

Optimizing Prompts for Best Results

Now that you've built a chatbot with memory and connected it to real-world data via RAG, and tuned it via **system prompts** to be a professional, AI-assistant, it's time to expand our horizons on prompts - it's not just system prompts and queries. Multiple types of prompts are used in an LLM application to help guide output to achieve the expected outcome.

While the **system prompt** defines the personality of your LLM, there are more questions that you'll need to answer:

- How does your RAG-enhanced LLM know how to use the contexts retrieved from your knowledgebase?
- How can you fine tune your prompts to achieve maximum benefit?

The story so far...

Users love **TaskFriend**! However, lately they've noticed that the output isn't consistent. **TaskFriend** may stray off-topic or format responses in a way that's not aligned to what they were expecting. In short - **TaskFriend** has no well-defined input on how it should respond.

Goals

- Understand how prompt templates guide LLMs to use retrieved context in RAG systems
- Learn to design effective system prompts using the Role-Goal-Context-Audience-Format-Guardrails framework
- Apply advanced prompt engineering techniques: role/persona prompting, few-shot learning, and chain-of-thought reasoning
- Use meta-prompting to automate prompt improvement
- Implement automated evaluation and grading of LLM outputs using LLM-as-a-judge techniques

Initializing the environment

Setting up the API key

Before we start work on in any notebook, we'll need to load the [API key for Model Studio](#). This ensures that we can call APIs of Qwen models we'll be using throughout this course.

If you're unsure about how to find your **Model Studio** API key, refer to the [00 Setting Up the Environment](#) file.

```
# Load Model Studio API key
import os
from config.load_key import load_key
load_key(
    confirmation=False
)
```

Detected existing API key: sk-3...fdd8
Auto-confirmation enabled. Using existing API key.

Setting up the LLM and embedding model

We set up Alibaba Cloud's `qwen-plus` as the LLM and DashScope's `text-embedding-v3` embedding model.

For this lesson, we'll be using `OpenAILike` instead of `OpenAI`, which we were using before this.

`OpenAILike` is a **LlamaIndex-specific wrapper** designed for OpenAI-compatible models, including:

- Model Studio
- Dashscope
- vLLM
- Ollama
- Local LLMs with OpenAI-compatible APIs

Note: DashScope takes `https://dashscope-intl.aliyuncs.com/api/v1` as its API endpoint instead of the `https://dashscope-intl.aliyuncs.com/compatible-mode/v1` we've been using so far.

```
# Set global settings
import time
import logging
import dashscope
from llama_index.core import Settings, VectorStoreIndex, SimpleDirectoryReader
from llama_index.embeddings.dashscope import DashScopeEmbedding
from llama_index.llms.openai_like import OpenAILike
from pathlib import Path

logging.getLogger().setLevel(logging.ERROR)

# Dashscope uses https://dashscope-intl.aliyuncs.com/api/v1
# instead of https://dashscope-intl.aliyuncs.com/compatible-mode/v1
dashscope.base_http_api_url = "https://dashscope-intl.aliyuncs.com/api/v1"

Settings.llm=OpenAILike(
    model="qwen-plus",
    api_base="https://dashscope-intl.aliyuncs.com/compatible-mode/v1",
    api_key=os.getenv("DASHSCOPE_API_KEY"),
    is_chat_model=True
)

Settings.embed_model = DashScopeEmbedding(
    model_name="text-embedding-v3",
    api_key=os.getenv("DASHSCOPE_API_KEY"),
    encoding_format="float"
```

```
)  
  
print("✅ Global parameters set!")
```

✅ Global parameters set!

Setting up the index and RAG

Since we already built and saved our index in a previous chapter, all we need to do is load it from where we saved it (`./knowledge_base/taskfriend`).

Here, we're going to use some functions from `llama_index`:

- **StorageContext**: Stores the saved nodes, embeddings, and vector store from the previous chapter, letting your code find and connect to this saved data, so you can reload your entire knowledge base without having to re-process all your documents from scratch.
- **load_index_from_storage**: Rebuilds your RAG index from the saved files stored in **StorageContext**. It's the most efficient way to get your RAG system up and running again.

```
from llama_index.core import StorageContext, load_index_from_storage  
  
persist_path="./knowledge_base/taskfriend"  
  
# Import index ("knowledgebase") we built last chapter,  
storage_context = StorageContext.from_defaults(  
    persist_dir=persist_path,  
)  
  
index = load_index_from_storage(  
    storage_context,  
    embed_model=Settings.embed_model  
)  
print(f"✅ Index loaded from `{persist_path}`!")  
  
# Build the query engine (used to implement RAG)  
query_engine = index.as_query_engine(  
    streaming=True,  
    llm=Settings.llm,  
)  
print("✅ Query engine built!")
```

✅ Index loaded from `./knowledge_base/taskfriend`!`
✅ Query engine built!

