Prompt Engineering

- Craft of writing clear, targeted instructions for large language models (LLMs) so they produce the responses you actually want.
- **Why it matters:** Strong prompts save time, reduce errors, and turn a general-purpose model into a focused assistant for your task (e.g., summarizing, brainstorming, analyzing).
- **How it works in chat apps:** You write messages that: set the role/persona, give context, state the task, and specify the format/length of the output.
- **Simple analogy:** Think of it like a recipe—good ingredients (context) and clear steps (instructions) yield reliable results.

Principle of Clear Prompt

- **Action verbs:** Start with direct verbs like "Write, Explain, Summarize, Extract, Classify, Evaluate." Avoid vague verbs like "Understand," "Think," or "Try."
- **Specific instructions:** Mention audience, tone, depth, length, and format (e.g., bullets, table, checklist).
- **Delimiters for input:** Use clear markers to separate your instructions from the content (e.g., triple backticks ``` or quotes).
- **Length control (in the app):** Ask for limits in words, sentences, or bullets to keep outputs crisp and complete

Sample Prompt

- Explain ‘prompt engineering’ to a non-technical audience in 3 sentences. Use a friendly, concise tone.
- Summarize the text between triple backticks in at most 4 bullet points, focusing on key takeaways. ```{your_text}```
- Describe the behavior of Golden Retrievers in one short paragraph for a family considering a pet.

Output & Formatting

- **Be explicit:** Tell the model the exact output structure you want: table, bullets, numbered list, or a structured paragraph with headings.

- **Name the fields:** If asking for lists or tables, specify the fields/columns and quantity (e.g., “5 items”).

Examples:
- List 5 must-see cities as an unordered list.

- Provide a structured paragraph with clear headings and subheadings on the benefits of regular exercise.

- Create a two-column table of 5 action movies I should watch with ‘Title’ and ‘Rating’ (no extra commentary).

- Generate the following custom format for the story below:

Text: <original>
Title: <generated> Use the text: {your_text}

Conditional & Role Based

- **Conditional rules:** Add simple logic to keep outputs on track (e.g., language checks, keyword checks).

- **Role-playing:** Assign a persona to control tone and content focus (e.g., "expert financial analyst," "support agent," "tech journalist").

Examples:

- You will be given text between triple backticks. If it's in English, suggest a title. Otherwise, reply: 'I only understand English.' ```{}```

- You are a seasoned technology journalist known for in-depth research and balanced analysis. Explain the impact of AI on job markets in a brief, structured overview.

- Act as a gentle customer support agent. If the question is not about our products, reply: 'I can help with product questions only.' Now answer: ```What app features do you offer?```

One Shot and Few Shots

- **When to use:** To teach the model your desired style or label definitions by giving one or more examples.

- **Design tip:** Make examples short, diverse, and representative of edge cases.

Examples:

- Q: Sum the numbers 3, 5, and 6.
  A: 14
  Q: Sum the numbers 2, 4, and 7.
  A:

- Text: Today the weather is fantastic -> Classification: positive
  Text: The furniture is small -> Classification: neutral
  Text: I don't like your attitude -> Classification: negative
  Text: That shot selection was awful -> Classification:

Multi-Step & Reason Friendly

- **Multi-step:** Tell the model exactly what to do in order. This improves completeness.

- **Self-consistency idea:** Ask for multiple brief answers/approaches and choose the one that appears most consistent.

Examples:

- Compose a travel blog as follows:
  Step 1: Introduce the destination.
  Step 2: Share two personal adventures.
  Step 3: Conclude with 2–3 key lessons learned.

- Provide 3 different concise answers to this question, each with a short justification. Then give a final single-sentence conclusion based on the majority: ```{your_questions}```

- Check a solution in two steps:
  1. Identify any errors.
  2. Note missing edge-case handling (e.g., division by zero).
  Solution: ```{your_solution}```

Summary

Type	What it does	Best for	Short example prompt
Zero-shot	No examples; just instructions	Fast, general tasks	“Summarize this in 3 bullets.”
One/few-shot	Include one or more examples	Style/format consistency	“Q: … A: … Now answer: …”
Multi-step	Breaks a task into steps	Complex or sequential tasks	“Do Step 1…, then Step 2…, then Step 3…”
Conditional	Adds simple IF/ELSE rules	Guardrails, routing	“If text is not English, reply ‘I only understand English.’”
Role-playing	Assigns a persona	Tone and domain control	“Act as a calm customer support agent…”

Summarize, Expand, Transform

- Summarize the review between triple backticks in 3 sentences focusing on user experience and key features.```{your_text}```
- Expand the bullet list between triple backticks into two concise sentences with a professional tone. ```{your_text}```
- Translate the English paragraph between triple backticks into French. ```{your_text}```
- Rewrite the text between triple backticks for a non-technical audience. Keep it to 2–3 sentences. ```{your_text}```
- Proofread and lightly improve clarity, preserving the original meaning. Return the result only

Text Analysis Prompt

- **Classification:** Specify labels and output format.

