Prompt Engineering Fundamentals

Introduction

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 until 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.

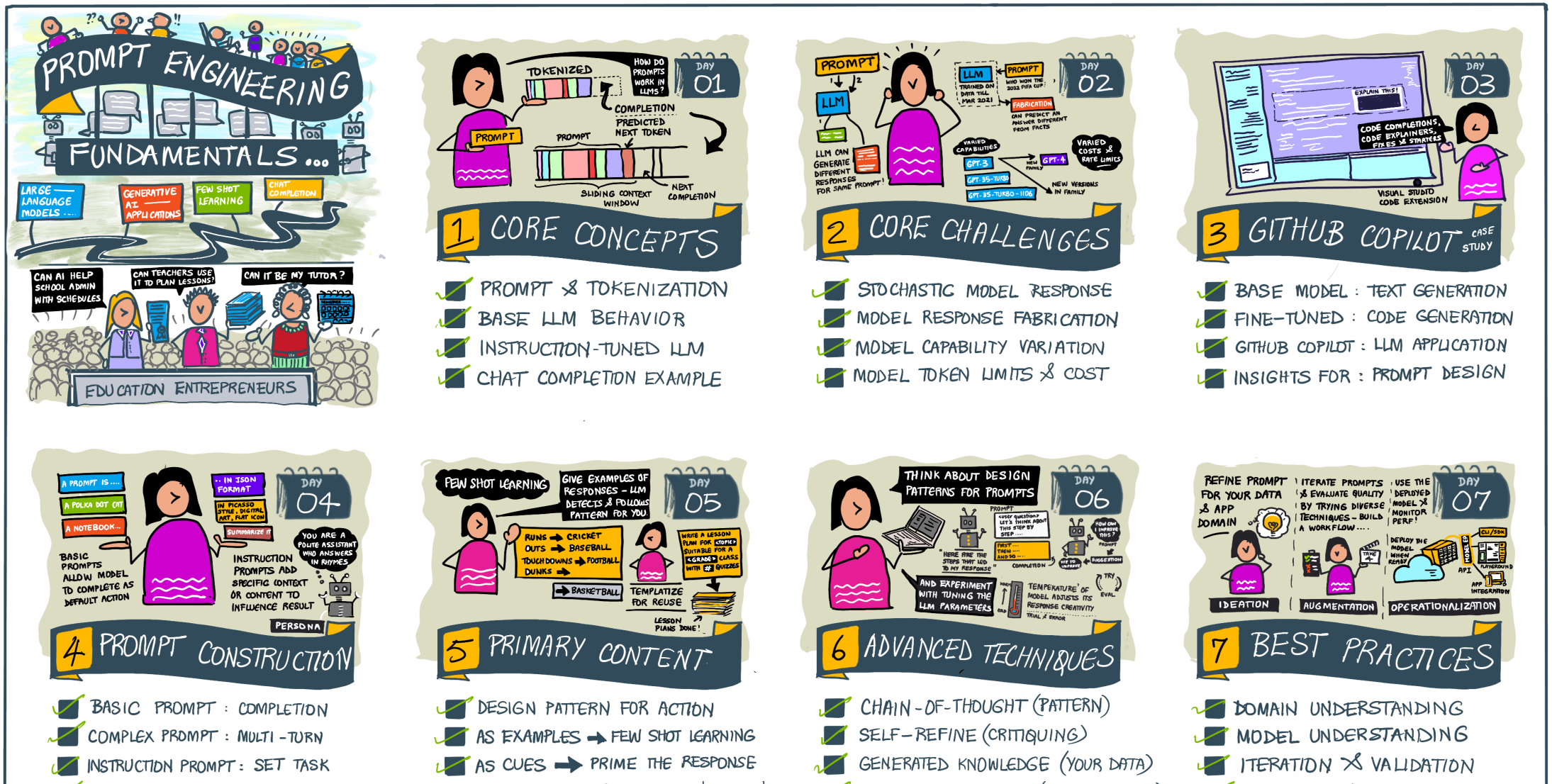
By the end of this lesson we will be able to:

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.

Key Terms

- Prompt Engineering: The practice of designing and refining inputs to guide AI models toward producing desired outputs.
- Tokenization: The process of converting text into smaller units, called tokens, that a model can understand and process.
- Instruction-Tuned LLMs: Large Language Models (LLMs) that have been fine-tuned with specific instructions to improve their response accuracy and relevance.

Illustrated Guide



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.

What is Prompt Engineering?

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.

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.

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.
2. **Models can fabricate responses.** Models are pre-trained with *large but finite* datasets, meaning they lack knowledge about concepts outside that training scope.
3. **Models capabilities will vary.** Newer models or model generations will have richer capabilities but also bring unique quirks and tradeoffs in cost & complexity.