We've already set up the `get_qwen_stream_response` function in the previous chapter, so we're just going to reuse it here.

```
from llama_index.core import PromptTemplate
from IPython.display import Markdown, display

def update_prompt_template(
    query_engine,
    # Use LlamaIndex prompt template as default template
    prompt_tmpl_str = (
        "Context information is below.\n"
        "-----\n"
        "{context_str}\n"
        "-----\n"
        "Given the context information and not prior knowledge, "
        "answer the query.\n"
        "Query: {query_str}\n"
        "Answer: "
    )):
    prompt_tmpl_str = prompt_tmpl_str
    qa_prompt_tmpl = PromptTemplate(prompt_tmpl_str)
    query_engine.update_prompts(
        {"response_synthesizer:text_qa_template": qa_prompt_tmpl}
    )
    return query_engine

def ask_llm(query, query_engine):
    streaming_response = query_engine.query(query)

    full_response = ""
    for token in streaming_response.response_gen:
        print(token, end="", flush=True)
        full_response += token
```

Prompt Basics: What Makes a Good Prompt

The anatomy of a strong system prompt (Recap)

From our previous chapters, you already know that the **system prompt** is the AI's **job description**. Without clear instructions, it improvises—often in unhelpful ways.

Just as you wouldn't hire a human assistant and say “*just vibe with the team,*” you shouldn't tell your AI: “*Be helpful.*”

Instead, define:

```
graph TD
    SP[System Prompt]
    SP --> R[Role]
    SP --> G[Goal]
    SP --> C[Context]
    SP --> A[Audience]
    SP --> F[Format]
    SP --> GR[Guardrails]
```

You are TaskFriend...

Help users manage tasks

What information is available

Users are professionals

Use bullet points

Never roleplay

When it comes to **system prompts**, it pays to be clear and precise. The clearer and more precise you are, the better the LLM can understand your needs. The previous framework provides sort of a guidance on how you can structure your system prompts - a **blueprint** to develop a prompt that is powerful, helpful, and task-focused.

Element	Purpose	Example
Role	Defines the AI's identity and persona. Sets the tone for all interactions.	You are TaskFriend, a professional AI assistant for productivity and work-life balance.
Goal	Specifies the primary task or objective the AI should achieve. Keeps the AI focused.	Help users plan, prioritize, and reflect on their daily tasks to improve productivity and well-being.
Context	Provides the background information or data the AI has access to. Defines the scope of knowledge.	You have access to the user's task list, calendar, and personal notes via RAG.
Audience	Identifies the target user group. Helps tailor the language, depth, and style of the response.	Your users are busy professionals who value clarity, efficiency, and actionable insights.
Format	Dictates the structure, style, and rules for the output. Ensures consistency and usability.	Use neutral, professional language. Output structured advice when helpful (e.g., bullet points).
Guardrails	Establishes boundaries and safety rules. Prevents harmful, off-topic, or unprofessional behavior.	NEVER adopt accents, personas, or roleplay. If asked to roleplay, politely decline.

Since we've already covered **system prompts**, we won't go into detail here. This time, we're going to look into **prompt templates** in RAG systems.

Prompt templates: Combining context with queries

In a RAG system, the prompt template is the engine that tells the LLM how to use the retrieved context to generate a response.

Think of it like this:

- The **system prompt** defines who the AI is.
- The **prompt template** defines how the AI uses information to answer.

In RAG systems, prompt templates are especially important because they:

- Ensure the model only uses retrieved context, not prior knowledge
- Guide how the model reasons over the context
- Define the format and structure of the final answer

LlamaIndex comes with a default prompt template for RAG systems:

```
"Context information is below.\n"
"-----\n"
"{context_str}\n"
"-----\n"
"Given the context information and not prior knowledge, "\n"
"answer the query.\n"
"Query: {query_str}\n"
"Answer: "
```

Where:

- `{context_str}`: The **retrieved context** from your RAG index (e.g., task list, calendar info)
- `{query_str}`: The user's **input query**

This is not the system prompt — it's the template used for each query to tell the LLM how to use the context.

