This article will provide an overview of prompt engineering using LLM-based-tools like ChatGPT. By focusing on applications of prompt engineering with respect to python, the article will provide a practical guide for beginners in data science and python, who want to enhance their coding abilities and learn python for data science.

Why prompt engineering?

Prompt engineering has been on everyone's lips for quite a while now. Especially the success of OpenAI's ChatGPT and other Large Language Models further emphasizes the relevance of the topic. Prompt Engineering will help you to debug your code fast and efficiently, generate stunning reports quickly and plan your strategy in a more structured way. Mastering prompt engineering can lead to an unprecendented increase of productivity in your work with python. So, let us dive in.

Prerequisites

As this is a practical topics, readers should already have tested some LLM-based tools like ChatGPT in order to get a feeling for how these tools generate output based on your input. Furthermore, a basic understanding of Python and little experience in Jupyter Notebooks is advised aswell. Especially users should know, where to double-check the output of the used tools.

How prompt engineering works

Basic idea of large language models

The core of tools like ChatGPT are Large Language Models (LLMs). These models belong to the class of deep learning algorithms and are trained with an enormous quantity of data. LLMs can also be viewed as an outstanding example of generative artifical intelligence. Therefore LLMs are able to generate (new) content based on input (prompts) consisting of human language. Any attempt to explain generative AI and deep learning in only a few sentences are probably doomed to fail. Hence, we will focus on the basic concept regarding the way an LLM interprets and used the given input. (Amazon Web Services n.d a.) https://aws.amazon.com/what-is/large-language-model/

Very simply put, the LLM learns the distribution, relationships and concepts related to words based on huge amounts of text and unsupervised machine learning approaches. Hence, the modell is later able to predict the next words in a sentence or generate new text based on the input. (Lee 2024) https://blogs.nvidia.com/blog/what-are-large-language-models-used-for/

However, the way LLMs interpret input is not comparable to the way the human brains does. A crucial step in the input-interpretation and output-generation of an LLM is tokenization. Tokenization refers to the process of breaking down text input into small units which can then be represented as numbers. Those numerical interpretations are then fed into the model. The model evaluates the input and creates an answer, which is then detokenized and put out as human language. (Mistral AI n.d.) https://docs.mistral.ai/guides/tokenization/

One can already see that the way an LLM interprets input is very structured and based on the way the model was trained. This is why prompt engineering can be helpful to guarantee better and more relevant outcomes while working with LLM-based tools.

Explanation of selected prompting techniques

Since the user of an LLM-based tool is not able to see behind the curtain of the LLM-tool, one should be advised that prompt engineering is often an iterative process. Also prompt engineering is a relatively new idea, since it only became relevant when people started using tools like ChatGPT. Therefore prompt engineering is a highly dynamic field and best practices are still evolving. Nevertheless, in the last years some basic approaches towards prompt engineering emerged. (Amazon Web Services n.d. b) https://aws.amazon.com/what-is/prompt-engineering/?nc1=h_ls

Generally speaking, a user should always state what outcome, piece of information is most important for the task at hand. A clear and logical structure of the prompt is also helpful. Furthermore, it can be helpful to break down complex tasks into smaller and more simple prompts.

The simplest kind of prompt is the so-called **zero-shot-prompt**. It consists of a simple question or instruction with no additional context. More advanced but still a zero-shot prompt is the**assignment of a role**to the model. Assigning a role is often helpful, since the model will act as if it was a person with a specific professional background or opinion.

zero-shot-prompt:"Act as a professional travel-planner. Next week I will be spending a few days in Paris. Plan a 2-day trip for me."

