Prompt Engineering Quick Start (SDK)

This quick start will walk through how to create, test, and iterate on prompts using the SDK. In this tutorial we will use OpenAI, but you can use whichever LLM you want.

> **QUICKSTART**
>
> This tutorial uses the SDK for prompt engineering, if you are interested in using the UI instead, read this guide.

1. Setup

First, install the required packages:

Python TypeScript

```
yarn add langsmith @langchain/core langchain openai
```

Next, make sure you have signed up for a LangSmith account, then create and set your API key. You will also want to sign up for an OpenAI API key to run the code in this tutorial.

```
LANGSMITH_API_KEY = '<your_api_key>'
OPENAI_API_KEY = '<your_api_key>'
```

2. Create a prompt

To create a prompt in LangSmith, define the list of messages you want in your prompt and then wrap them using the `ChatPromptTemplate` function (Python) or TypeScript function. Then all you

have to do is call `push_prompt` (Python) or `pushPrompt` (TypeScript) to send your prompt to LangSmith!

Python TypeScript

```typescript
import { Client } from "langsmith";
import { ChatPromptTemplate } from "@langchain/core/prompts";

// Connect to the LangSmith client
const client = new Client();

// Define the prompt
const prompt = ChatPromptTemplate.fromMessages([
  ["system", "You are a helpful chatbot."],
  ["user", "{question}"],
]);

// Push the prompt
await client.pushPrompt("my-prompt", {
  object: prompt,
});
```

3. Test a prompt

To test a prompt, you need to pull the prompt, invoke it with the input values you want to test and then call the model with those input values. your LLM or application expects.

Python TypeScript

```typescript
import { OpenAI } from "openai";
import { pull } from "langchain/hub";
import { convertPromptToOpenAI } from "@langchain/openai";

// Connect to LangSmith and OpenAI
const oaiClient = new OpenAI();

// Pull the prompt to use
// You can also specify a specific commit by passing the commit hash "my-prompt:<commit-hash>"
const prompt = await pull("my-prompt");
```

```
// Format the prompt with the question
const formattedPrompt = await prompt.invoke({
  question: "What is the color of the sky?",
});

// Test the prompt
const response = await oaiClient.chat.completions.create({
  model: "gpt-4o",
  messages: convertPromptToOpenAI(formattedPrompt).messages,
});
```

4. Iterate on a prompt

LangSmith makes it easy to iterate on prompts with your entire team. Members of your workspace can select a prompt to iterate on, and once they are happy with their changes, they can simply save it as a new commit.

To improve your prompts:

- We recommend referencing the documentation provided by your model provider for best practices in prompt creation, such as Best practices for prompt engineering with the OpenAI API and Gemini's Introduction to prompt design.
- To help with iterating on your prompts in LangSmith, we've created Prompt Canvas — an interactive tool to build and optimize your prompts. Learn about how to use Prompt Canvas.

To add a new commit to a prompt, you can use the same `push_prompt` (Python) or `pushPrompt` (TypeScript) methods as when you first created the prompt.

Python **TypeScript**

```
import { Client } from "langsmith";
import { ChatPromptTemplate } from "@langchain/core/prompts";

// Connect to the LangSmith client
const client = new Client();

// Define the prompt
const newPrompt = ChatPromptTemplate.fromMessages([
  ["system", "You are a helpful chatbot. Speak in Spanish."],
```

```
  ["user", "{question}"],
]);

// Push the updated prompt making sure to use the correct prompt name
// Tags can help you remember specific versions in your commit history
await client.pushPrompt("my-prompt", {
  object: newPrompt,
  tags: ["Spanish"],
});
```

5. Next steps

- Learn more about how to store and manage prompts using the Prompt Hub in these how-to guides
- Learn more about how to use the playground for prompt engineering in these how-to guides

Was this page helpful?

You can leave detailed feedback on GitHub.