SIX WEEK INTERNSHIP REPORT

ON

(PROMPT ENGINEERING)

COMPLETED AT

DOOZY INFOTECH - PATNA, BIHAR

Submitted By:
Aditya Raj(211537)
BATCH: 2021-2025

Department of Computer Science and Engineering

School of Engineering and Technology

CENTRAL UNIVERSITY OF HARYANA

Acknowledgement

I would like to express my special thanks to our mentor **Mr. Kunal** for his time and efforts he provided throughout the internship. Your useful advice and suggestions were really helpful to me during the project's completion. In this aspect, I am eternally grateful to you.

I would like to express my profound gratitude to **Dr. Rakesh Kumar, HoD of Department of Computer Science & Engineering , Central University of Haryana** for their contributions to the completion of my summer internship on Prompt Engineering at Doozy Infotech.

I am extremely grateful to my department staff members and friends who helped me in successful completion of this internship.

I would like to acknowledge that this internship was completed entirely by me, not by someone else.

Certificate

Doozy Infotech
BR260089063, MSME, GOI

getdoozy.tech
m@getdoozy.tech
Patna, Bihar

To Whom It May Concern,

Date: 7 AUG 2023

I am pleased to write this letter of recommendation for **Aditya Raj** from Central University of Haryana, who successfully completed a six-week internship with us at Doozy Infotech.

Throughout his tenure with us, Aditya consistently demonstrated an exceptional level of dedication, coupled with a strong work ethic and a genuine passion for learning. His proficiency in **Prompt Engineering** was particularly commendable and significantly contributed to the success of his assigned project.

One of Aditya's most noteworthy attributes is his adeptness at swiftly adapting to new challenges and technologies. He exhibited an impressive ability to grasp intricate concepts and effectively apply them in practical scenarios. His analytical thinking and adept problem-solving skills proved to be invaluable assets to our team.

Furthermore, Aditya displayed outstanding teamwork and communication skills. He collaborated seamlessly with colleagues, taking on leadership roles when necessary and fostering a positive work environment. His communication style, characterized by clarity and conciseness, greatly enriched our team dynamics.

We hold every confidence that Aditya Raj possesses a promising future in the field of computer science. We extend our warmest wishes to Aditya Raj in all of his future endeavors.

Kunal Kumar
Project Manager
Doozy Infotech

DZY/CC35/013/05

Doozy Infotech

CERTIFICATE

OF INTERNSHIP

This certificate is proudly awarded to

Aditya Raj

successfully completed a six-week internship at Doozy Infotech from June 26, 2023, to August 4, 2023.
His primary task was to work on a client project involving Prompt Engineering

Kunal

Project manager, Doozy

TABLE OF CONTENTS

Prompt Engineering

Prompt engineering is a relatively new discipline for developing and optimizing prompts to efficiently use language models (LMs) for a wide variety of applications and research topics. Prompt engineering skills help to better understand the capabilities and limitations of large language models (LLMs).

Researchers use prompt engineering to improve the capacity of LLMs on a wide range of common and complex tasks such as question answering and arithmetic reasoning. Developers use prompt engineering to design robust and effective prompting techniques that interface with LLMs and other tools.

Prompt engineering is not just about designing and developing prompts. It encompasses a wide range of skills and techniques that are useful for interacting and developing with LLMs. It's an important skill to interface, build with, and understand capabilities of LLMs. We can use prompt engineering to improve safety of LLMs and build new capabilities like augmenting LLMs with domain knowledge and external tools.

Motivated by the high interest in developing with LLMs, we have created this new prompt engineering guide that contains all the latest papers, learning guides, models, lectures, references, new LLM capabilities, and tools related to prompt engineering.

Introduction

Prompt engineering is an emerging discipline that focuses on creating and improving prompts for the effective usage of language models (LMs) in a wide range of applications and research areas. Quick engineering skills make it easier to comprehend the potential and constraints of large language models (LLMs). In order to increase LLMs' performance on a variety of common and challenging activities, including question-answering and mathematical reasoning, researchers utilize quick engineering. Prompt engineering is a strategy used by developers to create reliable and efficient prompting methods that interface with LLMs and other tools.

A Word on LLM Settings

When working with prompts, we interact with the LLM via an API or directly. we can configure a few parameters to get different results for wer prompts.

Temperature - In short, the lower the temperature, the more deterministic the results in the sense that the highest probable next token is always picked. Increasing temperature could lead to more randomness, which encourages more diverse or creative outputs. we are essentially increasing the weights of the other possible tokens. In terms of application, we might want to use a lower temperature value for tasks like fact-based QA to encourage more factual and concise responses. For poem generation or other creative tasks, it might be beneficial to increase the temperature value.

Top_p - Similarly, with top_p, a sampling technique with temperature called nucleus sampling, we can control how deterministic the model is at generating a response. If we are looking for exact and factual answers keep this low. If we are looking for more diverse responses, increase to a higher value.

The general recommendation is to alter one, not both.

Before starting with some basic examples, keep in mind that results may vary depending on the version of LLM we use.

Basics of Prompting

Basic Prompts

Simple prompts can help us achieve a lot, but how effectively they are written and how much information we provide them will determine how good the outcomes are. The command or query we are giving the model can be included in a prompt, along with other information like *context*, *inputs*, or *examples*. These components can help us educate the model more effectively and, as a result, produce better results.

Here is a basic example of a simple prompt:

Prompt

```
The Sky is
```

Output:

```
blue

The sky is blue on a clear day. On a cloudy day, the sky may be
grey or white.
```

As we can see, the language model outputs a continuation of strings that make sense given the context "***The sky is***". The output might be unexpected or far from the task we want to accomplish.

This basic example also highlights the necessity to provide more context or instructions on what specifically we want to achieve.

