**Prompting Strategy**

In this implementation, I used Google’s Gemini model (version 1.5-flash) to generate detailed testing instructions for digital product features based on screenshots. The prompting strategy is divided into two phases:

**System Prompt Setup**:

I provided an initial **system prompt** to the LLM instructing it on how to structure the output. This included the necessary details for each test case (Test Case ID, Description, Pre-conditions, Test Data, Testing Steps, Expected Result).

*[system\_prompt = """ Generate detailed testing instructions for the provided screenshots. Each test case should include: -Test Case ID:Assigning a unique identifier to the test case. - Description: What the test case is about. - Pre-conditions: What needs to be set up before testing. - Test Data:Necessary test data required to perform the test. - Testing Steps: Step-by-step instructions to perform the test. - Expected Result: What the correct result looks like. Be elaborate. """]*

**Contextual Prompting**:

When screenshots and an optional context were provided,the tool builds utilised a **contextual prompt** by embedding the uploaded images and any user-provided context into the prompt.

*[prompt = f"Generate test cases for these screenshots based on the following context: {context}"]*

The response was generated and displayed **iteratively**. This helped to simulate the response in real-time, and enhancing user experience by progressively showing the instructions as they are formed.

**Code Implementation**

1. Model Initialization

* The code initializes the Google Gemini multimodal LLM using the provided API key.
* A chat session is created (chat = model.start\_chat(history=[])) to interact with the model and provide instructions.

2. Processing Screenshots Function

* process\_screenshots:
  + Takes in uploaded images and optional text context.
  + It uses a system prompt to instruct the LLM to generate detailed testing instructions for the images.
  + A prompt is formed based on the images and context, asking the model to generate test cases.
  + The response is fetched and displayed iteratively (character-by-character) to simulate real-time output using time.sleep().

3. Gradio Front-End

* gr.Blocks(): Builds a simple user interface using Gradio.
* Inputs:
  + A textbox for optional context and a file uploader for multiple screenshots.
* Outputs:
  + A textbox to display the generated testing instructions.
* Button:
  + When clicked, it triggers the process\_screenshots function to generate instructions based on the inputs (images and context).

4. Launching the Interface

* demo.launch(debug=True) launches the Gradio app, allowing users to upload images, provide context, and view generated test cases interactively.