURRENCY|amount|fromCurrency|toCurrency’
8. ‘FIND_HOTELS|city|checkInDate|checkOutDate|rating’
9. ‘GET_MOVIE_INFO|title|year’
10. ‘GET_BOOK_INFO|title|author’
11. ‘GET_NEARBY_RESTAURANTS|location|cuisineType|radius’
12. ‘FIND_EVENTS|city|category|startDate|endDate’
13. ‘GET_TRAVEL_GUIDE|destination|language’
14. ‘FIND_JOBS|jobTitle|location|employmentType’
15. ‘GET_SPORTS_SCORES|team|date’
16. ‘GET_MUSIC_INFO|artist|songTitle’
17. ‘GET_TV_SHOW_INFO|title|season|episode’
18. ‘FIND_LOCAL_SERVICES|serviceType|location|rating’
19. ‘GET_HOROSCOPE|sign|date’
20. ‘GET_QUOTE|category’

To use a specific tool, carefully ask the user for all the necessary information. Once you have gathered all the
data, provide the selected command and its input arguments in a single line as shown in the examples above.

Listing 15: GPT-4 generated prompt 4

G Prompt 5 generated by GPT-4

System Prompt: Please provide information for the following external API commands that GPT-4 can use to answer user requests. Ensure that the input arguments, format, and type are clear and easily parsed by another program performing the API request. Each command should occupy only one line.

1. command=translate; text={text}; source_language={source_lang}; target_language={target_lang}
2. command=weather; city={city}; country_code={country_code}
3. command=currency_convert; from_currency={from_currency}; to_currency={to_currency}; amount={amount}
4. command=recipe_search; query={query}; cuisine={cuisine}; diet={diet}; type={type}
5. command=movie_recommendation; genre={genre}; year={year}; rating={rating}; language={language}
6. command=flight_search; origin={origin}; destination={destination}; departure_date={departure_date}; return_date={return_date}
7. command=stock_price; symbol={symbol}; interval={interval}
8. command=sports_scores; team={team}; league={league}; season={season}
9. command=music_search; artist={artist}; album={album}; song_title={song_title}; genre={genre}
10. command=book_search; title={title}; author={author}; genre={genre}; language={language}
11. command=image_search; query={query}; size={size}; color={color}; license={license}
12. command=define_word; word={word}; language={language}
13. command=synonyms; word={word}; language={language}
14. command=antonyms; word={word}; language={language}
15. command=distance_calculation; from_location={from_location}; to_location={to_location}; unit={unit}
16. command=url_shortener; long_url={long_url}
17. command=qr_code_generator; data={data}; size={size}; color={color}
18. command=geolocation; ip_address={ip_address}
19. command=trivia_question; category={category}; difficulty={difficulty}
20. command=news_search; query={query}; language={language}; sort_by={sort_by}
To use a specific tool, GPT-4 will ask you all the necessary questions to gather the required data. Once it has all the data, GPT-4 will provide you with the selected command and its input arguments based on the data you provided.

Listing 16: GPT-4 generated prompt 5

H  Prompt 6 generated by GPT-4

Please use the following 20 external API commands to answer user requests. Make sure to understand the input arguments with their format and type. These commands can be easily parsed by another program that will handle the API request. Each command occupies only one line.

1. `api_command: weather.get_current(city: string, units: string)`

2. `api_command: currency.convert(from_currency: string, to_currency: string, amount: float)`

3. `api_command: news.get_top_headlines(country: string, category: string)`

4. `api_command: translation.translate_text(text: string, source_language: string, target_language: string)`

5. `api_command: wikipedia.page_summary(title: string)`

6. `api_command: movie.search_movie(query: string, year: integer)`

7. `api_command: nutrition.get_food_info(food: string)`

8. `api_command: travel.get_flight_prices(origin: string, destination: string, departure_date: date, return_date: date)`

9. `api_command: sports.get_team_schedule(team: string, season: string)`

10. `api_command: dictionary.define_word(word: string)`

11. `api_command: finance.get_stock_price(stock_symbol: string)`
Based on the user’s intention, ask all the necessary questions to gather the required data to use a specific tool. Once you have all the data, provide the selected command and its input arguments using the data obtained from the user.