Let's try to improve it a bit:

Prompt:

```
Complete the sentence:

The sky is
```

Output:

```
 so  beautiful today.
```

Is that better? Well, we told the model to complete the sentence so the result looks a lot better as it follows exactly what we told it to do ("complete the sentence"). This approach of designing optimal prompts to instruct the model to perform a task is what's referred to as prompt engineering.

The example above is a basic illustration of what's possible with LLMs today. Today's LLMs can perform all kinds of advanced tasks that range from text summarization to mathematical reasoning to code generation.

Prompt Formatting:

We have tried a very simple prompt above. A standard prompt has the following format:

```
<Question>?
```

This can be formatted into a QA format, which is standard in a lot of QA dataset, as follows:

```
Q: <Question>?
A:
```

When prompting like the above, it's also referred to as zero-shot prompting, i.e., we are directly prompting the model for a response without any examples or demonstrations about the task we want it to achieve. Some large language models do have the ability to perform zero-shot prompting but it depends on the complexity and knowledge of the task at hand.

Given the standard format above, one popular and effective technique to prompting is referred to as few-shot prompting where we provide examples (i.e., demonstrations). we can format few-shot prompts as follows:

```
<Question>?
<Answer>

<Question>?
<Answer>

<Question>?
<Answer>

<Question>?
```

And we can already guess that its QA format version would look like this:

```
Q: <Question>?
A: <Answer>

Q: <Question>?
A: <Answer>

Q: <Question>?
A: <Answer>

Q: <Question>?
A:
```

Keep in mind that it's not required to use QA format. The format depends on the task at hand. For instance, we can perform a simple classification task and give examples that demonstrate the task as follows:

Prompt:

```
This is awesome! // Positive
This is bad! // Negative
Wow that movie was rad! // Positive
What a horrible show! //
```

Output:

```
Negative
```

Few-shot prompts enable in-context learning, which is the ability of language models to learn tasks given only a few examples. We will see more of this in action in the upcoming guides.

Elements of a Prompt :

As we cover more and more examples and applications with prompt engineering, we will notice that certain elements make up a prompt.

A prompt contains any of the following elements:

Instruction - a specific task or instruction we want the model to perform

Context - external information or additional context that can steer the model to better responses

Input Data - the input or question that we are interested to find a response for

Output Indicator - the type or format of the output.

we do not need all the four elements for a prompt and the format depends on the task at hand. We will touch on more concrete examples in upcoming guides.

General Tips for Designing Prompts

Here are some tips to keep in mind while we are designing wer prompts:

Start Simple

As we get started with designing prompts, we should keep in mind that it is really an iterative process that requires a lot of experimentation to get optimal results. Using a simple playground from OpenAI or Cohere is a good starting point.

We can start with simple prompts and keep adding more elements and context as we aim for better results. Iterating the prompt along the way is vital for this reason. As we read the guide, we will see many examples where specificity, simplicity, and conciseness will often give us better results.

When we have a big task that involves many different subtasks, we can try to break down the task into simpler subtasks and keep building up as we get better results. This avoids adding too much complexity to the prompt design process at the beginning.

The Instruction

You can design effective prompts for various simple tasks by using commands to instruct the model what you want to achieve, such as "Write", "Classify", "Summarise", "Translate", "Order", etc.

Keep in mind that you also need to experiment a lot to see what works best. Try different instructions with different keywords, contexts, and data and see what works best for your particular use case and task. Usually, the more specific and relevant the context is to the task you are trying to perform, the better. We will touch on the importance of sampling and adding more context in the upcoming guides.

Others recommend that you place instructions at the beginning of the prompt. Another recommendation is to use some clear separator like "###" to separate the instruction and context.

For instance:

Prompt:

```
### Instruction ###
Translate the text below to Spanish:

Text: "hello!"
```

Output:

```
¡Hola!
```

Specificity

Be very specific about the instruction and task you want the model to perform. The more descriptive and detailed the prompt is, the better the results. This is particularly important when you have a desired outcome or style of generation you are seeking. There aren't specific tokens or keywords that lead to better results. It's more important to have a good format and descriptive prompt. In fact, providing examples in the prompt is very effective to get desired output in specific formats.

When designing prompts, you should also keep in mind the length of the prompt as there are limitations regarding how long the prompt can be. Thinking about how specific and detailed you should be. Including too many unnecessary details is not necessarily a good approach. The details should be relevant and contribute to the task at hand. This is something you will need to experiment with a lot. We encourage a lot of experimentation and iteration to optimise prompts for your applications.

As an example, let's try a simple prompt to extract specific information from a piece of text.

Prompt:

```
Extract the name of places in the following text.

Desired format:
Place: <comma_separated_list_of_company_names>

Input: "Although these developments are encouraging to
researchers, much is still a mystery. "We often have a black box
between the brain and the effect we see in the periphery," says
Henrique Veiga-Fernandes, a neuroimmunologist at the Champalimaud
Centre for the Unknown in Lisbon. "If we want to use it in the
therapeutic context, we need to understand the mechanism."
```

Output:

```
Place: Champalimaud Centre for the Unknown, Lisbon
```

The input text obtained from [Your brain could be controlling how sick you get — and how you recover]().

Avoid Impreciseness

Given the tips above about being detailed and improving format, it's easy to fall into the trap of wanting to be too clever about prompts and potentially creating imprecise descriptions. It's often better to be specific and direct. The analogy here is very similar to effective communication -- the more direct, the more effective the message gets across.

For example, you might be interested in learning the concept of prompt engineering. You might try something like:

```
Explain the concept of prompt engineering. Keep the explanation
short, only a few sentences, and don't be too descriptive.
```

It's not clear from the prompt above how many sentences to use and what style. You might still somewhat get good responses with the above prompts but the better prompt would be one that is very specific, concise, and to the point. Something like:

```
Use 2-3 sentences to explain the concept of prompt engineering to
a high school student.
```

Examples of Prompts

The previous section introduced a basic example of how to prompt LLMs.

