Docs

# Prompt engineering techniques

Prompt engineering involves crafting inputs for language models to elicit specific behaviors or responses. It plays a pivotal role in directing the sequence of actions and outcomes generated by the model.

This page lists a few techniques that can be used to guide the model in performing specific tasks or sequences of actions.

## Input delimiters

Input delimiters play a vital role in structuring the instructions for the language model. They mark the boundaries between different sections or segments of the prompt, guiding the model in processing distinct pieces of information.

For instance, hyphens ( `---` ) or similar markers delineate various components within the prompt, aiding the model in discerning user input, intermediate steps, and the final actions to be executed.

```
Objective: Write a summary of the text delimited by ---

---
{text}
---
```

Text is the user text (method parameter in an AI Service). Input delimiters also avoids prompt injections.

## Few-Shot techniques

Few-shot learning techniques are instrumental in training models to comprehend and generate output based on a limited number of examples or demonstrations. This approach involves presenting the model with a handful of examples, allowing it to generalize and adapt to similar scenarios.

By strategically crafting a few-shot prompt that includes diverse yet concise samples of the desired actions, one can guide the model to understand and execute tasks based on the provided exemplars.

Using a few-shot technique guides a model to understand sentiment analysis based on minimal examples:

```
Provide sentiment labels for the statements delimited by ---
The response must be either 'Positive', 'Neutral', or 'Negative'.

Here are a few examples:
- 'I love this product' - Positive
- 'Not bad, but could be better' - Neutral
- 'I'm thoroughly disappointed' - Negative

---
{text}
---
```

In this example, the model is presented with a few labeled statements to learn sentiment analysis, followed by a new `{text}` to analyze.

## Passing a list of actions

To instruct a language model to perform a sequence of actions, a carefully structured prompt containing a list of actions is essential. This involves delineating each action along with its associated tool and parameters within the prompt.

For example, a structured list could entail a set of instructions such as:

```
Action 1: Tool A with parameters X, Y
Action 2: Tool B with parameters Z
Action 3: Tool C with parameter W
```

Guiding the model through a well-defined sequence of actions within the prompt facilitates the execution of a chain of tasks or tools.

For example:

```
Objective: Perform a series of data processing tasks:

Task 1: CleanData with parameters {method: 'standard', missing_values: 'drop'}
Task 2: ScaleData with parameters {method: 'min-max', feature_range: (0, 1)}
Task 3: ApplyModel with parameters {model: 'random-forest', features: 'all'}"
```

Here, the prompt instructs the model to execute a sequence of data processing task involving data

cleaning, scaling, and model application. Each task is defined by a tool and its associated parameters.

## Asking for a specific JSON format

Prompt engineering can also involve requesting specific structured data formats, such as JSON, from the language model. In this scenario, specifying the expected JSON keys aids the model in generating the desired output structure:

```
Objective: Generate a JSON response with specific keys:

JSON Structure:
{
    'name': 'String',
    'age': 'Number',
    'email': 'String',
    'address': {
        'street': 'String',
        'city': 'String',
        'zip_code': 'String'
    }
}

Provided Data: John Doe, 30, john@example.com, 123 Main St, New York, 10001
```

In this example, the prompt specifies the expected structure for the JSON response, comprising keys like `name`, `age`, `email`, and `address`, with their respective value types. The `Provided Data` segment serves as the input to be formatted into the requested JSON structure.

Being able to describe the JSON output is essential when an AI method returns an object.

Even if a JSON format is not directly provided by the developer in the prompt, the LangChain4j will automatically add it, and the Quarkus LangChain4j extension will use the JSON structure to create an instance of the return type.

In this case, when working with JSON formats, it's good to specify this fact in a prompt and always ask for a valid and unwrapped JSON, for example:

```
You must respond in a valid JSON format.
You must not wrap JSON response in backticks, markdown, or in any other way, but
return it as plain text.
```

## Control tokens and prefixes

Control tokens or prefixes within prompts guide the model's behavior. For example:

```
Answer: What is the capital of France?
```

Here, the prefix `Answer:` directs the model to provide the solution to the question that follows.

## Domain-specific language and vocabulary

Using domain-specific language or vocabulary in prompts enhances the model's understanding within a specific domain. For instance:

```
Medicine: List the symptoms and treatment for strep throat.
```

Employing domain-specific terms like `Medicine` ensures the model addresses medical-related queries specifically. These techniques offer additional methods to direct language models effectively, providing further insight and control over their output.

## Giving a role to the AI

In relation to the previous technique, giving a role to the AI is a technique that involves assigning a specific role to the language model, such as a teacher, student, or assistant. This is generally done in the system message to guide the model's behavior:

```
@SystemMessage("""
    You are a bank account fraud detection AI. You have to detect frauds in
transactions.
""")
```

Copyright (C) 2020-2024 Red Hat and individual contributors to Quarkiverse.