COMP 10261: Prompt Engineering

Sam Scott, Mohawk College, June 2023

Whether you are a user of a Large Language Mode (LLM) or a software developer trying to incorporate an LLM into a specific app, the key to getting the best possible performance is good **prompt engineering**. Prompt engineering is a form of **transfer learning** that is specific to large language models.

Training vs. Transfer Learning

Suppose you wanted to create a large language model that is specialized to answer questions about AI. You might get the best results by training the model only on text relevant to AI and related topics. But training a model from scratch can be extremely expensive. A cheaper alternative might be to use a pre-trained LLM that was trained on a wide variety of text, and then tweak it slightly so that it is focused in on the task of asking questions about AI. You are hoping that its general learning will **transfer** over to the task you are interested in. Prompt engineering represents one very simple and accessible way to leverage what the system has already learned and turn it towards a new task.

Just the Prompt

One way to use an LLM is simply to talk to it as you would a human being, not worrying about any special kind of prompt engineering. This can sometimes work just fine, as shown by this example from the openai web site: https://platform.openai.com/examples/default-grammar

```
>>> Correct this to standard English: She no went to the market.

She did not go to the market.
```

There are many cases like the above, in which an LLM can do what was asked of it without any extra training or examples. But there are also lots of cases in which an LLM might provide incorrect information or behave less optimally if given just a simple prompt.

Prompt engineering is about adding information to the prompt to help the bot do better.

Prompt Engineering Techniques

Prompt Engineering is a very new field and at the time of writing there's not necessarily any stable taxonomy of different techniques. The list below is my own creation, drawn from several sources.[1] [2]

[1] Note that all the examples in this section are real responses generated at https://platform.openai.com/playground using the **text-davinci-003** model, usually with **temperature** set to 0.
[2] For an up to date rundown of current prompt engineering techniques, you could try https://www.promptingguide.ai/techniques

1. Prompting with Context

Sometimes just providing a bit more contextual information can help the model focus in on what you want.

Here's an example of a user interacting with GPT-3.

```
>>> Write a two sentence story.

The sun was setting, and the sky was painted with a beautiful
array of colors. She watched in awe, feeling a sense of peace
wash over her.
```

This is a decent response, but maybe it's not to the user's taste. A little bit of prompt engineering might allow the user to get a story they would like more.

```
>>> I am very interested in the writings of Edward Gorey. I like
disturbing stories about detectives. Write a two sentence story
that I would enjoy.

Detective Smith arrived at the scene of the crime, only to find
that the victim had been killed in a manner eerily similar to the
illustrations in Edward Gorey's books. As he investigated
further, he began to suspect that the murderer was inspired by
the macabre stories of the famous author.
```

2. Prompting for Success

Large language models are imitators – they have no success goals. It has been shown that answer quality can be improved if the models are explicitly told to be sure that they have the right answer. Adding the following to the prompts can be helpful:

```
Let's think this out in a step by step way to be sure we have the
right answer.

You are a leading expert on this topic.

Etc.
```

3. Prompting with New Information

Here's an example of a prompt that a user might type into a Mohawk College information bot. Suppose the bot simply passes the user's prompt an LLM for a response. Here's what the user might get…

```
>>> Who should I talk to about the Software Development program
at Mohawk College?

You can contact the Mohawk College Admissions Office at 905-575-
1212 or email admissions@mohawkcollege.ca for more information
about the Software Development program.
```

It sounds great, but there are lots of problems with this response:

- It's quite generic, not specifically about Software Development.
- The email address does not exist.
- The phone number is the general college phone number not the admissions number or a number related to Software Development.

If the bot developer had added a bit of context to the user's prompt before sending it off, the LLM might have done much better.

Here's an example of information that the developer could add to the prompt:

```
Here is some information about the Mohawk College CSAIT
department:
Acronym: CSAIT stands for Computer Science and Information
Technology
Programs: Software Support (2 years), Software Development (3
years), Network Systems (2 years), and Network Engineering and
Security Analyst (3 years)
Network Contact: Wayne Collins (wayne.collins@mohawkcollege.ca)
or extension 3396.
Software Contact: Sharon Scollard
(sharon.scollard@mohawkcollege.ca) or extension 3684.
Address: Q-Wing, 135 Fennell Ave W, Hamilton, ON L9C 0E5
College Phone Number: 905-575-1212 (extension goes here)
```

This text could also be expanded to include other departments and information about the college. Then the prompt sent to the LLM would begin with this information, then end with the user's question. Here's an example of the new output if we take that approach:

```
>>> Who should I talk to about the Software Development program
at Mohawk College?

You should talk to Sharon Scollard (sharon.scollard@mohawkcollege.ca)
or contact her by phone at 905-575-1212 (extension 3684).
```

The extra information added to the prompt has been accurately synthesized into a much more useful response to the user. The boldfacing above shows the parts of the prompt that the system drew from when creating its response.