Another technique involves the **usage of examples**. Such approaches are called **one-, few- and multi-shot prompts**. A one-shot prompt consists of one example, a few-shot prompt of a few examples and a multi-shot prompt of many examples. This technique is helpful if you want the model to replicate a pattern or when the output has to be structured in a very specific way, which is often the case if the output has to be scalable or replicable.

multishot-shot-prompt:"Your task is to categorize customer reviews into the categories positive, neutral or negative. Here are some examples:

1. Statement: "The product exceeded my expectations", Category: Positive
2. Statement: "The product was not as good as I expected", Category: Negative
3. Statement: "The product was okay, but not great", Category: Neutral

4. Statement: "The product was a total waste of money", Category: Negative

Please categorize this statement: "I sent the product back because it was broken".

In order to combine human intelligence with the power of LLMs, so-called **Chain-of-thought-prompts (CoT)** are another crucial technique. These prompts are iterative and demand the model to explain it's reasoning. (Google for Developers n.d.) https://developers.google.com/machine-learning/resources/prompt-eng

chain-of-thought-prompt: "What could be a successful strategy for a company to increase it's revenue given the current market situation and a restricted budget? Think step by step.""

Another concept called **scene defining** refers to the specification of a given context. This is helpful since it emphasized the most crucial components of the task at hand.

scene-defining-prompt: "Act as a CEO of an AI-startup, based in Lüneburg, Germany. You are currently planning the next steps for an expansion strategy towards the US-market. An inital market-analysis has shown that Americans are specifically interested in AI-based solutions for the healthcare sector. What would be your next steps?"

A very simple way of refining a prompt is using another or even the same LLM to generate, evaluate and improve your initial prompt. This idea is called **prompt-refinement**. This might be especially helpful if you are not experienced in working with the respective model.

prompt-refining-prompt: "Act as an expert in prompt engineering. I am using a image-generation tool called "Midjourney" and I want to generate a picture of a cat. Please generate a prompt for me."

(Zaghir et al. 2024)

[https://www.jmir.org/2024/1/e60501}

Table 1: Overview of selected prompting techniques

Table 1: Prompting Tech

Technique	Description
Zero-shot	A simple instruction or question given to the model without any additional context or e
Few-shot	A method where a few examples are provided within the prompt to help the model und
Multi-shot	Extends the few-shot approach by including multiple examples, creating a more compre
Chain-of-thought	Guides the model to provide step-by-step reasoning in its responses. This technique is p
Scene defining	Establishes a clear context or assigns a specific role to the model, such as acting as an e
Prompt refinement	Involves using iterative feedback to improve the quality of the prompt itself. The goal is

Examples for prompt engineering with Python

Some tools like ChatGPT or Anthropic's Claude provide an outstanding opportunity to learn writing code and conducting data science tasks. Generally, the explained prompt-engineering-techniques also apply to writing code in python. There is not one specific to find high-qualitiy prompts with regard to python. Nevertheless, the following workflow and the prompting examples will be helpful to get a feeling for prompt engineering using python. Assume that you are responsible for the analysis of a dataset. Since analysing data is a complex task, you will not be able to finish a task in one prompt.

Rather you would start by introducing the context and the topic of your task to your chosen LLM-tool.

Step one: Scene defining (e.g. setting the context, assigning a role, specifiying the task and the desired outcome, add your prefered way of working): "Act as a data scientist with at least 5 years of experience and a professional background in `[add topic with regard to your data set]`. Our task is to analyse a dataset containing information about `[add information about your dataset]`. The desired output is a report containing python-code, visualizations and a summary of the most important insights. Focus on the following aspects: `[add aspect you want to focus on]`. I will ask you specific questions and give you instructions on how to proceed and you will generate the python code for that particular step."

Now the LLM should be setup to assist you in the analysis of the dataset. It also can assist with idea generation for interpretation of the data, since we assigned it to be trained in the topic of specific dataset.

In the next step you could upload the dataset to the LLM-tool (always consider privacy issues before you do that). In that case the LLM knows all the names of variables and the structure of the data. Assume that you have specific idea for an analysis (e.g. you want to do a group comparison, or you want to build a generalized linear model etc.) Altenatively you could ask the LLM for specific idea which kind of analysis would be appropriate for the dataset.