This section will provide more examples of how to use prompts to achieve different tasks and introduce key concepts along the way. Often, the best way to learn concepts is by going through examples. The few examples below illustrate how you can use well-crafted prompts to perform different types of tasks.

Topics:

- Text Summarization
- Information Extraction
- Question Answering
- Text Classification
- Conversation
- Code Generation
- Reasoning

Text Summarization

One of the standard tasks in natural language generation is text summarization. Text summarization can include many different flavours and domains. In fact, one of the most promising applications of language models is the ability to summarise articles and concepts into quick and easy-to-read summaries. Let's try a basic summarization task using prompts.

Let's say you are interested to learn about antibiotics, you could try a prompt like this:

prompt:

```
Explain antibiotics

A:
```

Output:

```
Antibiotics are a type of medication used to treat bacterial
infections. They work by either killing the bacteria or
preventing them from reproducing, allowing the body's immune
system to fight off the infection. Antibiotics are usually taken
orally in the form of pills, capsules, or liquid solutions, or
sometimes administered intravenously. They are not effective
against viral infections, and using them inappropriately can lead
to antibiotic resistance.
```

The "A:" is an explicit prompt format that you use in question answering. You used it here to tell the model that there is an answer expected further. In this example, it's not clear how this is useful vs not using it but we will leave it that for later examples. Let's just assume that this is too much information and you want to summarise it further. In fact, you can instruct the model to summarise into one sentence like so:

Prompt:

```
Antibiotics are a type of medication used to treat bacterial
infections. They work by either killing the bacteria or
preventing them from reproducing, allowing the body's immune
system to fight off the infection. Antibiotics are usually taken
orally in the form of pills, capsules, or liquid solutions, or
sometimes administered intravenously. They are not effective
against viral infections, and using them inappropriately can lead
to antibiotic resistance.

Explain the above in one sentence:
```

Output:

```
Antibiotics are medications used to treat bacterial infections by
either killing the bacteria or stopping them from reproducing,
but they are not effective against viruses and overuse can lead
to antibiotic resistance.
```

Without paying too much attention to the accuracy of the output above, which is something we will touch on in a later guide, the model tried to summarise the paragraph in one sentence. You can get clever with the instructions but we will leave that for a later chapter. Feel free to pause here and experiment to see if you get better results.

Information Extraction

While language models are trained to perform natural language generation and related tasks, it's also very capable of performing classification and a range of other natural language processing (NLP) tasks.

Here is an example of a prompt that extracts information from a given paragraph.

Prompt:

```
Author-contribution statements and acknowledgements in research
papers should state clearly and specifically whether, and to what
extent, the authors used AI technologies such as ChatGPT in the
preparation of their manuscript and analysis. They should also
indicate which LLMs were used. This will alert editors and
reviewers to scrutinise manuscripts more carefully for potential
biases, inaccuracies and improper source crediting. Likewise,
scientific journals should be transparent about their use of
LLMs, for example when selecting submitted manuscripts.

Mention the large language model based product mentioned in the
paragraph above:
```

Output:

```
The large language model based product mentioned in the
paragraph above is ChatGPT.
```

There are many ways we can improve the results above, but this is already very useful.

By now it should be obvious that you can ask the model to perform different tasks by simply instructing it what to do. That's a powerful capability that AI product builders are already using to build powerful products and experiences.

Paragraph source: ChatGPT: five priorities for research

Question Answering

One of the best ways to get the model to respond to specific answers is to improve the format of the prompt. As covered before, a prompt could combine instructions, context, input, and output indicators to get improved results. While these components are not required, it becomes a good practice as the more specific you are with instruction, the better results you will get. Below is an example of how this would look following a more structured prompt.

Prompt:

```
Answer the question based on the context below. Keep the answer
short. Respond "Unsure about answer" if not sure about the
answer.

Context: Teplizumab traces its roots to a New Jersey drug company
called Ortho Pharmaceutical. There, scientists generated an early
version of the antibody, dubbed OKT3. Originally sourced from
mice, the molecule was able to bind to the surface of T cells and
limit their cell-killing potential. In 1986, it was approved to
help prevent organ rejection after kidney transplants, making it
the first therapeutic antibody allowed for human use.

Question: What was OKT3 originally sourced from?

Answer:
```

Output:

```
Mice.
```

Context obtained from Nature.

Text Classification

So far, you have used simple instructions to perform a task. As a prompt engineer, you need to get better at providing better instructions. But that's not all! You will also find that for harder use cases, just providing instructions won't be enough. This is where you need to think more about the context and the different elements you can use in a prompt. Other elements you can provide are `input data` or `examples`.

Let's try to demonstrate this by providing an example of text classification.

Prompt:

```
Classify the text into neutral, negative or positive.

Text: I think the food was okay.
Sentiment:
```

Output:

```
Neutral
```

You gave the instruction to classify the text and the model responded with `'Neutral'`, which is correct. Nothing is wrong with this but let's say that what you really need is for the model to give the label in the exact format you want. So instead of `Neutral`, you want it to return `neutral`. How do you achieve this? There are different ways to do this. You care about specificity here, so the more information you can provide the prompt, the better results. You can try providing examples to specify the correct behaviour. Let's try again:

Prompt:

```
Classify the text into neutral, negative or positive.

Text: I think the vacation is okay.
Sentiment: neutral

Text: I think the food was okay.
Sentiment:
```

OutPut:

```
neutral
```

Perfect! This time the model returned `neutral` which is the specific label I was looking for. It seems that the example provided in the prompt helped the model to be specific in its output.

Conversation

Perhaps one of the more interesting things you can achieve with prompt engineering is instructing the LLM system on how to behave, its intent, and its identity. This is particularly useful when you are building conversational systems like customer service chatbots.