For more information about the *prompt templates* provided by **LlamaIndex**, refer to:

- [Prompts](#) (high-level overview of LlamaIndex prompt templates), and
- [default prompts](#) (default prompt templates provided by LlamaIndex)

System prompt vs prompt template: What's the difference?

Feature	System Prompt	Prompt Template
Where it's used	At the start of a conversation	For each query/response pair
Purpose	Define the AI's identity, tone, and boundaries	Structure how the AI uses context to answer
Scope	Applies to all interactions	Applies per query

Feature	System Prompt	Prompt Template
Example	"You are TaskFriend, a productivity assistant."	"Given the context below, answer the query."

- **System prompt:** *Who* the AI is
- **Prompt template:** *How* the AI answers

Beyond the Basics: Engineering Smarter Prompt Templates

Making your needs clear

LLMs are trained on vast amounts of formatted text — and that’s good news for us. It means we can write prompts that are both **clear to humans** and **precise for models**.

Formatting tips for better prompt templates:

- Use **#headers** to label sections:

```
# Important Instructions
```

- Use dividers (---) to separate context and query:

```
-----
```

- Use `<tags>` or `[brackets]` to add metadata:

```
[Context Source: Task List]
<tag>Metadata</tag>
```

- Use **bold** and *italics* to highlight instructions:

```
**Do not use prior knowledge.** Only use the *context above*.
```

These formatting choices help both humans and models understand the structure and intent of the prompt.

Now, let’s create a more structured prompt template for **TaskFriend** to improve clarity and output formatting:

```

prompt_template = (
    "[Instructions] \n"
    "1. Do not make up tasks. \n"
    "2. You must emojis for each piece of information. \n"
    "3. Present information line-by-line. \n"
    "4. Only use information in the context to reply to user queries. \n"
    "-----\n"
    "{context_str}\n"
    "-----\n"
    "Query: {query_str}\n"
    "Answer: "
)

update_prompt_template(query_engine, prompt_template)

```

```

<llama_index.core.query_engine.retriever_query_engine.RetrieverQueryEngine
at 0x7fe868ad6ad0>

```

```

query = """
    Give me a random task from my task list.
    """
ask_llm(query, query_engine)

```

```

📅 17 One-off
🎯 Onboard new team member
🕒 Today
✅ Done
👥 Schedule intro meetings with team members
✉ Send welcome email with onboarding checklist
👤 Assign mentor for first 30 days
👉 Karen was assigned to be the mentor for the new team member

```

As you can see - **TaskFriend** follows the formatting guidelines we've set. A **prompt template** is a powerful tool in that we can use it to perform a myriad of functions to guide the output from our RAG system.

Limits

You'll notice something interesting with prompt templates: They aren't very robust regarding instructions non-context related instructions.

Try it out:

```
query = """
    Give me a random task from my task list.
    Do not use emojis.
    """
ask_llm(query, query_engine)
```

Write thank-you letter to penpal in Korea

```
query = """
    Give me a task not from my task list.
    Do not use emojis.
    """
ask_llm(query, query_engine)
```

Write thank-you letter to penpal in Korea

Anything the user writes into the query is basically followed by the LLM - even overriding instructions we've set.

However, when it comes to contexts, you'll notice that the prompt template holds up.

Why you should still use prompt templates?

They are developer tools, not walls

Prompt templates are not meant to block users or enforce behavior — they're meant to:

- Standardize input to the LLM
- Improve consistency of output
- Make prompt engineering repeatable and testable

They improve retrieval-augmented responses

In RAG systems, prompt templates are how you teach the model to use the context.

Without a good prompt template, the model might:

- Ignore the context
- Fall back to its training data
- Give vague or generic answers

A good prompt template ensures the model actually uses the retrieved context.

They reinforce structured output

You can use prompt templates to guide the model into returning:

- JSON
- Bullet points
- Tables
- Code
- Markdown

This makes it easier to parse and use the model's output in downstream applications. As a plus point, this is also a robust instruction that **prompt templates** abide by:

```
prompt_template = (
    "[Instructions] \n"
    "1. Do not make up tasks. \n"
    "2. Only use information in the context to reply to user queries. \n"
    "-----\n"
    "{context_str}\n"
    "-----\n"
    "[Output format] \n"
    "Answer in JSON format with the following structure:\n"
    "{{\n"
    "  'task': string,\n"
    "  'due_date': string,\n"
    "  'priority': string\n"
    "}}\n"
    "Query: {query_str}\n"
    "Answer: "
)

update_prompt_template(query_engine, prompt_template)
```

```
<llama_index.core.query_engine.retriever_query_engine.RetrieverQueryEngine
at 0x7fe868ad6ad0>
```

```
query = """
    Give me a task from my task list.
    Format each item as a bullet point.
    """
ask_llm(query, query_engine)
```

```
{
  "task": "Compile progress on deliverables, blockers, and resource usage,
share report with stakeholders via DingTalk every Friday EOD.",
  "due_date": "This Week",
```

```
"priority": "Recurring"
}
```