Listing 17: GPT-4 generated prompt 6

I Prompt 7 generated by GPT-4

System Prompt: Welcome to GPT-4! We offer 20 different external API commands to help you find the best answers to your requests. Please provide your request and follow further instructions to gather the necessary data. Our commands are:

1. api://weather/getWeather?location={LOCATION}&date={DATE}&units={UNITS}
2. api://news/getNews?category={CATEGORY}&country={COUNTRY}&limit={LIMIT}
3. api://translator/translate?text={TEXT}&source_language={SRC_LANG}&target_language={TGT_LANG}
Based on your request, GPT-4 will ask you relevant questions to gather necessary input data. Once all data is collected, GPT-4 will respond with the appropriate command and input arguments to help you.

Listing 18: GPT-4 generated prompt 7

J Prompt 8 generated by GPT-4

System Prompt: GPT-4, please assist users in gathering necessary data for the following 20 external API
commands. Ensure that you ask all relevant questions to collect the required input arguments. Once you have all the data, provide the selected command and its input arguments based on the user’s information.

1. get_weather_data | location (str) | date (YYYY-MM-DD) | time (HH:mm) | unit (str: optional = default_metric)
2. translate_text | text (str) | source_language (str) | target_language (str)
3. search_news_articles | query (str) | date_range (str: YYYY-MM-DD_to_YYYY-MM-DD) | language (str: optional = default_en) | sort_by (str: optional = default_relevancy)
4. find_stock_price | symbol (str) | date (YYYY-MM-DD) | time (HH:mm: optional)
5. book_flight | origin (str) | destination (str) | departure_date (YYYY-MM-DD) | return_date (YYYY-MM-DD: optional) | passengers (int) | class (str: optional = default_economy)
6. get_movie_showtimes | title (str) | location (str) | date (YYYY-MM-DD) | time (HH:mm: optional)
7. convert_currency | amount (float) | source_currency (str) | target_currency (str) | date (YYYY-MM-DD: optional)
8. find_restaurants | location (str) | cuisine (str) | price_range (str: optional) | rating (float: optional)
9. get_horoscope | sign (str) | date (YYYY-MM-DD)
10. calculate_route | origin (str) | destination (str) | mode (str) | avoid (str: optional)
11. search_song_lyrics | title (str) | artist (str: optional)
12. get_covid_stats | country (str) | date (YYYY-MM-DD: optional)
13. find_hotels | location (str) | check_in (YYYY-MM-DD) | check_out (YYYY-MM-DD) | rating (float: optional) | price_range (str: optional)
14. get_exchange_rates | base_currency (str) | target_currency (str: optional) | date (YYYY-MM-DD: optional)
15. search_jobs | position (str) | location (str) | industry (str: optional) | experience_level (str: optional)
16. get_recipe | dish (str) | cuisine (str: optional) | dietary_restrictions (str: optional)
17. find_events | location (str) | category (str) | date_range (str: YYYY-MM-DD_to_YYYY-MM-DD: optional)
18. get_crypto_price | symbol (str) | date (YYYY-MM-DD: optional) | time (HH:mm: optional)
19. search_books | title (str) | author (str: optional) | genre (str: optional) | publication_year (int: optional)
20. get_tv_schedule | channel (str) | date (YYYY-MM-DD) | time_range (str: HH:mm_to_HH:mm: optional)

Listing 19: GPT-4 generated prompt 8

In "GPT-4 generated prompt 7" the model included optional arguments and format instructions for some of the input arguments.
K Prompt 9 generated by GPT-4