For instance, let's create a conversational system that's able to generate more technical and scientific responses to questions. Note how you are explicitly telling it how to behave through the instruction. This is sometimes referred to as *role prompting*.

Prompt:

```
The following is a conversation with an AI research assistant.
The assistant tone is technical and scientific.

Human: Hello, who are you?
AI: Greetings! I am an AI research assistant. How can I help you
today?
Human: Can you tell me about the creation of blackholes?
AI:
```

Output:

```
Sure! Black holes are regions of spacetime where the
gravitational force is so strong that nothing, not even light,
can escape from it. They are created when a very massive star
dies and its core collapses in on itself, forming a singularity
of infinite density. The intense gravity of the singularity pulls
in all the matter and radiation around it, creating the black
hole.
```

Our AI research assistant sounds a bit too technical, right? Okay, let's change this behaviour and instruct the system to give more accessible answers.

Prompt:

```
The following is a conversation with an AI research assistant.
The assistant answers should be easy to understand even by
primary school students.

Human: Hello, who are you?
AI: Greetings! I am an AI research assistant. How can I help you
today?
Human: Can you tell me about the creation of black holes?
AI:
```

Output:

```
Sure! Black holes are created when a very massive star runs out
of fuel and collapses in on itself. The gravity of the star is so
strong that nothing, not even light, can escape its pull. This
creates a point in space-time with an intense gravitational force
called a black hole.
```

You made some progress. You can continue improving it. If you add more examples, you might get even better results.

Code Generation

One application where LLMs are quite effective is code generation. Copilot is a great example of this. There are a vast number of code-generation tasks you can perform with clever prompts. Let's look at a few examples below.

First, let's try a simple program that greets the user.

Prompt:

```
/*
Ask the user for their name and say "Hello"
*/
```

Output:

```
let name = prompt("What is your name?");
console.log(`Hello, ${name}!`);
```

Notice that you didn't even need to specify the language to use.

Let's switch levels a bit. The example below shows how powerful LLMs can be with a little more effort in designing the prompts.

Prompt:

```
"""
Table departments, columns = [DepartmentId, DepartmentName]
Table students, columns = [DepartmentId, StudentId, StudentName]
Create a MySQL query for all students in the Computer Science
Department
"""
```

Output:

```
SELECT StudentId, StudentName
FROM students
WHERE DepartmentId IN (SELECT DepartmentId FROM departments WHERE
DepartmentName = 'Computer Science');
```

This is very impressive. In this case, you provided data about the database schema and asked it to generate a valid MySQL query.

Reasoning

Perhaps one of the most difficult tasks for an LLM today is one that requires some form of reasoning. Reasoning is one of most interesting areas due to the types of complex applications that can emerge from LLMs.

There have been some improvements in tasks involving mathematical capabilities. That said, it's important to note that current LLMs struggle to perform reasoning tasks so this requires even more advanced prompt engineering techniques. We will cover these advanced techniques in the next guide. For now, we will cover a few basic examples to show arithmetic capabilities.

Prompt:

```
What is 9,000 * 9,000?
```

Output:

```
81,000,000
```

Let's try something more difficult.

Prompt:

```
The odd numbers in this group add up to an even number: 15, 32,
5, 13, 82, 7, 1.

A:
```

Output:

```
No, the odd numbers in this group add up to an odd number: 119.
```

That's incorrect! Let's try to improve this by improving the prompt.

Prompt:

```
The odd numbers in this group add up to an even number: 15, 32,
5, 13, 82, 7, 1.

Solve by breaking the problem into steps. First, identify the odd
numbers, add them, and indicate whether the result is odd or
even.
```

Output:

```
Odd numbers: 15, 5, 13, 7, 1
Sum: 41
41 is an odd number.
```

Much better, right? By the way, I tried this a couple of times and the system sometimes fails. If you provide better instruction combined with examples, it might help get more accurate results.

We will continue to include more examples of common applications in this section of the guide.

In the upcoming guides, we will cover even more advanced prompt engineering concepts for improving performance on all these and more difficult tasks.

Advanced Prompting

By this point, it should be obvious that it helps to improve prompts to get better results on different tasks. That's the whole idea behind prompt engineering.

While those examples were fun, let's cover a few concepts more formally before we jump into more advanced concepts.

Topics:

- **Zero-shot Prompting**
- **Few-shot Prompting**
- **Chain-of-Thought Prompting**
- **Zero-shot CoT**
- **Self-Consistency**
- **Generate Knowledge Prompting**
- **Automatic Prompt Engineer**
- **Zero-Shot Prompting**

LLMs today trained on large amounts of data and tuned to follow instructions, are capable of performing tasks zero-shot. We tried a few zero-shot examples in the previous section. Here is one of the examples we used:

Prompt:

```
Classify the text into neutral, negative, or positive.

Text: I think the vacation is okay.
Sentiment:
```

Output:

```
Neutral
```

Note that in the prompt above we didn't provide the model with any examples -- that's the zero-shot capabilities at work. When zero-shot doesn't work, it's recommended to provide demonstrations or examples in the prompt. Below we discuss the approach known as few-shot prompting.

Few-Shot Prompting

While large-language models already demonstrate remarkable zero-shot capabilities, they still fall short on more complex tasks when using the zero-shot setting. To improve on this, few-shot prompting is used as a technique to enable in-context learning where we provide demonstrations in the prompt to steer the model to better performance. The demonstrations serve as conditioning for subsequent examples where we would like the model to generate a response.

Let's demonstrate few-shot prompting via an example that was presented by Brown et al. 2020. In the example, the task is to correctly use a new word in a sentence.

