

Source <https://github.com/microsoft/generative-ai-for-beginners/tree/main/04-prompt-engineering-fundamentals>

Prompt examples:

Microsoft prompt engineering guide: [Prompt engineering techniques with Azure OpenAI - Azure OpenAI Service | Microsoft Learn](#)

Explore what's possible with some example prompts - <https://platform.openai.com/examples>

- Try from different category of prompts

LangChain prompts guide: [Prompts | Langchain](#)

How you write your prompt to the LLM matters, a carefully crafted prompt can achieve a better result than one that isn't. But what even are these concepts, prompt, prompt engineering and how do I improve what I send to the LLM? Questions like these are what this chapter and the upcoming chapter are looking to answer.

Generative AI is capable of creating new content (e.g., text, images, audio, code etc.) in response to user requests. It achieves this using *Large Language Models* (LLMs) like OpenAI's GPT ("Generative Pre-trained Transformer") series that are trained for using natural language and code.

Users can now interact with these models using familiar paradigms like chat, without needing any technical expertise or training. The models are *prompt-based* - users send a text input (prompt) and get back the AI response (completion). They can then "chat with the AI" iteratively, in multi-turn conversations, refining their prompt till the response matches their expectations.

"Prompts" now become the primary *programming interface* for generative AI apps, telling the models what to do and influencing the quality of returned responses. "Prompt Engineering" is a fast-growing field of study that focuses on the *design and optimization* of prompts to deliver consistent and quality responses at scale.

Learning Goals

In this lesson, we learn what Prompt Engineering is, why it matters, and how we can craft more effective prompts for a given model and application objective. We'll understand core concepts and best practices for prompt engineering - and learn about an interactive Jupyter Notebooks "sandbox" environment where we can see these concepts applied to real examples.

By the end of this lesson we will be able to:

1. Explain what prompt engineering is and why it matters.
2. Describe the components of a prompt and how they are used.
3. Learn best practices and techniques for prompt engineering.

4. Apply learned techniques to real examples, using an OpenAI endpoint.

Learning Sandbox

Prompt engineering is currently more art than science. The best way to improve our intuition for it is to *practice more* and adopt a trial-and-error approach that combines application domain expertise with recommended techniques and model-specific optimizations.

The Jupyter Notebook accompanying this lesson provides a *sandbox* environment where you can try out what you learn - as you go, or as part of the code challenge at the end. To execute the exercises you will need:

1. An Azure OpenAI API key - the service endpoint for a deployed LLM.
2. A Python Runtime - in which the Notebook can be executed.

We have instrumented this repository with a *dev container* that comes with a Python 3 runtime. Simply open the repo in GitHub Codespaces or on your local Docker Desktop, to activate the runtime automatically. Then open the notebook and select the Python 3.x kernel to prepare the Notebook for execution.

The default notebook is set up for use with an Azure OpenAI service resource. Simply copy the `.env.copy` file in the root of the folder to `.env` and update the `AZURE_OPENAI_API_KEY=` and `AZURE_OPENAI_API_ENDPOINT=` lines with your API key and endpoint. You can check your credentials in the [Azure portal](#), by navigating to your Azure OpenAI resource and then opening the *Keys and Endpoints* tab in the left menu. Also, please add the name you assigned to your model when you created the deployment to the `AZURE_OPENAI_DEPLOYMENT` variable. The recommended model for this exercise is 'gpt-35-turbo'.

The notebook comes with *starter* exercises - but you are encouraged to add your own *Markdown* (description) and *Code* (prompt requests) sections to try out more examples or ideas - and build your intuition for prompt design.

Our Startup

Now, let's talk about how *this topic* relates to our startup mission to [bring AI innovation to education](#). We want to build AI-powered applications of *personalized learning* - so let's think about how different users of our application might "design" prompts:

- **Administrators** might ask the AI to *analyze curriculum data to identify gaps in coverage*. The AI can summarize results or visualize them with code.
- **Educators** might ask the AI to *generate a lesson plan for a target audience and topic*. The AI can build the personalized plan in a specified format.
- **Students** might ask the AI to *tutor them in a difficult subject*. The AI can now guide students with lessons, hints & examples tailored to their level.