1. `!getWeather(location: string, date: YYYY-MM-DD)`
2. `!translateText(text: string, targetLanguage: string)`
3. `!calculateRoute(startLocation: string, endLocation: string, mode: string)`
4. `!findRecipes(ingredients: string[], diet: string, exclude: string[])`
5. `!getNews(source: string, keyword: string, date: YYYY-MM-DD)`
6. `!currencyConverter(amount: number, fromCurrency: string, toCurrency: string)`
7. `!getStockPrice(symbol: string, date: YYYY-MM-DD)`
8. `!playSong(songName: string, artistName: string)`
9. `!movieRecommendation(genre: string, year: YYYY, minRating: number)`
10. `!getSportsScores(sport: string, date: YYYY-MM-DD)`
11. `!getFlightInfo(flightNumber: string, departureDate: YYYY-MM-DD)`
12. `!sendEmail(to: string, subject: string, body: string)`
13. `!setReminder(reminderText: string, dateTime: YYYY-MM-DD HH:mm)`
14. `!getHoroscope(sign: string, date: YYYY-MM-DD)`
15. `!searchImages(query: string, imageType: string, orientation: string)`
16. `!getJoke(jokeType: string)`
17. `!findNearby(location: string, placeType: string, radius: number)`
18. `!getCryptoPrice(cryptoSymbol: string, currency: string)`
19. `!getTriviaQuestion(category: string, difficulty: string)`
20. `!getDictionaryDefinition(word: string, partOfSpeech: string)`

To use a specific tool, I need to gather all the required data from you. Please answer the following questions:

1. What is the location?
2. What is the date? (Format: YYYY-MM-DD)

Once I have all the data, I will answer with the selected command and its input arguments, like this:

`!getWeather("New York", "2022-01-01")`

Listing 20: GPT-4 generated prompt 9

An interesting thing in the "GPT-4 generated prompt 9" is the use of the "!" character before each command, which might indicate a specific token or sequence of tokens that
were learned by the model. Also, it is interesting how all the commands are encapsulated in backticks.

L Prompt 10 generated by GPT-4

System Prompt: GPT-4, please gather the necessary information from the user and then provide the appropriate API command with its required input arguments. The available external API commands are:

1. GetCurrentWeather(city: str)
2. GetWeatherForecast(city: str, date: YYYY-MM-DD)
3. ConvertCurrency(from_currency: str, to_currency: str, amount: float)
5. GetNews(category: str, country: str)
7. GetMovieDetails(title: str, release_year: int)
9. CalculateRoute(start_location: str, end_location: str, mode: enum [driving, walking, bicycling, transit])
10. FindRecipe(ingredients: str[], diet: str, exclude: str[])
12. GetEventDetails(event_name: str, date: YYYY-MM-DD, location: str)
13. ImageRecognition(image_url: str)
14. GetSongLyrics(artist: str, song_title: str)
15. GetLocalTime(city: str, current_time: HH:mm)
16. GetCovidStats(location: str)
17. MeasureDistance(point1: str, point2: str, unit: enum [miles, kilometers])
18. AnalyzeSentiment(text: str)
19. GetHistoricalWeather(city: str, date: YYYY-MM-DD)
20. GetCryptoPrice(crypto_symbol: str)

Based on the user’s intention, ask all necessary questions to gather the required data for the selected API command. Once you have collected all the data, provide the chosen command with its input arguments.

In the prompt "GPT-4 generated prompt 10" there is the use of enums to list all the options possible for an input argument.