Prompt:

```
A "whatpu" is a small, furry animal native to Tanzania. An
example of a sentence that uses
the word whatpu is:
We were travelling in Africa and we saw these very cute whatpus.
To do a "farduddle" means to jump up and down really fast. An
example of a sentence that uses
the word farduddle is:
```

Output:

```
When we won the game, we all started to farduddle in celebration.
```

We can observe that the model has somehow learned how to perform the task by providing it with just one example (i.e., 1-shot). For more difficult tasks, we can experiment with increasing the demonstrations (e.g., 3-shot, 5-shot, 10-shot, etc.).

Following the findings from Min et al. (2022) , here are a few more tips about demonstrations/exemplars when doing few-shot:

- "the label space and the distribution of the input text specified by the demonstrations are both important (regardless of whether the labels are correct for individual inputs)"
- The format you use also plays a key role in performance, even if you just use random labels, this is much better than no labels at all.

- Additional results show that selecting random labels from a true distribution of labels (instead of a uniform distribution) also helps.

Let's try out a few examples. Let's first try an example with random labels (meaning the labels Negative and Positive are randomly assigned to the inputs):

Prompt:

```
This is awesome! // Negative
This is bad! // Positive
Wow that movie was rad! // Positive
What a horrible show! //
```

Output:

```
Negative
```

We still get the correct answer, even though the labels have been randomised. Note that we also kept the format, which helps too. In fact, with further experimentation, it seems the newer GPT models we are experimenting with are becoming more robust to even random formats.
Example:

Prompt:

```
Positive This is awesome!
This is bad! Negative
Wow that movie was rad!
Positive
What a horrible show! --
```

Output:

```
Negative
```

There is no consistency in the format above but the model still predicted the correct label. We have to conduct a more thorough analysis to confirm if this holds for different and more complex tasks, including different variations of prompts.

Chain-of-Thought Prompting

Introduced in Wei et al. (2022) , chain-of-thought (CoT) prompting enables complex reasoning capabilities through intermediate reasoning steps. You can combine it with few-shot prompting to get better results on more complex tasks that require reasoning before responding.

Prompt:

```
The odd numbers in this group add up to an even number: 4, 8, 9,
15, 12, 2, 1.
A: Adding all the odd numbers (9, 15, 1) gives 25. The answer is
False.

The odd numbers in this group add up to an even number: 17,  10,
19, 4, 8, 12, 24.
A: Adding all the odd numbers (17, 19) gives 36. The answer is
True.

The odd numbers in this group add up to an even number: 16,  11,
14, 4, 8, 13, 24.
A: Adding all the odd numbers (11, 13) gives 24. The answer is
True.

The odd numbers in this group add up to an even number: 17,  9,
10, 12, 13, 4, 2.
A: Adding all the odd numbers (17, 9, 13) gives 39. The answer is
False.

The odd numbers in this group add up to an even number: 15, 32,
5, 13, 82, 7, 1.
A:
```

Output:

```
Adding all the odd numbers (15, 5, 13, 7, 1) gives 41. The answer
is False.
```

Wow! We can see a perfect result when we provide the reasoning step. We can solve this task by providing even fewer examples, i.e., just one example seems enough:

Prompt:

```
The odd numbers in this group add up to an even number: 4, 8, 9,
15, 12, 2, 1.
A: Adding all the odd numbers (9, 15, 1) gives 25. The answer is
False.

The odd numbers in this group add up to an even number: 15, 32,
5, 13, 82, 7, 1.
A:
```

Output:

```
Adding all the odd numbers (15, 5, 13, 7, 1) gives 41. The answer
is False.
```

Keep in mind that the authors claim that this is an emergent ability that arises with sufficiently large language models.

Zero-Shot CoT

One recent idea that came out more recently is the idea of zero-shot CoT (Kojima et al. 2022) that essentially involves adding "Let's think step by step" to the original prompt. Let's try a simple problem and see how the model performs:

Prompt:

```
I went to the market and bought 10 apples. I gave 2 apples to the
neighbour and 2 to the repairman. I then went and bought 5 more
apples and ate 1. How many apples did I remain with?
```

Output:

```
11 apples
```

The answer is incorrect! Now Let's try with the special prompt.

Prompt:

```
I went to the market and bought 10 apples. I gave 2 apples to the
neighbour and 2 to the repairman. I then went and bought 5 more
apples and ate 1. How many apples did I remain with?

Let's think step by step.
```

Output:

```
First, you started with 10 apples.
You gave away 2 apples to the neighbour and 2 to the repairman,
so you had 6 apples left.
Then you bought 5 more apples, so now you had 11 apples.
Finally, you ate 1 apple, so you would remain with 10 apples
```

.

It's impressive that this simple prompt is effective at this task. This is particularly useful where you don't have too many examples to use in the prompt.

Self-Consistency

Perhaps one of the more advanced techniques out there for prompt engineering is self-consistency. Proposed by [Wang et al. (2022)](), self-consistency aims "to replace the naive greedy decoding used in chain-of-thought prompting". The idea is to sample multiple, diverse reasoning paths through few-shot CoT, and use the generations to select the most consistent answer. This helps to boost the performance of CoT prompting on tasks involving arithmetic and commonsense reasoning.