Step two: Prompt refinement (e.g. generating a specific analysis idea): ""I want to build a generalized linear model to predict the variable `[add variable name]`. Please explain the workflow for such an analysis and explain the reasons behind each step."

The LLM-tool will then come up with an outline for your analysis. You can then ask for the python code for each step. Since it is often not obvious, why the LLM came up with a specific answer or code in this case, one should always advise the LLM to document and explain the code clearly. With respect to the techniques we defined earlier, it can be very helpful to use examples and a multi-shot prompt in this case. For instance, if you found a code-snippet in the internet you could use that as an outline for the code the LLM should generate.

Step three: Multi-shot prompt (e.g. generating the code for the analysis): "Please generate the code for `[step 1 of the analysis you defined earlier]`. Here is an example of code I found on the internet: `[add code-snippet]`. Use this snippet as an outline for our code and adapt it to our dataset and variables. Also always explain the code clearly and document each step in the code."

From that on you can continue with each step of the analysis in the same way. You can always test the code the LLM generated simultaneously in a Jupyter Notebook. If an error occurs you can ask the LLM to debug and specify the error-message you received.

Step four: Debugging (e.g. by asking for the reasons behind an error message): "I received the following error message in this step: `[add error message]`. Please debug the code and explain the reasons for the error."

The most crucial part in prompt engineering for python is viewing the LLM as an enhancement to your work and not as an replacement of critical thinking. The proposed workflow is iterative and focused around the reasoning of the LLM and therefore enables the user to doublecheck the generated output with reliable sources such as the documentations of the respective python-libraries. Therefore the interaction between the user and the LLM amples to oppurtunity to learn and improve your coding skills while already producing high-qualitiy data analysis in python.

Opportunities and risks of prompt engineering

Prompt engineering opens up a wide range of opportunities for data scientists and python coders. The generation of code is fast, efficient and often of high quality. The LLM-models can also assist in debugging code, analzying error or even help you analyse results of your analysis. Good prompts will help you define a strategy for your analysis, plan the execution of your workflow and explain steps you do not understand. In general prompt engineering enables users to achieve a level of knowledge and quality that might be very difficult to obtain for a beginner in python. Furthermore and probably even more important, prompt engineering can serve as a cornerstone in learning data science and python. Iterating on the output of the LLM, comparing it with other trusted sources, discovering errors and improvements is a great strategy to learn fast and generate adequate results in the meantime. In a broader point of view, prompt engineering contributes to the democratization of AI and data science, enabling more and more people to work on meaningful projects.

However, prompt engineering is also associated with risks. Crucially, the LLM works behind a curtain. If the user tests the code iteratively in a Jupyter Notebook, one could at least determine which code-snippet raises an error. But without a deep understanding of the code and the used statistical methods, you are forced to blindly trust the LLM. This implies the risks that the generated code will either not work or is very inefficient. In such cases, debugging the code can become a total nightmare. Especially if the user does not understand the code and the generated code-snippet is complex and long. Therefore even great prompt engineering might be not enough to guarantee high-quality results. In addition, prompt engineering might lead to a fatigue in learning the underlying python concepts and statistical methods. Some users might feel like it is not neccessary to double-check the generated code and will therefore remain on a rather low professional level. Also even prompts that consider a variety of different prompting techniques might not work out as expected. There is still variety in the generated output, even if you are using the same prompt. Of course the generated code will also differ depending on which model or tool you use. In a nutshell, prompt engineering might lead to a loss in the users critical thinking and problem-solving ability, therefore staying on the same level as a python-coder.