M Prompt 11 generated by GPT-4
1. ‘API_WEATHER_FORECAST: location=<city,country>&date=<YYYY-MM-DD>’
2. ‘API_STOCK_DATA: symbol=<stock_symbol>&interval=<1min,5min,15min,30min,60min,daily,weekly,monthly>&startdate=<YYYY-MM-DD>&enddate=<YYYY-MM-DD>’
3. ‘API_NEWS_HEADLINES: topic=<keywords>&language=<en,es,de,fr,it>&sortby=<relevance,publishedAt>’
4. ‘API_CURRENCY_EXCHANGE: base=<currency_code>&target=<currency_code>&date=<YYYY-MM-DD>’
5. ‘API_MOVIE_DETAILS: title=<movie_title>&year=<YYYY>’
6. ‘API_BOOK_SEARCH: title=<book_title>&author=<author_name>’
7. ‘API_FLIGHT_STATUS: flight_number=<flight_number>&date=<YYYY-MM-DD>’
8. ‘API_RESTAURANT_NEARBY: location=<city,country>&cuisine=<cuisine_type>&radius=<distance_in_km>’
9. ‘API_TRAFFIC_CONDITIONS: origin=<address_or_coordinates>&destination=<address_or_coordinates>&departure_time=<HH:mm>’
10. ‘API_SPORTS_SCORES: league=<league_name>&date=<YYYY-MM-DD>’
11. ‘API_EVENT_SEARCH: location=<city,country>&category=<event_category>&date=<YYYY-MM-DD>’
12. ‘API_TV_SHOW_DETAILS: title=<tv_show_title>&season=<season_number>’
13. ‘API_RECIPE_SEARCH: ingredients=<list_of_ingredients>&diet=<diet_type>&intolerances=<list_of_intolerances>’
14. ‘API_TRANSLATE_TEXT: text=<text_to_translate>&source_language=<language_code>&target_language=<language_code>’
15. ‘API_HOLIDAYS: country=<country_code>&year=<YYYY>’
16. ‘API_HOROSCOPE_SIGN: birthdate=<YYYY-MM-DD>’
17. ‘API_IMAGE_SEARCH: query=<keywords>&type=<photo,illustration,vector,all>&orientation=<horizontal,vertical,all>’
18. ‘API_JOB_SEARCH: keywords=<job_keywords>&location=<city,country>&employment_type=<full-time,part-time,contract,internship,temporary>’
19. ‘API_MUSIC_SEARCH: artist=<artist_name>&track=<track_title>’
20. ‘API_COVID19_STATS: country=<country_code>&date=<YYYY-MM-DD>’

Please provide the necessary information for the API command you want to use. Once you have provided all the required information, I will respond with the appropriate command and input arguments based on the data gathered.
API Commands:
10. [API-TV-SCHEDULE] GetTVSchedule(channel:str, date:YYYY-MM-DD, time:HH:mm)
17. [API-GEOLOCATION] Geolocation(ip_address:str)
18. [API-DISTANCE-CALCULATOR] CalculateDistance(from_location:str, to_location:str, unit:str)
19. [API-ASTRONOMY] GetAstronomyData(date:YYYY-MM-DD, location:str)
20. [API-NUMBER-TRIVIA] GetNumberTrivia(number:int)

Instructions:
1. Identify user’s intention and the required API command.
2. Ask questions to gather all necessary input arguments.
3. Format the input arguments according to the given format .
4. Respond with the selected API command and input arguments in one line.
Example:

User: How far is it from New York to Los Angeles?

GPT-4: What unit would you like the distance in? (Miles/ Kilometers)

User: Miles

GPT-4: [API-DISTANCE-CALculator] CalculateDistance( from_location: New York, to_location: Los Angeles, unit: Miles)

Listing 23: GPT-4 generated prompt 12

In "GPT-4 generated prompt 12" the model separates the commands from instructions and examples. Giving an example, also known as a few-shot prompt, is usually related to better results. [MISSING CITATION]

O Prompt 13 generated by GPT-4

System Prompt: GPT-4, please provide a list of 20 external API commands that you can use to help users with their requests. After detecting their intention, ask the necessary questions to collect the required data for the selected command. Once you have all the data, provide the command with the input arguments obtained from the user.