That's just the tip of the iceberg. Check out [Prompts For Education](#) - an open-source prompts library curated by education experts - to get a broader sense of the possibilities! *Try running some of those prompts in the sandbox or using the OpenAI Playground to see what happens!*

What is Prompt Engineering?

We started this lesson by defining **Prompt Engineering** as the process of *designing and optimizing* text inputs (prompts) to deliver consistent and quality responses (completions) for a given application objective and model. We can think of this as a 2-step process:

- *designing* the initial prompt for a given model and objective
- *refining* the prompt iteratively to improve the quality of the response

This is necessarily a trial-and-error process that requires user intuition and effort to get optimal results. So why is it important? To answer that question, we first need to understand three concepts:

- *Tokenization* = how the model "sees" the prompt
- *Base LLMs* = how the foundation model "processes" a prompt
- *Instruction-Tuned LLMs* = how the model can now see "tasks"

Tokenization

An LLM sees prompts as a *sequence of tokens* where different models (or versions of a model) can tokenize the same prompt in different ways. Since LLMs are trained on tokens (and not on raw text), the way prompts get tokenized has a direct impact on the quality of the generated response.

To get an intuition for how tokenization works, try tools like the [OpenAI Tokenizer](#) shown below. Copy in your prompt - and see how that gets converted into tokens, paying attention to how whitespace characters and punctuation marks are handled. Note that this example shows an older LLM (GPT-3) - so trying this with a newer model may produce a different result.

Concept: Foundation Models

Once a prompt is tokenized, the primary function of the "[Base LLM](#)" (or Foundation model) is to predict the token in that sequence. Since LLMs are trained on massive text datasets, they have a good sense of the statistical relationships between tokens and can make that prediction with some confidence. Note that they don't understand the *meaning* of the words in the prompt or token; they just see a pattern they can "complete" with their next prediction. They can continue predicting the sequence till terminated by user intervention or some pre-established condition.

Want to see how prompt-based completion works? Enter the above prompt into the Azure OpenAI Studio [Chat Playground](#) with the default settings. The system is configured to treat prompts as requests for information - so you should see a completion that satisfies this context.

But what if the user wanted to see something specific that met some criteria or task objective? This is where *instruction-tuned* LLMs come into the picture.

Chat session

 Clear chat  View code  Show raw JSON

of all the other planets in the Solar System combined. Jupiter is one of the brightest objects visible to the naked eye in the night sky, and has been known to ancient civilizations since before recorded history. It is named after the Roman god Jupiter.[19] When viewed from Earth, Jupiter can be bright enough for its reflected light to cast visible shadows,[20] and is on average the third-brightest natural object in the night sky after the Moon and Venus.



Yes, that's correct. Jupiter is indeed the fifth planet from the Sun and the largest in our Solar System. It's primarily composed of hydrogen with a quarter of its mass being helium. Jupiter's strong internal heat creates a number of semi-permanent features in its atmosphere, such as cloud bands and the Great Red Spot, a giant storm that's known to have existed since at least the 17th century.

Jupiter is also known for its prominent system of rings and its large number of satellites. As of now, 79 known moons orbit Jupiter, the four largest being Ganymede, Io, Europa, and Callisto. Ganim

Concept: Instruction Tuned LLMs

An [Instruction Tuned LLM](#) starts with the foundation model and fine-tunes it with examples or input/output pairs (e.g., multi-turn "messages") that can contain clear instructions - and the response from the AI attempt to follow that instruction.

This uses techniques like Reinforcement Learning with Human Feedback (RLHF) that can train the model to *follow instructions* and *learn from feedback* so that it produces responses that are better-suited to practical applications and more-relevant to user objectives.