Let's try the following example for arithmetic reasoning:

Prompt:

```
When I was 6 my sister was half my age. Now
I'm 70. How old is my sister?
```

```
```

Output:

```
35
```

The output is wrong! How may we improve this with self-consistency? Let's try it out. We will use the few-shot exemplars from Wang et al. 2022 (Table 17):

Prompt:

```
Q: There are 15 trees in the grove. Grove workers will plant
trees in the grove today. After they are done,
There will be 21 trees. How many trees did the grove workers
plant today?
A: We start with 15 trees. Later we have 21 trees. The difference
must be the number of trees they planted.
So, they must have planted 21 - 15 = 6 trees. The answer is 6.

Q: If there are 3 cars in the parking lot and 2 more cars arrive,
how many cars are in the parking lot?
A: There are 3 cars in the parking lot already. 2 more arrive.
Now there are 3 + 2 = 5 cars. The answer is 5.

Q: Leah had 32 chocolates and her sister had 42. If they ate 35,
how many pieces do they have left in total?
A: Leah had 32 chocolates and Leah’s sister had 42. That means
there were originally 32 + 42 = 74
chocolates. 35 have been eaten. So in total they still have 74 -
35 = 39 chocolates. The answer is 39.

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now
Jason has 12 lollipops. How many lollipops
did Jason give to Denny?
A: Jason had 20 lollipops. Since he only has 12 now, he must have
given the rest to Denny. The number of
lollipops he has given to Denny must have been 20 - 12 = 8
lollipops. The answer is 8.

Q: Shawn has five toys. For Christmas, he got two toys each from
his mom and dad. How many toys does
he have now?
```

```
A: He has 5 toys. He got 2 from mom, so after that he has 5 + 2 =
7 toys. Then he got 2 more from dad, so
in total he has 7 + 2 = 9 toys. The answer is 9.

Q: There were nine computers in the server room. Five more
computers were installed each day, from
monday to thursday. How many computers are now in the server
room?
A: There are 4 days from monday to thursday. 5 computers were
added each day. That means in total 4 * 5 =
20 computers were added. There were 9 computers in the beginning,
so now there are 9 + 20 = 29 computers.
The answer is 29.

Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls.
On wednesday, he lost 2 more. How many
golf balls did he have at the end of wednesday?
A: Michael initially had 58 balls. He lost 23 on Tuesday, so
after that he has 58 - 23 = 35 balls. On
Wednesday he lost 2 more so now he has 35 - 2 = 33 balls. The
answer is 33.

Q: Olivia has $23. She bought five bagels for $3 each. How much
money does she have left?
A: She bought 5 bagels for $3 each. This means she spent 5

Q: When I was 6 my sister was half my age. Now I'm 70 how old is
my sister?
A:
```

Output 1:

```
When I was 6 my sister was half my age, so she was 3. Now I am
70, so she is 70 - 3 = 67. The answer is 67.
```

Output 2:

```
When the narrator was 6, his sister was half his age, which is 3.
Now that the narrator is 70, his sister would be 70 - 3 = 67
years old. The answer is 67.
```

Output 3:

```
When I was 6 my sister was half my age, so she was 3. Now I am
70, so she is 70/2 = 35. The answer is 35.
```

Computing for the final answer involves a few steps (check out the paper for the details) but for the sake of simplicity, we can see that there is already a majority answer emerging so that would essentially become the final answer.

Generated Knowledge Prompting

LLMs continue to be improved and one popular technique includes the ability to incorporate knowledge or information to help the model make more accurate predictions.

Using a similar idea, can the model also be used to generate knowledge before making a prediction? That's what is attempted in the paper by Liu et al. 2022 -- generate knowledge to be used as part of the prompt. In particular, how helpful is this for tasks such as commonsense reasoning?

Let's try a simple prompt:

Prompt:

```
Part of golf is trying to get a higher point total than others.
Yes or No?
```

Output:

```
Yes.
```

This type of mistake reveals the limitations of LLMs to perform tasks that require more knowledge about the world. How do we improve this with knowledge generation?

First, we generate a few "knowledges":

Prompt:

```
Input: Greece is larger than mexico.
Knowledge: Greece is approximately 131,957 sq km, while Mexico is
approximately 1,964,375 sq km, making Mexico 1,389% larger than
Greece.

Input: Glasses always fog up.
Knowledge: Condensation occurs on eyeglass lenses when water
vapour from your sweat, breath, and ambient humidity lands on a
cold surface, cools, and then changes into tiny drops of liquid,
forming a film that you see as fog. Your lenses will be
relatively cool compared to your breath, especially when the
outside air is cold.

Input: A fish is capable of thinking.
Knowledge: Fish are more intelligent than they appear. In many
areas, such as memory, their cognitive powers match or exceed
those of 'higher' vertebrates including non-human primates.
Fish's long-term memories help them keep track of complex social
relationships.

Input: A common effect of smoking lots of cigarettes in one's
lifetime is a higher than normal chance of getting lung cancer.
Knowledge: Those who consistently averaged less than one
cigarette per day over their lifetime had nine times the risk of
dying from lung cancer than never smokers. Among people who
smoked between one and 10 cigarettes per day, the risk of dying
from lung cancer was nearly 12 times higher than that of never
smokers.

Input: A rock is the same size as a pebble.
Knowledge: A pebble is a clast of rock with a particle size of 4
to 64 millimetres based on the Udden-Wentworth scale of
sedimentology. Pebbles are generally considered larger than
granules (2 to 4 millimetres diameter) and smaller than cobbles
(64 to 256 millimetres diameter).

Input: Part of golf is trying to get a higher point total than
others.
```

Knowledge:

Knowledge 1:

```
The objective of golf is to play a set of holes in the least
number of strokes. A round of golf typically consists of 18
holes. Each hole is played once in the round on a standard golf
course. Each stroke is counted as one point, and the total number
of strokes is used to determine the winner of the game.
```

Knowledge 2:

```
Golf is a precision club-and-ball sport in which competing
players (or golfers) use many types of clubs to hit balls into a
series of holes on a course using the fewest number of strokes.
The goal is to complete the course with the lowest score, which
is calculated by adding up the total number of strokes taken on
each hole. The player with the lowest score wins the game.
```

We are using the prompt provided in the paper by Liu et al. 2022.