1. WeatherForecast: GetWeatherForecast(location, date, time =NULL); location: string; date: YYYY-MM-DD; time: HH:mm (optional)
2. CurrencyConversion: ConvertCurrency(amount, from_currency, to_currency); amount: float; from_currency: string; to_currency: string
3. FlightStatus: CheckFlightStatus(airline, flight_number, date=NULL); airline: string; flight_number: string; date : YYYY-MM-DD (optional)
4. RestaurantSearch: FindRestaurants(location, cuisine=NULL , price_range=NULL); location: string; cuisine: string (optional); price_range: string (optional)
5. MovieShowtimes: GetMovieShowtimes(location, movie=NULL, date=NULL, time=Null); location: string; movie: string (optional); date: YYYY-MM-DD (optional); time: HH:mm (optional)
6. NewsHeadlines: GetNewsHeadlines(category, country=NULL); category: string; country: string (optional)
7. StockPrice: GetStockPrice(stock_symbol); stock_symbol: string
8. SongLyrics: GetSongLyrics(artist, song_title); artist: string; song_title: string
9. SportsScores: GetSportsScores(league, team=NULL, date=NULL); league: string; team: string (optional); date: YYYY-MM-DD (optional)
10. BookRecommendations: GetBookRecommendations(genre, author=NULL); genre: string; author: string (optional)
11. RecipeSearch: FindRecipes(ingredient, dish=NULL); ingredient: string; dish: string (optional)
12. EventSearch: FindEvents(location, event_type=NULL, date=NULL); location: string; event_type: string (optional); date: YYYY-MM-DD (optional)
13. TriviaQuestions: GetTriviaQuestions(category, difficulty=NULL); category: string; difficulty: string (optional)
14. JobSearch: FindJobs(location, job_title=NULL, company=NULL); location: string; job_title: string (optional); company: string (optional)
15. ProductPrice: GetProductPrice(product_name, store=NULL); product_name: string; store: string (optional)
16. DictionaryLookup: DefineWord(word); word: string
17. LanguageTranslation: TranslateText(text, source_language, target_language); text: string; source_language: string; target_language: string
18. TVShowSchedule: GetTVShowSchedule(show_title, date=NULL); show_title: string; date: YYYY-MM-DD (optional)
19. LocalTime: GetLocalTime(city, country); city: string; country: string
20. Horoscope: GetHoroscope(sign, date=NULL); sign: string; date: YYYY-MM-DD (optional)

Example:
User Request: I want to know the weather forecast for London on December 5th.

GPT-4: What time would you like the weather forecast for? (Please provide the time in HH:mm format or type "Any" if you don’t have a specific time in mind.)

User: Any

GPT-4: GetWeatherForecast("London", "2022-12-05")

P Prompt 14 generated by GPT-4
1. GET_WEATHER | location : str | date : YYYY-MM-DD | time : HH:mm (optional)
2. CONVERT_CURRENCY | from : str | to : str | amount : float
3. TRANSLATE_TEXT | text : str | target_language : str
4. DEFINE_WORD | word : str
5. GET_STOCK_PRICE | symbol : str | date : YYYY-MM-DD (optional)
6. GET_RECIPE | ingredient : str | cuisine : str (optional) | dish_type : str (optional)
7. FIND_MOVIE | title : str | year : int (optional) | genre : str (optional)
8. GET_BOOK | title : str | author : str (optional) | genre : str (optional)
9. CALCULATE_DISTANCE | start_location : str | end_location : str
10. GET_NEWS | topic : str | date : YYYY-MM-DD (optional) | source : str (optional)
11. GET_EVENT | event_name : str | date : YYYY-MM-DD (optional) | location : str (optional)
12. GET_FLIGHT_INFO | flight_number : str | date : YYYY-MM-DD
13. FIND_RESTAURANT | name : str | location : str | cuisine : str (optional)
14. GET_SPORTS_SCORE | team_name : str | sport : str | date : YYYY-MM-DD (optional)
15. GET_HOROSCOPE | sign : str | date : YYYY-MM-DD
17. GET_PRODUCT_INFO | product_name : str | brand : str (optional)
18. GET_TRAFFIC_INFO | route : str | time : HH:mm (optional)
19. GET_JOB_LISTING | position : str | location : str | industry : str (optional)
20. GET_TV_SCHEDULE | channel : str | date : YYYY-MM-DD

[INSTRUCTIONS]
1. Identify the user’s intention.
2. Ask all necessary questions to gather the required information for the specific API command.
3. Once all information is obtained, provide the appropriate API command with its input arguments in the following format:

   API_COMMAND | input1 : value1 | input2 : value2 | ...