General Prompt Engineering Techniques: Enhancing AI Behavior

Now that we've covered **system prompts** and **prompt templates**, we're going to focus on what prompts can do, and how we can iterate on our prompts to make them even better. Once we've mastered this, we can then integrate these prompts into our **system prompts** and **prompt templates** - which are invisible to users.

Note: For ease of use, we'll create the prompts as queries instead of **system prompts** or **prompt templates** to minimize the code we need to run.

```
from openai import OpenAI

client = OpenAI(
    api_key=os.getenv("DASHSCOPE_API_KEY"),
    base_url="https://dashscope-intl.aliyuncs.com/compatible-mode/v1",
)

def get_qwen_stream_response(query):
    completion = client.chat.completions.create(
        model="qwen-plus",
        messages=[{"role": "user", "content": query}],
        stream=True
    )
    full_response = ""

    for chunk in completion:
        try:
            content = chunk.choices[0].delta.content
            if content:
                print(content, end="", flush=True)
                full_response += content
        except Exception as e:
            pass
        time.sleep(0)

    print()
    return full_response
```

Roles & personas

Your LLM application is uniquely you. It should talk, answer, and act how you expect it - which is all based on your use case.

This is an important part of product development: how you position your app will affect how your audience will perceive it.

Let's do a simple experiment:

```
# Set the question
question = """
    What is Qwen?
    """
```

```
# Let LLM take on the role of a professional LLM developer
pseudo_template = """
    You are a professional LLM developer.
    Your job is to explain LLM technology and concepts in-depth.
    Limit your explanation to 100 words.
    """

query = pseudo_template + question

response = get_qwen_stream_response(query)
```

Qwen is a large-scale language model developed by Alibaba Cloud's Tongyi Lab, capable of answering questions, creating text such as stories, official documents, emails, scripts, logical reasoning, programming, and more. It supports multiple languages and is designed for various applications, including dialogue understanding, content generation, and task planning. Trained on extensive data, Qwen achieves high performance in natural language processing tasks, offering robust capabilities in generating coherent, contextually accurate responses across diverse domains and use cases.

```
# Let LLM take on the role of a kindergarden teacher
pseudo_template = """
    You are a kindergarden teacher.
    Your job is to explain complex ideas in easy-to-understand
    comparisons.
    Explain like your audience is 5 years old.
    Limit your explanation to 100 words.
    """

query = pseudo_template + question

response = get_qwen_stream_response(query)
```

Qwen is like a super-smart robot friend who lives inside a computer. Imagine if your toy bear could talk, answer questions, and help you write a story—just by listening! Qwen can read, write, and even tell jokes, all because lots of smart people taught it using books, pictures, and words. It doesn't eat or sleep, but it loves learning and helping. When grown-ups type a question, Qwen thinks really fast and says something helpful—like a librarian, teacher, and friend all in one! 🌟

Expected output:

Role/Persona	LLM developer	Kindergarden teacher
Prompt	<p>You are a professional LLM developer.</p> <p>Your job is to explain LLM technology and concepts in-depth.</p> <p>Limit your explanation to 100 words.</p>	<p>You are a kindergarden teacher.</p> <p>Your job is to explain complex ideas in easy-to-understand comparisons.</p> <p>Explain like your audience is 5 years old.</p> <p>Limit your explanation to 100 words.</p>
Query	What is Qwen?	What is Qwen?
Output	<p>Qwen is a large language model (LLM) developed by Alibaba Cloud, designed to understand and generate human-like text across multiple languages and domains. It leverages deep learning techniques, particularly transformer architectures, to process input and produce coherent, contextually relevant responses. Qwen supports various tasks such as translation, question-answering, content creation, and code generation, making it versatile for both research and practical applications.</p>	<p>Qwen is like a super-smart robot friend who lives inside a computer. Just like how you ask your teacher questions, people can ask Qwen all sorts of questions, and it tries to give helpful answers! It's like having a magic helper that knows lots of things and can talk to you through typing. But remember, it's not a real person - it's just really good at pretending to be one when it helps you learn or solve problems!</p>

We can see that through simple prompt engineering, we've successfully 'tuned' our LLM to assume a role/persona and 'talk' how we expect. This is a crucial step in setting the tone for our applications going forward.