Automatic Prompt Engineer (APE)

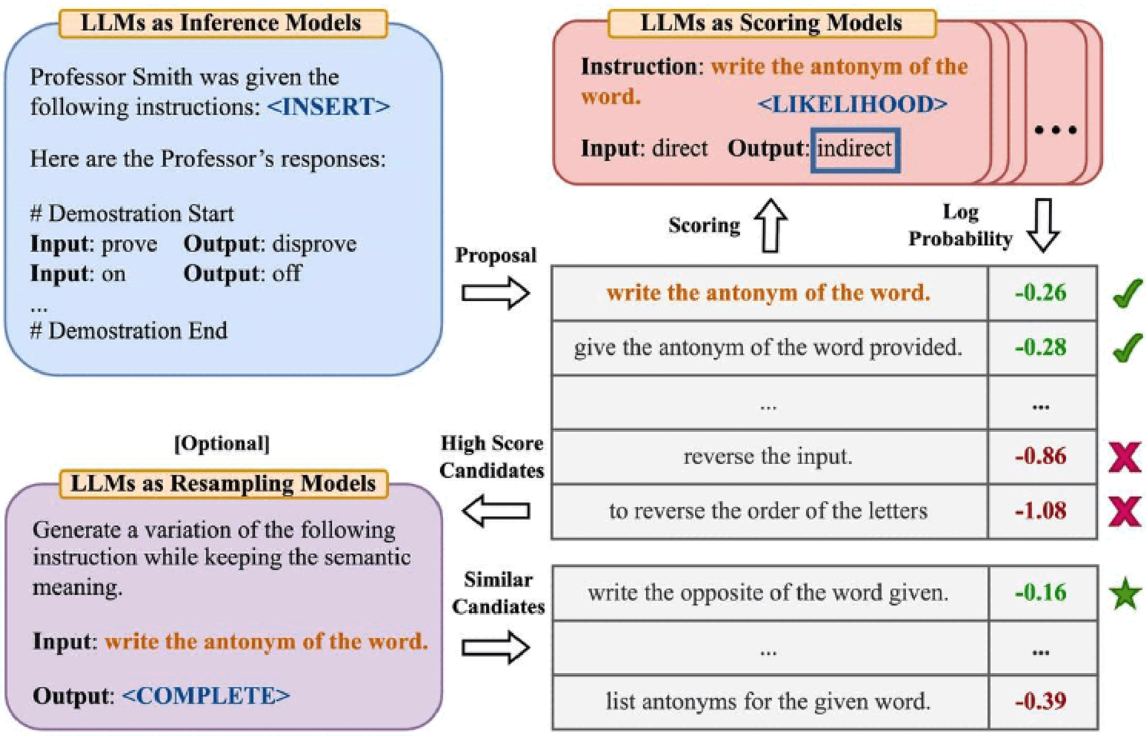

Image Source**:** [Zhou et al., (2022)]

[Zhou et al., (2022)] propose automatic prompt engineer (APE) , a framework for automatic instruction generation and selection. The instruction generation problem is framed as natural language synthesis addressed as a black-box optimization problem using LLMs to generate and search over candidate solutions.

The first step involves a large language model (as an inference model) that is given output demonstrations to generate instruction candidates for a task. These candidate solutions will guide the search procedure. The instructions are executed using a target model, and then the most appropriate instruction is selected based on computed evaluation scores.

APE discovers a better zero-shot CoT prompt than the human engineered "Let's think step by step" prompt(Kojima et al., 2022).

The prompt "Let's work this out in a step by step way to be sure we have the right answer." elicits chain-of-though reasoning and improves performance on the MultiArith and GSM8K benchmarks:

No.	Category	Zero-shot CoT Trigger Prompt	Accuracy
1	APE	Let's work this out in a step by step way to be sure we have the right answer.	**82.0**
2	Human-Designed	Let's think step by step. (*1)	78.7
3		First, (*2)	77.3
4		Let's think about this logically.	74.5
5		Let's solve this problem by splitting it into steps. (*3)	72.2
6		Let's be realistic and think step by step.	70.8
7		Let's think like a detective step by step.	70.3
8		Let's think	57.5
9		Before we dive into the answer,	55.7
10		The answer is after the proof.	45.7
-		(Zero-shot)	17.7

Image Source: [Zhou et al., (2022)]

This paper touches on an important topic related to prompt engineering which is the idea of automatically optimising prompts. While we don't go deep into this topic in this guide, here are a few key papers if you are interested in the topic:

- **AutoPrompt -** proposes an approach to automatically create prompts for a diverse set of tasks based on gradient-guided search.
- **Prefix Tuning -** a lightweight alternative to fine-tuning that prepends a trainable continuous prefix for NLG tasks.
- **Prompt Tuning -** proposes a mechanism for learning soft prompts through backpropagation.

Prompt Applications

Generating Data

LLMs have strong capabilities to generate text. Using effective prompt strategies can steer the model to produce better, more consistent, and more factual responses. LLMs can also be especially useful for generating data which is useful to run all sorts of experiments. For example, we can use it to generate quick samples for a sentiment classifier like so:

Prompt:

```
Produce 10 examples for sentiment analysis. Examples are
categorised as either positive or negative. Produce 2 negative
examples and 8 positive examples. Use this format for the
examples:
Q: <sentence>
A: <sentiment>
```

Output:

```
Q: I just got the best news ever!
A: Positive

Q: We just got a raise at work!
A: Positive

Q: I'm so proud of what I accomplished today.
A: Positive

Q: I'm having the best day ever!
A: Positive

Q: I'm really looking forward to the weekend.
A: Positive

Q: I just got the best present ever!
A: Positive

Q: I'm so happy right now.
```

```
A: Positive

Q: I'm so blessed to have such an amazing family.
A: Positive

Q: The weather outside is so gloomy.
A: Negative

Q: I just got some terrible news.
A: Negative
```

This is very useful. We use this example for a different test in another section of the guides.

Here is another prompt example showing how to leverage the LLM for data labelling/augmentation tasks.