[EXAMPLE]
User: Find me a recipe with chicken.
GPT-4: Do you have a preferred cuisine?
User: Italian.
GPT-4: What type of dish are you looking for? (optional)
User: Main course.
GPT-4: GET_RECIPE | ingredient : chicken | cuisine : Italian | dish_type : main course
Prompt 15 generated by GPT-4

System Prompt: Provide the following 20 external API commands that GPT-4 can use to answer user requests. Ensure each command is on one line and includes input arguments, format, types, input options, and which are optional.

1. GET_WEATHER: location (string), date (YYYY-MM-DD, optional), time (HH:mm, optional)
2. CONVERT_CURRENCY: from_currency (string), to_currency (string), amount (float)
3. TRANSLATE_TEXT: text (string), target_language (string), source_language (string, optional)
4. CALCULATE_DISTANCE: start_location (string), end_location (string), unit (string, optional: "km", "mi", "m")
5. FIND_RESTAURANT: location (string), cuisine (string, optional), price_range (string, optional: "low", "medium", "high")
6. BOOK_FLIGHT: origin (string), destination (string), departure_date (YYYY-MM-DD), return_date (YYYY-MM-DD, optional), passengers (int)
7. FIND_HOTEL: location (string), check_in (YYYY-MM-DD), check_out (YYYY-MM-DD), stars (int, optional), price_range (string, optional: "low", "medium", "high")
8. SEARCH_MOVIE: title (string, optional), genre (string, optional), release_year (int, optional), director (string, optional)
9. GET_STOCK_PRICE: stock_symbol (string), date (YYYY-MM-DD, optional)
10. CALCULATE_TAX: income (float), country (string), state (string, optional)
11. GET_NEWS: topic (string), date_range (YYYY-MM-DD - YYYY-MM-DD, optional), source (string, optional)
12. FIND_EVENT: location (string), category (string, optional), date (YYYY-MM-DD, optional)
13. GET_RECIPE: ingredients (string list), cuisine (string, optional), dietary_restrictions (string, optional)
14. SEARCH_JOB: location (string), job_title (string), company (string, optional), experience_level (string, optional: "entry", "mid", "senior")
15. GET_HOROSCOPE: zodiac_sign (string), time_period (string, optional: "daily", "weekly", "monthly")
16. **FIND_NEARBY_PLACE** : location (string), type (string: "restaurant", "gas_station", "hospital", etc.), radius (int, optional, in meters)

17. **GET_TRAFFIC_UPDATE** : start_location (string), end_location (string), departure_time (HH:mm, optional)

18. **GET_QUOTE** : category (string, optional), author (string, optional)

19. **SEARCH_BOOK** : title (string, optional), author (string, optional), genre (string, optional), publication_year (int, optional)

20. **GET_SPORTS_SCORE** : team (string), league (string, optional), date (YYYY-MM-DD, optional)

**Instructions for GPT-4:** Based on the user’s intention, ask all necessary questions to gather the required data for the specific command. When you have all the data, answer with the selected command and its input arguments obtained from the user.

**Listing 26: GPT-4 generated prompt 15**

In this last prompt (GPT-4 generated prompt 15), there is a better description of the input arguments.

### Prompt 16 generated by GPT-4

**System Prompt:** Welcome to the GPT-4 API Assistant! Here is a list of 20 external API commands that GPT-4 can utilize to answer your requests. In order to proceed with a specific command, please follow the instructions provided by GPT-4 to gather the necessary input arguments. Each command is formatted as follows: ‘COMMAND_NAME(ARG1, ARG2, ...)’.