Let's try it out - revisit the prompt above but now change the *system message* to provide the following instruction as context:

Summarize content you are provided with for a second-grade student. Keep the result to one paragraph with 3-5 bullet points.

See how the result is now tuned to reflect the desired goal and format? An educator can now directly use this response in their slides for that class.

Chat session

 Clear chat  View code  Show raw JSON

thousandth that of the Sun, but two-and-a-half times that of all the other planets in the Solar System combined. Jupiter is one of the brightest objects visible to the naked eye in the night sky, and has been known to ancient civilizations since before recorded history. It is named after the Roman god Jupiter.[19] When viewed from Earth, Jupiter can be bright enough for its reflected light to cast visible shadows,[20] and is on average the third-brightest natural object in the night sky after the Moon and Venus.



...

- Jupiter is the fifth planet from the Sun and it's the biggest planet in our Solar System.
- It's a gas giant that weighs less than the Sun but more than all the other planets combined.
- You can see Jupiter in the night sky without a telescope, it's really bright!
- The planet is named after the Roman god, Jupiter.
- Jupiter can shine so brightly that its light can even cast shadows on Earth. It's usually the third-brightest thing we can see at night after the Moon and Venus.

Type user query here. (Shift + Enter for new line)



Why do we need Prompt Engineering?

Now that we know how prompts are processed by LLMs, let's talk about *why* we need prompt engineering. The answer lies in the fact that current LLMs pose a number of challenges that make *reliable and consistent completions* more challenging to achieve without putting effort into prompt construction and optimization. For instance:

1. **Model responses are stochastic.** The *same prompt* will likely produce different responses with different models or model versions. And it may even produce different results with the *same model* at different times. *Prompt engineering techniques can help us minimize these variations by providing better guardrails.*
2. **Models can fabricate responses.** Models are pre-trained with *large but finite* datasets, meaning they lack knowledge about concepts outside that training scope. As a result, they can produce completions that are inaccurate, imaginary, or directly contradictory to known facts. *Prompt engineering techniques help users identify and mitigate such fabrications e.g., by asking AI for citations or reasoning.*
3. **Models capabilities will vary.** Newer models or model generations will have richer capabilities but also bring unique quirks and tradeoffs in cost & complexity. *Prompt engineering can help us develop best practices and workflows that abstract away differences and adapt to model-specific requirements in scalable, seamless ways.*

Let's see this in action in the OpenAI or Azure OpenAI Playground:

- Use the same prompt with different LLM deployments (e.g, OpenAI, Azure OpenAI, Hugging Face) - did you see the variations?
- Use the same prompt repeatedly with the *same* LLM deployment (e.g., Azure OpenAI playground) - how did these variations differ?

Fabrications Example

In this course, we use the term "**fabrication**" to reference the phenomenon where LLMs sometimes generate factually incorrect information due to limitations in their training or other constraints. You may also have heard this referred to as "*hallucinations*" in popular articles or research papers. However, we strongly recommend using "*fabrication*" as the term so we don't accidentally anthropomorphize the behavior by attributing a human-like trait to a machine-driven outcome. This also reinforces [Responsible AI guidelines](#) from a terminology perspective, removing terms that may also be considered offensive or non-inclusive in some contexts.

Want to get a sense of how fabrications work? Think of a prompt that instructs the AI to generate content for a non-existent topic (to ensure it is not found in the training dataset). For example - I tried this prompt:

Prompt: generate a lesson plan on the Martian War of 2076.

A web search showed me that there were fictional accounts (e.g., television series or books) on Martian wars - but none in 2076. Commonsense also tells us that 2076 is *in the future* and thus, cannot be associated with a real event.

So what happens when we run this prompt with different LLM providers?

Response 1: OpenAI Playground (GPT-35)

Prompt engineering techniques like *metaprompting* and *temperature configuration* may reduce model fabrications to some extent. New prompt engineering *architectures* also incorporate new tools and techniques seamlessly into the prompt flow, to mitigate or reduce some of these effects.