Prompt:

```
produce 3 wine reviews and label taste, flavour, aroma related token;
present the result as a json file, in addition add the coordinate of each
term for NER task
```

Output:


```
[
  {
    "review": "This Chardonnay has a rich, creamy texture with
notes of vanilla and buttery oak. The taste is smooth and
well-balanced, with a hint of tropical fruit on the finish.",
    "tokens": [
      {"text": "rich", "label": "flavour", "start": 5, "end": 9},
      {"text": "creamy", "label": "texture", "start": 10, "end":
16},
      {"text": "vanilla", "label": "flavour", "start": 28, "end":
35},
      {"text": "buttery", "label": "flavour", "start": 40, "end":
47},
      {"text": "oak", "label": "aroma", "start": 48, "end": 51},
      {"text": "smooth", "label": "taste", "start": 61, "end":
67},
```

```
      {"text": "well-balanced", "label": "taste", "start": 68,
"end": 81},
      {"text": "hint", "label": "flavour", "start": 89, "end":
93},
      {"text": "tropical fruit", "label": "flavour", "start": 98,
"end": 112},
      {"text": "finish", "label": "taste", "start": 116, "end":
122}
    ]
  },
  {
    "review": "This Cabernet Sauvignon has a bold, full-bodied
flavour with notes of black cherry and dark chocolate. The aroma
is rich and complex, with hints of tobacco and leather.",
    "tokens": [
      {"text": "bold", "label": "flavour", "start": 5, "end": 9},
      {"text": "full-bodied", "label": "texture", "start": 10,
"end": 21},
      {"text": "black cherry", "label": "flavour", "start": 30,
"end": 42},
      {"text": "dark chocolate", "label": "flavour", "start": 47,
"end": 60},
      {"text": "rich", "label": "aroma", "start": 69, "end": 73},
      {"text": "complex", "label": "aroma", "start": 78, "end":
85},
      {"text": "hints", "label": "aroma", "start": 90, "end":
96},
      {"text": "tobacco", "label": "aroma", "start": 101, "end":
108},
      {"text": "leather", "label": "aroma", "start": 113, "end":
120}
    ]
  },
  {
    "review": "This Riesling has a crisp, refreshing taste with
notes of green apple and honey. The aroma is floral and fruity,
with a hint of citrus.",
    "tokens": [
      {"text": "crisp", "label": "texture", "start": 5, "end":
10},
      {"text": "refreshing", "label": "texture", "start": 12,
"end": 22},
      {"text": "green apple", "label": "flavour", "start": 31,
"end": 42},
    ]
  }
```

PROJECT TITLE:

Optimising AI Responses with Prompt Engineering: A Comprehensive A/B Testing Study

During my internship at Doozy Infotech, I had the incredible opportunity to delve into the fascinating world of prompt engineering, a crucial aspect of improving user interactions with AI models. Over the course of six week, I embarked on a comprehensive journey to test, evaluate, and optimise prompts for enhanced AI responses.

Exploring the Role of Prompt Engineering

Prompt engineering, as I discovered, is the art of crafting user inputs that yield the best possible results from AI models. It's about tailoring the way we ask questions or give commands to AI systems to ensure that the responses are not only accurate but also contextually relevant and user-friendly.

Daily Testing of 100 Prompts

A significant part of my internship involved the daily testing of approximately 100 prompts. These prompts came in various forms, including text and audio inputs, mirroring the diverse ways users engage with AI. This daily testing routine allowed me to explore the nuances of different prompts and understand their impact on AI performance.

The A/B Testing Approach

To thoroughly evaluate prompt effectiveness, I implemented an A/B testing approach. Each day, I randomly assigned two different prompts, Prompt A and Prompt B, to AI algorithms developed by Doozy Infotech. This methodology helped me compare the AI responses generated by these prompts.

Manual Ranking of AI Responses

Beyond quantitative metrics, I recognized the importance of the human touch in assessing prompt quality. Hence, I manually ranked the AI responses based on various criteria, such as coherence, informativeness, and naturalness. This meticulous ranking process offered valuable qualitative insights into the prompts' performance.

Six week of Discovery

Over the course of 45 days, I witnessed fascinating trends and patterns in prompt effectiveness. Some prompts consistently led to more accurate and user-friendly responses, while others revealed areas for improvement. This journey not only honed my analytical skills but also deepened my appreciation for the intricacies of prompt engineering.

The Bigger Picture

My internship at Doozy Infotech wasn't just about testing prompts; it was a holistic learning experience. I had the privilege of working with a talented team, gaining insights into AI algorithms, and contributing to the company's mission of enhancing user interactions with AI technology.

In conclusion, my internship in prompt engineering at **Doozy Infotech** was a transformative experience that allowed me to explore the intersection of AI, user experience, and human communication. It reinforced my passion for the field and equipped me with valuable skills that I'm excited to carry forward in my future endeavours.

About Doozy Infotech:

Doozy Infotech is a pioneering force in the world of advanced language models and a trusted leader in the industry. Doozy's expertise lies in providing cutting-edge Large Language Models (LLMs) that power a wide range of applications across various sectors.

Founded on a commitment to innovation and a passion for harnessing the power of language, Doozy has emerged as a go-to provider for organizations seeking to leverage the potential of natural language processing. The LLM solutions are designed to revolutionise communication, enabling businesses to interact, understand, and engage with their audience in unprecedented ways.

In addition to proficiency in language models, Doozy infotech boasts a dynamic team of experienced professionals with a diverse skill set. Doozy is also at the forefront of software development, app development, and website development. Experts of Doozy collaborate seamlessly to create bespoke solutions tailored to meet the unique needs of each client.

With a proven track record of numerous successful projects and a client base spanning various industries, Doozy has earned a reputation for excellence.

Doozy's mission is to empower businesses with the tools they need to thrive in the digital age. Whether it's through the state-of-the-art language models or our bespoke software and app development services, doozy is committed to helping the clients achieve their goals and exceed their expectations.

Thank You