1. ‘GET_WEATHER(CITY, COUNTRY)’ – Retrieves the current weather conditions for the specified city and country.
2. ‘GET_STOCK_PRICE(STOCK_SYMBOL)’ – Retrieves the current stock price for the specified stock symbol.
3. ‘GET_NEWS(TOPIC, NUM_RESULTS)’ – Retrieves the latest news articles related to the specified topic, limited to the number of results requested.
4. ‘TRANSLATE_TEXT(TEXT, TARGET_LANGUAGE)’ – Translates the input text into the specified target language.
5. ‘GET_CURRENCY_EXCHANGE_RATE(FROM_CURRENCY, TO_CURRENCY)’ – Retrieves the current exchange rate between the specified currencies.
6. ‘GET_FLIGHTS(ORIGIN, DESTINATION, DEPARTURE_DATE)’ – Retrieves available flights from the origin to the destination on the specified departure date.
7. ‘GET_RESTAURANT_RECOMMENDATIONS(LOCATION, CUISINE, NUM_RESULTS)’ - Retrieves restaurant recommendations based on the specified location, cuisine, and number of results.
8. ‘GET_TRAFFIC_CONDITIONS(START_POINT, END_POINT)’ - Retrieves current traffic conditions between the specified start and end points.
9. ‘CALCULATE_DISTANCE(START_POINT, END_POINT, UNIT)’ - Calculates the distance between the specified start and end points in the specified unit (e.g., miles, kilometres).
10. ‘FIND_NEARBY_PLACES(LOCATION, PLACE_TYPE, RADIUS)’ - Finds nearby places of the specified type within the specified radius around the given location.
11. ‘GET_MOVIE_RECOMMENDATIONS(GENRE, NUM_RESULTS)’ - Retrieves movie recommendations based on the specified genre, limited to the number of results requested.
12. ‘GET_RECIPE(INGREDIENTS, NUM_RESULTS)’ - Retrieves recipes containing the specified ingredients, limited to the number of results requested.
13. ‘GET_BOOK_RECOMMENDATIONS(GENRE, NUM_RESULTS)’ - Retrieves book recommendations based on the specified genre, limited to the number of results requested.
14. ‘GET_MUSIC_RECOMMENDATIONS(GENRE, NUM_RESULTS)’ - Retrieves music recommendations based on the specified genre, limited to the number of results requested.
15. ‘GET_SPORTS_SCORES(TEAM, DATE)’ - Retrieves the sports scores for the specified team on the given date.
16. ‘GETCALENDAR_EVENTS(DATE, EVENT_TYPE)’ - Retrieves calendar events of the specified type on the specified date.
17. ‘GET_JOKE(CATEGORY)’ - Retrieves a joke from the specified category.
18. ‘GET_QUOTES(CATEGORY, NUM_RESULTS)’ - Retrieves quotes from the specified category, limited to the number of results requested.
19. ‘GET_FACTS(CATEGORY, NUM_RESULTS)’ - Retrieves facts from the specified category, limited to the number of results requested.
20. ‘CREATE_REMINDER(TITLE, DATE, TIME)’ - Creates a reminder with the specified title, date, and time.

Please provide the number of the command you’d like to use, and GPT-4 will guide you through gathering the necessary input arguments. After obtaining all the data, GPT-4 will provide the selected command along with its input arguments based on the information gathered from you.
Welcome to GPT-4! To help you answer user requests, you can use the following 20 external API commands. Make sure to gather all necessary input arguments from the user before executing a command. After gathering the required data from the user, provide the selected command along with its input arguments.

1. API_WEATHER_GET <city> <country_code> - Retrieves the current weather for a specified city and country code.

2. API_TRANSLATE_TEXT <text> <source_lang> <target_lang> - Translates the provided text from the source language to the target language.

3. API_NEWS_HEADLINES_GET <category> <country> - Fetches the latest news headlines for a specified category and country.

4. API_CURRENCY_CONVERT <amount> <from_currency> <to_currency> - Converts an amount of money from one currency to another.

5. API_STOCK_GET <symbol> - Retrieves the current stock price for a specified stock symbol.

6. API_MOVIE_RECOMMEND <genre> <year> - Suggests a movie based on the specified genre and release year.

7. API_RECIPE_SEARCH <ingredients> - Searches for a recipe that includes the provided list of ingredients.

8. API_EVENT_NEARBY_GET <location> <date> - Finds events happening near a specified location on a given date.

9. API_TRAVEL_DESTINATION_SUGGEST <interests> <budget> - Suggests a travel destination based on user interests and budget.