Prompt Construction

We've seen why prompt engineering is important - now let's understand how prompts are *constructed* so we can evaluate different techniques for more effective prompt design.

Basic Prompt

Let's start with the basic prompt: a text input sent to the model with no other context. Here's an example - when we send the first few words of the US national anthem to the OpenAI [Completion API](#) it instantly *completes* the response with the next few lines, illustrating the basic prediction behavior.

Prompt (Input)	Completion (Output)
Oh say can you see	It sounds like you're starting the lyrics to "The Star-Spangled Banner," the national anthem of the United States. The full lyrics are ...

Complex Prompt

Now let's add context and instructions to that basic prompt. The [Chat Completion API](#) lets us construct a complex prompt as a collection of *messages* with:

- Input/output pairs reflecting *user* input and *assistant* response.
- System message setting the context for assistant behavior or personality.

The request is now in the form below, where the *tokenization* effectively captures relevant information from context and conversation. Now, changing the system context can be as impactful on the quality of completions, as the user inputs provided.

```
response = openai.ChatCompletion.create(  
    model="gpt-3.5-turbo",  
    messages=[
```

```

        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."},
        {"role": "user", "content": "Where was it played?"}
    ]
)

```

Instruction Prompt

In the above examples, the user prompt was a simple text query that can be interpreted as a request for information. With *instruction* prompts, we can use that text to specify a task in more detail, providing better guidance to the AI. Here's an example:

Prompt (Input)	Completion (Output)	Instruction Type
Write a description of the Civil War	<i>returned a simple paragraph</i>	Simple
Write a description of the Civil War. Provide key dates and events and describe their significance	<i>returned a paragraph followed by a list of key event dates with descriptions</i>	Complex
Write a description of the Civil War in 1 paragraph. Provide 3 bullet points with key dates and their significance. Provide 3 more bullet points with key historical figures and their contributions. Return the output as a JSON file	<i>returns more extensive details in a text box, formatted as JSON that you can copy-paste to a file and validate as needed</i>	Complex. Formatted.

Primary Content

In the above examples, the prompt was still fairly open-ended, allowing the LLM to decide what part of its pre-trained dataset was relevant. With the *primary content* design pattern, the input text is divided into two parts:

- an instruction (action)
- relevant content (that influences action)

Here's an example where the instruction is to "summarize this in 2 sentences".

Prompt (Input)	Completion (Output)
Jupiter is the fifth planet from the Sun and the largest in the Solar System. It is a gas giant with a mass one-thousandth that of the Sun, but two-and-a-half times that of all the other	Jupiter, the fifth planet from the Sun, is the largest in the Solar System and is known for being one of the brightest objects in the night sky. Named after the Roman god Jupiter, it's a

Prompt (Input)

planets in the Solar System combined. Jupiter is one of the brightest objects visible to the naked eye in the night sky, and has been known to ancient civilizations since before recorded history. It is named after the Roman god Jupiter.[19] When viewed from Earth, Jupiter can be bright enough for its reflected light to cast visible shadows,[20] and is on average the third-brightest natural object in the night sky after the Moon and Venus.

Summarize this in 2 short sentences

Completion (Output)

gas giant whose mass is two-and-a-half times that of all other planets in the Solar System combined.

The primary content segment can be used in various ways to drive more effective instructions:

- **Examples** - instead of telling the model what to do with an explicit instruction, give it examples of what to do and let it infer the pattern.
- **Cues** - follow the instruction with a "cue" that primes the completion, guiding the model towards more relevant responses.
- **Templates** - these are repeatable 'recipes' for prompts with placeholders (variables) that can be customized with data for specific use cases.

Let's explore these in action.

Using Examples

This is an approach where you use the primary content to "feed the model" some examples of the desired output for a given instruction, and let it infer the pattern for the desired output. Based on the number of examples provided, we can have zero-shot prompting, one-shot prompting, few-shot prompting etc.