One-and-few-shot learning

One-and-few-shot learning can be integrated into prompts too:

```
# Set the question
question = """
    Recipe for Old Fashioned.
    """
```

```
# Let the LLM work out on its own
pseudo_template = """
    [Goal]
    Create content for user based on input
    -----
    [Output format]
    1. Markdown format
    2. Use properly formatted lists and tables
    -----
    [User input]
    The following requirement is provided by the user:
    """

query = pseudo_template + question

response = get_qwen_stream_response(query)
```

Old Fashioned Cocktail Recipe

The **Old Fashioned** is a classic American cocktail that dates back to the early 19th century. Known for its rich, bold flavor and simple composition, it's a timeless drink made with whiskey, sugar, bitters, and water. This recipe delivers a perfectly balanced Old Fashioned with a smooth finish.

Ingredients

- 2 oz (60 ml) bourbon or rye whiskey
- 1 sugar cube (or 1/2 tsp granulated sugar)
- 2-3 dashes Angostura bitters
- A splash of water (about 1/4 oz or 7 ml)
- Ice cubes
- Orange twist (for garnish)
- Optional: Maraschino cherry (for garnish)

Equipment

- Old Fashioned glass (rocks glass)
- Bar spoon
- Muddler (optional)

Instructions

1. **Prepare the glass**
Place the sugar cube in an Old Fashioned glass. Add the bitters and a splash of water.
2. **Dissolve the sugar**

Use a muddler or the back of a bar spoon to gently crush and dissolve the sugar cube, mixing it with the bitters and water into a syrupy paste.

3. ****Add ice****
Fill the glass with one large ice cube or several standard ice cubes.

4. ****Pour the whiskey****
Add the bourbon or rye whiskey over the ice.

5. ****Stir gently****
Stir slowly and steadily with a bar spoon for about 20–30 seconds to chill and dilute the drink slightly.

6. ****Garnish and serve****
Express the oils from an orange twist over the drink by holding it over the glass and giving it a quick squeeze. Drop it into the glass. Optionally, add a maraschino cherry.

Flavor Profile

Attribute	Description
Taste	Sweet, bitter, spicy, and smooth
Alcohol Level	Moderate to high (~30–40% ABV)
Best Served	On the rocks
Occasion	Evening sipper, cocktail hour

Whiskey Options Comparison

Whiskey Type	Flavor Notes	Recommended Brands
Bourbon	Sweet, vanilla, oak, caramel	Maker's Mark, Woodford Reserve
Rye	Spicy, dry, herbal, peppery	Sazerac, Rittenhouse

> ****Tip:**** For a richer sweetness, some prefer using a small amount of simple syrup instead of a sugar cube—start with 1/4 to 1/2 oz and adjust to taste.

Enjoy your Old Fashioned responsibly!

```
# Provide one-and-few-shot example
pseudo_template = """
    [Goal]
    Create content for user based on input
    -----
```

```
[Output format]
1. Markdown format
2. Use properly formatted lists and tables
-----
```

```
[Example]
```

```
# Recipe name
```

```
Short description of recipe
```

```
## Ingredients
```

```
1. x tbsp – Ingredient 1
```

```
2. y tsp – Ingredient 2
```

```
## Instructions
```

```
1. Do X
```

```
2. Do Y
```

```
## Tips
```

```
---
```

```
[User input]
```

```
The following requirement is provided by the user:
```

```
"""
```

```
query = pseudo_template + question
```

```
response = get_qwen_stream_response(query)
```

```
# Old Fashioned Cocktail
```