10. API_BOOK_RECOMMEND <genre> <author> - Recommends a book based on the specified genre and author.

11. API_SPORT_SCORE_GET <team1> <team2> <date> - Retrieves the score of a sports match between two teams on a given
12. API_RANDOM_FACT_GET <category> - Fetches a random fact from the specified category.

13. API_JOKE_GET <type> - Retrieves a joke of the specified type.

14. API_NEARBY_RESTAURANT_GET <location> <cuisine> - Finds nearby restaurants at a specified location with a given cuisine.

15. API_PRODUCT_SEARCH <keywords> <sort> - Searches for products based on keywords and sorts results by the selected criteria.

16. APICALENDAR_EVENT_CREATE <title> <description> <date> <time> - Creates a calendar event with the given title, description, date, and time.

17. API_PRICE_COMPARISON_GET <product_name> - Compares prices of a specified product across different retailers.

18. API_TRAFFIC_REPORT_GET <start_location> <end_location> - Provides a traffic report between the given start and end locations.

19. API_NEARBY_HOTEL_GET <location> <check_in> <check_out> - Finds nearby hotels at a specified location for the given check-in and check-out dates.

20. API_MUSIC_RECOMMEND <genre> <mood> - Recommends music based on the specified genre and mood.

Remember to ask all necessary questions to gather the required data from the user before selecting a command. When you have all the data, provide the selected command along with its input arguments, in the format shown above.

Listing 28: GPT-4 generated prompt 17

In "GPT-4 generated prompt 17", we have the input arguments separated by spaces and individualized by <>. Each command has a description so that the model can easily unequivocally identify each command. Additionally, this description could be used to provide more information about the input arguments, such as format, optional arguments, and input options for an argument.
You are a helpful bot that replies to a user. You can use a wide range of APIs to answer a user question. Here are the API commands you can use:

1. ‘API:SendEmail:Subject=<subject>,Message=<message>,Recipient=<recipient>’
2. ‘API:GetCalendarEvents:Date=<date>’
3. ‘API:CreateCalendarEvent:Name=<name>,Date=<date>,Time=<time>’
4. ‘API:CreateReminder:Name=<name>,Date=<date>,Time=<time>’
5. ‘API:GetCurrentDateTime’
6. ‘API:Weather:City=<city>,Country=<country>,Date=<date>’
7. ‘API:GetNews:Topic=<topic>,Country=<country>’
8. ‘API:ConvertTextToSpeech:Text=<text>,LanguageCode=<language_code>,Gender=<gender>’
9. ‘API:GetStockPrice:Stock=<tick_symbol>’
10. ‘API:GenerateImage:Text=<text>’

To use a specific command, you need to gather all the required data. Once you have all the data, answer with the command and its input arguments.

Example:
User Request: Create a reminder for tomorrow to buy milk in the morning

GPT-4: ‘API:CreateReminder:Name="Buy milk",Date="2023-04-24",Time="9:00"’

You have the following API commands to answer a user request. Ensure that the input arguments, format, and type are clear and easily parsed by another program performing the API request. Each command should occupy only one line.

1. command=SEND_EMAIL;message={message};subject={subject};recipient_email={recipient_email}
2. command=weather;city={city};country_code={country_code};date={date}
3. command=get_calendar_events;date={date}
4. command=create_calendar_event;name={name};date={date};time={time}
To use a specific tool, you should ask all necessary questions to gather the required data. Once you have all the data, you should respond with the selected command and input arguments based on the data received. If there is no API command available you should answer "I don’t know".

Example:
User Request: Create a reminder for tomorrow to buy milk in the morning

GPT-4: ‘command=create_reminder; name=Buy milk; date=2023-04-24; time=9:00’

Listing 30: Prompt 5 generated by GPT-4, one shot