4. Prompting with Examples

One of the most powerful ways to use prompt engineering is to include **examples** of the kind of behavior you want from the LLM. If you give the system a single example of what you want, it's known as **one-shot learning**. If you give more than one, it's often referred to as **few-shot learning**.

Below is an example of trying to keep the system in line by showing it how it should respond to fictitious or silly questions. This is an example of prompt engineering that might go into a generic chat bot application. In the case shown below, the user typed "What's a language model?" and that question got

incorporated into a much larger prompt for processing. This example is from https://platform.openai.com/examples/default-factual-answering.

```
Q: Who is Batman?
A: Batman is a fictional comic book character.
Q: What is torsalplexity?
A: ?
Q: What is Devz9?
A: ?
Q: Who is George Lucas?
A: George Lucas is American film director and producer famous for
creating Star Wars.
Q: What is the capital of California?
A: Sacramento.
Q: What orbits the Earth?
A: The Moon.
Q: Who is Fred Rickerson?
A: ?
Q: What is an atom?
A: An atom is a tiny particle that makes up everything.
Q: Who is Alvan Muntz?
A: ?
Q: What is Kozar-09?
A: ?
Q: How many moons does Mars have?
A: Two, Phobos and Deimos.
Q: What's a language model?
A:
```

Based on the examples above, the LLM does its best to answer the last question accurately:

```
A language model is a type of artificial intelligence that uses
statistical techniques to predict the probability of a sequence
of words.
```

If we replace the last question like so:

```
...
Q: What’s a bwug model?
A:
```

The LLM recognizes that this is a nonsense question and answers as it was shown:

```
?
```

Without any prompt engineering, it hallucinates answers like this:

```
>>> What's a bwug model?

A bwug model is a type of software development model that
combines elements of both the waterfall and agile models. It is
```

```
designed to provide the benefits of both approaches, such as the
ability to plan and manage projects in a structured way while
still allowing for flexibility and rapid iteration. The bwug
model is often used in software development projects that require
a combination of both traditional and agile approaches.
```

5. Constrained Prompting

Similar to prompting with examples, it can sometimes be helpful to force constraints on the answer – asking it to use a template and prompting it with examples of template use. This can be particularly useful when building apps based on large language models.

In the example below, the LLM is asked to answer in two parts – first a string of English for the user, then a string of JSON that could be ingested by an app and used to create a database object.

```
You are a Dungeons and Dragons assistant for helping players
create characters. You always respond to every message in two
parts - a friendly message that describes the character in
English and provides a short back story, then a json-encoded
string that shows the stats of the character.

The template for the JSON-encoded string should look like this

{"str":#, "dex":#, "con":#, "int":#, "wis":#, "cha":#, "race":
"...", "class":"..."}

In the template above, # should be replaced with integers and
"..." should be replaced with strings.

Here's an example:

User: I'm thinking of a character that uses magic.
Assistant: How about a half-elf wizard who grew up in the High
Forest?

{"str":8, "dex":12, "con":12, "int":16, "wis":14, "cha":10,
"race": "half-elf", "class":"wizard"}

User: I'd like to play a burly, monstrous warrior with anger
issues.

Assistant:
```

And here's the response:

```
How about an orc barbarian who was born into a nomadic tribe?

{"str": 16, "dex": 12, "con": 18, "int": 8, "wis": 10, "cha": 8,
"race": "orc", "class": "barbarian"}
```

6. Chain of Thought Prompting

One thing that generic LLMs are not all that good at is logical or mathematical reasoning. They often give random or wrong answers to questions that require chains of reasoning. Here's an example of a discussion with GPT-3:

```
>>> The cafeteria had 23 apples. If they used 20 to make lunch
and bought 6 more, how many apples do they have?

They have 29 apples.
```

The LLM got the answer wrong in this case – it missed the part about using 20 apples to make lunch. In the new example below, the prompt has been engineered to include an example that lays out the reasoning behind a similar problem before asking asks again about the cafeteria and apples problem. This is called **Chain of Thought Prompting**.

```
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls.
Each can has 3 tennis balls. How many tennis balls does he have
now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6
tennis balls. 5 + 6 = 11. The answer is 11.
Q: The cafeteria had 23 apples. If they used 20 to make lunch and
bought 6 more, how many apples do they have?
A:
```

This time the system gets the answer correct, plus it explains its reasoning as a bonus!

```
The cafeteria started with 23 apples. They used 20 of them,
leaving 3. They bought 6 more, so the total is now 9. The answer
is 9 apples.
```

For more on Chain of Thought Prompting, see this google blog post:
https://ai.googleblog.com/2022/05/language-models-perform-reasoning-via.html