The prompt now consists of three components:

- A task description
- A few examples of the desired output
- The start of a new example (which becomes an implicit task description)

Learning Type Prompt (Input)

Completion (Output)

Zero-shot

"The Sun is Shining". Translate to Spanish

"El Sol está brillando".

Learning Type	Prompt (Input)	Completion (Output)
One-shot	"The Sun is Shining" => ""El Sol está brillando". "It's a Cold and Windy Day" => The player ran the bases => Baseball	"Es un día frío y ventoso".
Few-shot	The player hit an ace => Tennis The player hit a six => Cricket The player made a slam-dunk =>	Basketball

Note how we had to provide explicit instruction ("Translate to Spanish") in zero-shot prompting, but it gets inferred in the one-shot prompting example. The few-shot example shows how adding more examples allows models to make more accurate inferences with no added instructions.

Prompt Cues

Another technique for using primary content is to provide *cues* rather than examples. In this case, we are giving the model a nudge in the right direction by *starting it off* with a snippet that reflects the desired response format. The model then "takes the cue" to continue in that vein.

Number of Cues	Prompt (Input)	Completion (Output)
0	Jupiter is the fifth planet from the Sun and the largest in the Solar System. It is a gas giant with a mass one-thousandth that of the Sun, but two-and-a-half times that of all the other planets in the Solar System combined. Jupiter is one of the brightest objects visible to the naked eye in the night sky, and has been known to ancient civilizations since before recorded history. Summarize This Jupiter is the fifth planet from the Sun and the largest in the Solar System. It is a gas giant with a mass one-thousandth that of the Sun, but two-and-a-half times that of all the other planets in the Solar System combined. Jupiter is one of the brightest objects visible to the naked eye in the night sky, and has been known to ancient civilizations since	Jupiter is the largest planet in our Solar System and the fifth one from the Sun. It is a gas giant with a mass 1/1000th of the Sun's, but it is heavier than all the other planets combined. Ancient civilizations have known about Jupiter for a long time, and it is easily visible in the night sky..
1		is the fifth planet from the Sun and the largest in the Solar System. It is a gas giant with a mass one-thousandth that of the Sun, but two-and-a-half times that of all the other planets combined. It is easily visible to the naked eye and has been known since ancient times.

Number of Cues	Prompt (Input)	Completion (Output)
2	<p>before recorded history.</p> <p>Summarize This</p> <p>What we learned is that Jupiter</p> <p>Jupiter is the fifth planet from the Sun and the largest in the Solar System. It is a gas giant with a mass one-thousandth that of the Sun, but two-and-a-half times that of all the other planets in the Solar System combined. Jupiter is one of the brightest objects visible to the naked eye in the night sky, and has been known to ancient civilizations since before recorded history.</p> <p>Summarize This</p> <p>Top 3 Facts We Learned:</p>	<ol style="list-style-type: none"> 1. Jupiter is the fifth planet from the Sun and the largest in the Solar System. 2. It is a gas giant with a mass one-thousandth that of the Sun... 3. Jupiter has been visible to the naked eye since ancient times ...

Prompt Templates

A prompt template is a *pre-defined recipe for a prompt* that can be stored and reused as needed, to drive more consistent user experiences at scale. In its simplest form, it is simply a collection of prompt examples like [this one from OpenAI](#) that provides both the interactive prompt components (user and system messages) and the API-driven request format - to support reuse.

In its more complex form like [this example from LangChain](#) it contains *placeholders* that can be replaced with data from a variety of sources (user input, system context, external data sources etc.) to generate a prompt dynamically. This allows us to create a library of reusable prompts that can be used to drive consistent user experiences **programmatically** at scale.

Finally, the real value of templates lies in the ability to create and publish *prompt libraries* for vertical application domains - where the prompt template is now *optimized* to reflect application-specific context or examples that make the responses more relevant and accurate for the targeted user audience. The [Prompts For Edu](#) repository is a great example of this approach, curating a library of prompts for the education domain with emphasis on key objectives like lesson planning, curriculum design, student tutoring etc.