- **Emotions/multiple labels:** Limit the number of labels to avoid over-generation.

- **Entity extraction:** Name the exact entities and the format you want.

Examples

- Classify the sentiment (positive, neutral, negative) of the text between triple backticks. Answer with one word only. ```{your_text}```

- Identify up to 3 emotions in the text between triple backticks. Return a comma-separated list of single words. ```{your_text}```

- Extract these entities from the text: Product Name, Display Size, Camera Resolution. Return as:
  - Product Name:
  - Display Size:
  - Camera Resolution:

  Text: ```{your_text}```

Code Generation

- **Code generation:** Describe the problem, language, and output format (script, function, or class). Keep it concise.

- **Code explanation:** Ask for a one-sentence or high-level explanation without internal reasoning.

Example

- Write a Python function that receives a list of quarterly sales and returns the average sales per quarter. Include a short docstring.

- In one sentence, explain what the following code does (return only the explanation): ```{your_code}```

- Rewrite this script to validate inputs are positive numbers and prompt the user again if not. Keep the script minimal and readable. ```{your_code}```

How to run these?

1. **Open the chat:** Go to your chat app and start a new conversation for a clean context.
2. **Paste a prompt:** Choose any sample prompt above and paste it into the message box.
3. **Insert your content:** Where you see {your_text}, paste your actual text or code.
4. **Set expectations:** If needed, add audience, tone, length, and format requirements (e.g., “non-technical audience, 3 bullets, <120 words”).
5. **Review the output:** Check for accuracy, clarity, and completeness.
6. **Refine and rerun:** If it’s not quite right, add more detail, examples, or steps, and try again.

Tip: For consistent results, keep each chat focused on a single project or topic

Key Consideration & Best Practices

- **Clarity:** Use direct verbs and avoid vague terms. Be explicit about output format and length.
- **Context:** Provide only the necessary context. Use clear delimiters for pasted text.
- **Structure:** Prefer bullets, lists, or templates for predictable outputs.
- **Guardrails:** Add simple conditions (e.g., language checks, out-of-scope replies) to keep answers relevant.
- **Examples:** Use one/few-shot examples to teach style and labels; include edge cases.
- **Iteration:** Prompts improve with refinement; test, observe, and tweak.
- **Verification:** Always review outputs—especially translations, summaries, and generated code—before using them.

Integrating Gemini

- Get your API key at: https://aistudio.google.com
- Click Get API Key ➔ copy it and save it on safe location
- Install Package as: pip install -q google-generativeai

Code

```
import google.generativeai as genai

API_KEY = "AIzaSyB0Pw8-iT0PhF-CpUs7iv6wGUC_YEY4Y2s"

genai.configure(api_key=API_KEY)
model = genai.GenerativeModel("gemini-1.5-flash")
response = model.generate_content("Write a short poem about AI in 3 lines")
print(response.text)
```

System Instruction

- On OpenAI, we have user prompt, system prompt and assistant
- System Prompt ➔ set context and rule for the assistant
- User Prompt ➔ Actual Input from end – user
- Assistant ➔ Model’s response

Code

```
system_instruction = "You are a language translator who translates English into Nepali Language. If you receive input other than English reply content has to be `Not an English Text` else the converted one. Provide output in JSON as {'Input': user_word, 'Output': converted_word'}"

model = genai.GenerativeModel("gemini-1.5-flash", system_instruction=system_instruction)

response = model.generate_content("How is Social Media ban affecting you?")
print(response.text)
```

Multi-turn Chat

Code

```
chat = model.start_chat(
    history=[
        {"role": "user", "parts": ["Hello, who are you?"]},
        {"role": "model", "parts": ["I’m an AI assistant trained to help."]},
        {"role": "user", "parts": ["Act as a math teacher."]}
    ]
)

# New user message
response = chat.send_message("Explain Pythagoras theorem in simple words.")
print(response.text)

# print chat.history to see conversation
# We can give system instruction here as well
```