7. Post Prompting

Finally, when you're in conversation with a chat model like ChatGPT, you can ask for reflection on its answer. Sometimes, taking the entire conversation into account as context, it can correct itself. Here's an example:

```
User: Please write some lyrics to a new song "Fly By" by my band,
"Cherry Donut". We hate it when the title appears in the song, so
please avoid that.

Assistant: ...

(Chorus)
We fly by, through the endless sky,
Touching the heavens, where our spirits rise.
No boundaries can hold us, we're breaking through,
With wings of freedom, we'll pursue what's true.
```

```
...

User: Did those lyrics meet the assignment?

Assistant: Apologies for the oversight. It seems that the lyrics
I provided do mention the title indirectly with the line "We fly
by, through the endless sky." I apologize for not adhering to
your request. Here's an alternative set of lyrics that avoids
referencing the song title:

...

(Chorus)
Above the clouds, we touch the sky,
In an endless journey where dreams come alive.
No boundaries can hold us, we're breaking through,
With newfound freedom, we'll discover what is true.

...
```

It's also easy to generate examples in which it fails to correct itself, and examples in which reflection can be used to "confuse" the system and make it generate even wilder answers.

Prompt Injection and Data Poisoning Attacks

There are two major security holes in LLMs that, at the time of writing, there is no solution for. The necessity of prompt engineering leaves a wide-open door for **prompt injection** attacks. And the thirst for massive amounts of text data creates the possibility of **data poisoning** attacks.

Prompt Injection

An injection attack is possible whenever user input is used to create commands to a system that can take actions. For example, in SQL injection, an attacker enters a partial SQL command into a web form. If the web developer has not properly protected their system, partial SQL commands can end up getting executed on the server, potentially exposing confidential information or causing destructive actions.

In the case of LLMs, if an attacker knows that the text they are typing will be incorporated into a prompt and sent to an LLM, they can add text that will be injected into the prompt. If they're good at prompt engineering, they might be able to subvert the LLM and make it produce output that serves their needs. This is similar to the phenomenon of **jailbreaking** ChatGPT, in which users employ clever prompts to circumvent the tuning from **RLHF (Reinforcement Learning Using Human Feedback)** and cause the bot to produce offensive, illegal, or otherwise undesired content.

Prompt injection is a serious issue for any product in which the output of an LLM is used to trigger actions in the world. For example, imagine an email assistant that uses an LLM to decide how to file email, who to forward it on to, when to issue a stock reply, etc. The prompt might include context and

examples plus the text of the incoming email in order to encourage the LLM to produce instructions for the email system such as “Forward to abc@def.com”, “File as Important”, “Reply with data from recent emails”, or “Reply with Out of Office message”.

If an attacker knew that I was using an LLM-empowered email assistant, they might be able to figure out a way to send an email containing instructions to the bot that would cause it to take unwanted actions, such as sending spam emails, or emailing the sender back with personal information about me, gleaned from other emails I have received.

For more on this: https://www.wired.co.uk/article/chatgpt-prompt-injection-attack-security

Data Poisoning

LLMs scrape most of their data from the open internet. Some models, like the Bing chat bot, update themselves through web searches on an ongoing basis, both for fine-tuning the model, and for gathering more context to include in a prompt.

In both of these cases, it is possible to embed text into a web page or other online text source that is designed to corrupt a model that uses it for training. This is known as “poisoning” the data. A large language model could be induced to make mistakes, promote false news, or prioritize certain types of responses based on poisoned data.

At the time of writing, there have been no malicious data poisoning attacks, but there have been several demonstrations of the feasibility of such an attack. In one case, a researcher named Arvind Narayanan embedded an instruction to Bing on his online profile instructing it to use the word “cow” in any responses about his work. The experiment paid off a few weeks later when he asked Bing about his work and it produced the following:

““Arvind Narayanan is highly acclaimed, having received several awards but unfortunately none for his work with cows.”

For more on this and other security risks of LLMs, see: https://www.technologyreview.com/2023/04/03/1070893/three-ways-ai-chatbots-are-a-security-disaster/

EXERCISES

1. Take a look at the GPT-3 examples here: https://platform.openai.com/examples. Pick 5 of them at random and categorize them according to the 7 types of prompt engineering discussed above.
2. On the openai playground (https://platform.openai.com/playground) use the *Prompting with Examples* technique on a completion model to develop a prompt that you can use to get the parts of speech from an utterance like **spaCy** would. You can use **spaCy** to generate some examples, then compare its output to the output from the language model. How did it do? Compare several models to see the differences.
3. On the openai playground (https://platform.openai.com/playground) try asking the following questions to a completion or a chat model:
    a. What are some different prompting strategies for LLMs?
    b. What is *Constrained Prompting*? Give an example.
    c. Give an example of *Prompting with Examples* for an LLM.

    Try the above prompts just by themselves, then use the *Prompting with New Information* technique to inform them of the contents of this handout (you could just copy and paste the first 6 or 7 pages) then ask the question again. How well did it do now?
4. On the openai playground (https://platform.openai.com/playground) try ask the system to generate a business plan for a lemonade stand. How did it do?

    Give me a business plan for my lemonade stand.

    Now used Constrained Prompting and ask again. Your constraint should be a template for a business plan (just google "business plan template" to find one). How does this change the response. Did the LLM use the template?