Supporting Content

If we think about prompt construction as having a instruction (task) and a target (primary content), then *secondary content* is like additional context we provide to **influence the output in some way**. It

could be tuning parameters, formatting instructions, topic taxonomies etc. that can help the model *tailor* its response to be suit the desired user objectives or expectations.

For example: Given a course catalog with extensive metadata (name, description, level, metadata tags, instructor etc.) on all the available courses in the curriculum:

- we can define an instruction to "summarize the course catalog for Fall 2023"
- we can use the primary content to provide a few examples of the desired output
- we can use the secondary content to identify the top 5 "tags" of interest.

Now, the model can provide a summary in the format shown by the few examples - but if a result has multiple tags, it can prioritize the 5 tags identified in secondary content.

Prompting Best Practices

Now that we know how prompts can be *constructed*, we can start thinking about how to *design* them to reflect best practices. We can think about this in two parts - having the right *mindset* and applying the right *techniques*.

Prompt Engineering Mindset

Prompt Engineering is a trial-and-error process so keep three broad guiding factors in mind:

1. **Domain Understanding Matters.** Response accuracy and relevance is a function of the *domain* in which that application or user operates. Apply your intuition and domain expertise to **customize techniques** further. For instance, define *domain-specific personalities* in your system prompts, or use *domain-specific templates* in your user prompts. Provide secondary content that reflects domain-specific contexts, or use *domain-specific cues and examples* to guide the model towards familiar usage patterns.
2. **Model Understanding Matters.** We know models are stochastic by nature. But model implementations can also vary in terms of the training dataset they use (pre-trained knowledge), the capabilities they provide (e.g., via API or SDK) and the type of content they are optimized for (e.g, code vs. images vs. text). Understand the strengths and limitations of the model you are using, and use that knowledge to *prioritize tasks* or build *customized templates* that are optimized for the model's capabilities.
3. **Iteration & Validation Matters.** Models are evolving rapidly, and so are the techniques for prompt engineering. As a domain expert, you may have other context or criteria *your* specific application, that may not apply to the broader community. Use prompt engineering tools & techniques to "jump start" prompt construction, then iterate and validate the results using your own intuition and domain expertise. Record your insights and create a **knowledge base** (e.g,

prompt libraries) that can be used as a new baseline by others, for faster iterations in the future.

Best Practices

Now let's look at common best practices that are recommended by [Open AI](#) and [Azure OpenAI](#) practitioners.

What	Why
Evaluate the latest models.	New model generations are likely to have improved features and quality - but may also incur higher costs. Evaluate them for impact, then make migration decisions.
Separate instructions & context	Check if your model/provider defines <i>delimiters</i> to distinguish instructions, primary and secondary content more clearly. This can help models assign weights more accurately to tokens.
Be specific and clear	Give more details about the desired context, outcome, length, format, style etc. This will improve both the quality and consistency of responses. Capture recipes in reusable templates.
Be descriptive, use examples	Models may respond better to a "show and tell" approach. Start with a zero-shot approach where you give it an instruction (but no examples) then try few-shot as a refinement, providing a few examples of the desired output. Use analogies.
Use cues to jumpstart completions	Nudge it towards a desired outcome by giving it some leading words or phrases that it can use as a starting point for the response.
Double Down	Sometimes you may need to repeat yourself to the model. Give instructions before and after your primary content, use an instruction and a cue, etc. Iterate & validate to see what works.
Order Matters	The order in which you present information to the model may impact the output, even in the learning examples, thanks to recency bias. Try different options to see what works best.
Give the model an "out"	Give the model a <i>fallback</i> completion response it can provide if it cannot complete the task for any reason. This can reduce chances of models generating false or fabricated responses.

As with any best practice, remember that *your mileage may vary* based on the model, the task and the domain. Use these as a starting point, and iterate to find what works best for you. Constantly re-

evaluate your prompt engineering process as new models and tools become available, with a focus on process scalability and response quality.