Normativity

Ethical considerations regarding prompt engineering become clear when we think about the usage of examples for enhancing our prompts. With regard to using examples in your prompt the danger of using private data (e.g. names, adresses, confidential information etc.) arrises. Even though a user can normally specify that their respective input data (e.g. what they type in ChatGPT) should not be used for training the model, it is not clear in every case what the respective company behind the LLM does with the data. Sharing private information about yourself or someone else while prompting can in the worst case lead to a data breach and vulnerable damage to the respective person, resulting in severe fines or other legal consequences. Also, the user must be sure that the provided code is not protected under any license and can be used for the intended purpose. Another reason for the danger of using examples might be a bias in the examples a user provides. The input can carry a sensitive bias which will be reproduces in the generated output. This might seem arbitrary when (only) generating code, but if you prompt the LLM to analyze and identify patterns in your code's outputs, the bias can be very harmful. Prompt engineering makes coding more accessable for everyone, including people that might have dark intentions. As a technique prompt engineering is neither good or bad, such as artificla intelligence itself. Nevertheless, it should be mentioned that prompt engineering might enable people to generate code for malicious purposes. Since prompt engineering becomes more an more common it might lead to an intransparency in research or work environments. If code is shared online, users might not be able anymore to differentiate between human written code and code that was written by or with using a LLM. Last but not least, most LLM-tools can even be embedded in automatic workflows trough the use of APIs. Therefore prompt engineering will contribute to shifts ins the labour market, leaving people in repetitive occoputations without a job. Some of theses issues can definitely not be specifically tailored towards prompt engineering, but are rather general issues regarding the usage of artifical intelligence. Nevertheless prompt engineering opens the door to use human language to interact with high complex and very developed AI-tools. Their role in possible negative consequences regarding an omnipotent usage of AI should not be underestimated.

Outlook

There are already approaches to use artificial intelligence to automatically suggest prompts while you are writing them, tailoring them to your specific needs and developed with a deep understanding of the respective model. So in the future we might not be forced to know high-quality prompting techniques, as the respective model will suggest them to us. In addition there are also large tech-companies like OpenAI compiling databases of high-quality prompts and examples. Those different databases and guidelines could be used to develop prompting-standards similar to coding-standards like PEP8 for python.

Further readings

Since examples might be helpful for understanding the tweaks of prompt engineering, here is a list of prompt engineering guides and examples from key-players in the AI-industry for you to refine your knowledge.

- 1 - OpenAI's guide to prompt engineering.
- 2 - Mistral's guide to prompt engineering.
- 3 - Anthropic's guide to prompt engineering.
- 4 - Google's guide to prompt engineering.
- 5 - Amazon's guide to prompt engineering.

For any **advanced python user**, this documentations might be helpful, when integrating and prompt engineering LLMs in your workflows:

- 6 - OpenAI's API documentation aimed at developers aiming to integrate LLMs directly in their workflows and applications.
- [https://www.make.com/en/help/app/openai-dall-e-chatgpt}] - Make.com's guide to integrating OpenAI's LLMs in your workflows through APIs. (make.com is a no or low-code platform for integrating AI-tools in your workflows. altough it is very convenient, it still requires you to prompt the LLM-modules)

References

- Amazon Web Services, Inc. "What Is LLM? - Large Language Models Explained - AWS," n.d. a https://aws.amazon.com/what-is/large-language-model/.
- Amazon Web Services, Inc. "What Is Prompt Engineering? - AI Prompt Engineering Explained - AWS," n.d. b https://aws.amazon.com/what-is/prompt-engineering/?nc1=h_ls.
- Lee, Angie. "What Are Large Language Models and Why Are They Important? | NVIDIA Blog." NVIDIA Blog, October 16, 2024. https://blogs.nvidia.com/blog/what-are-large-language-models-used-for/.

- Google for Developers. “Prompt Engineering for Generative AI,” n.d. https://developers.google.com/machine-learning/resources/prompt-eng.
- Mistral AI, Inc. “Tokenization | Mistral AI Large Language Models,” n.d. https://docs.mistral.ai/guides/tokenization/.
- Zaghir, Jamil, Marco Naguib, Mina Bjelogrlic, Aurélie Névéol, Xavier Tannier, and Christian Lovis. “Prompt Engineering Paradigms for Medical Applications: Scoping Review.” Journal of Medical Internet Research 26 (September 10, 2024): e60501. https://doi.org/10.2196/60501.

Author and date:

Luca Gemballa, 2024-12-11