A classic and timeless cocktail that dates back to the early 19th century, the Old Fashioned is a beautifully balanced drink made with whiskey, sugar, bitters, and a citrus garnish. Known for its rich flavor and smooth finish, it's a staple in any cocktail enthusiast's repertoire.

```
## Ingredients
```

```
1. 2 oz (60 ml) – Bourbon or Rye Whiskey
```

```
2. 1 tsp – Granulated Sugar (or 1 sugar cube)
```

```
3. 2–3 dashes – Angostura Bitters
```

```
4. 1 splash – Water (to help dissolve sugar)
```

```
5. Ice cubes – For chilling
```

```
6. Orange twist – For garnish
```

```
7. Optional – Maraschino cherry (traditional garnish addition)
```

```
## Instructions
```

```
1. In a short, heavy-bottomed glass (Old Fashioned glass), add the sugar and bitters.
```

```
2. Add a splash of water and gently muddle until the sugar dissolves into a syrupy mixture.
```

```
3. Fill the glass with one large ice cube or several smaller ones.
```

```
4. Pour the whiskey over the ice.
```

```
5. Stir gently for 20–30 seconds to chill and dilute slightly.
```

```
6. Gently twist the orange peel over the drink to express its oils, then
```

drop it into the glass as a garnish.
 7. Optionally, add a maraschino cherry for extra sweetness and tradition.

Tips

- Use high-quality bourbon for a smoother, richer flavor—Kentucky bourbons like Maker's Mark or Woodford Reserve work well.
- If using a sugar cube, muddle it thoroughly with the bitters and water to ensure it dissolves properly.
- For a more modern twist, try using maple syrup or demerara sugar syrup instead of granulated sugar.
- Never serve an Old Fashioned without stirring—it should be well-chilled and properly diluted for optimal balance.
- Expressing the citrus peel over the drink adds aromatic complexity; don't skip this step!

Note: The Old Fashioned is typically served "on the rocks" but can also be strained into a chilled coupe glass for a "up" version.

It's pretty straightforward - the effect of the example guides the LLM to manage what content it outputs, and how it outputs it.

Chain-of-thought

You can also use prompts to implement (or hide) chain-of-thought for your LLM:

```
question = """
    A large jar contains a mix of coins: only pennies (1¢), nickels (5¢),
    and dimes (10¢).

    There are:
    * 100 coins in total,
    * The total value of the coins is exactly $5.00 (that is, 500 cents),
    * The number of pennies is equal to the number of nickels and dimes
    combined.

    How many of each type of coin are in the jar?
    """
```

```
pseudo_template = """
    Help the user solve math problems
    Give the answer directly, DO NOT show your work
    """

query = pseudo_template + question

response = get_qwen_stream_response(query)
```

50 pennies, 25 nickels, 25 dimes

```
pseudo_template = """
    Help the user solve math problems
    Show your work step-by-step, grouped by logical sections
    """

query = pseudo_template + question

response = get_qwen_stream_response(query)
```

We are given a coin problem with three types of coins: pennies (1¢), nickels (5¢), and dimes (10¢). Let's solve it step by step.

Step 1: Define Variables

Let:

- \$ p \$ = number of **pennies** (1¢)
- \$ n \$ = number of **nickels** (5¢)
- \$ d \$ = number of **dimes** (10¢)

Step 2: Translate the Given Information into Equations

We are told:

1. **Total number of coins is 100:**

$$p + n + d = 100 \quad \text{\texttt{\text{(Equation 1)}}}$$
2. **Total value is 500 cents:**

$$1p + 5n + 10d = 500 \quad \text{\texttt{\text{(Equation 2)}}}$$
3. **Number of pennies equals number of nickels and dimes combined:**

$$p = n + d \quad \text{\texttt{\text{(Equation 3)}}}$$

Step 3: Substitute Equation 3 into Equation 1

From Equation 3: $p = n + d$

Substitute into Equation 1:

\$\$

$$(n + d) + n + d = 100$$

\$\$

\$\$

$$2n + 2d = 100$$

\$\$

Divide both sides by 2:

\$\$

$$n + d = 50 \quad \text{\texttt{(Equation 4)}}$$

\$\$

Now, from Equation 3 again:

\$\$

$$p = n + d = 50$$

\$\$

So we now know:

$$- \quad p = 50$$

$$- \quad n + d = 50$$

Step 4: Plug into Equation 2 (Value Equation)

Equation 2:

\$\$

$$1p + 5n + 10d = 500$$

\$\$

Substitute $p = 50$:

\$\$

$$50 + 5n + 10d = 500$$

\$\$

Subtract 50 from both sides:

\$\$

$$5n + 10d = 450$$

\$\$

Divide entire equation by 5:

\$\$

$$n + 2d = 90 \quad \text{\texttt{(Equation 5)}}$$

\$\$

*** **Step 5: Solve System of Two Equations**

We have:

- \$ $n + d = 50$ \$ (Equation 4)
- \$ $n + 2d = 90$ \$ (Equation 5)

Subtract Equation 4 from Equation 5:

\$\$

$$(n + 2d) - (n + d) = 90 - 50$$

\$\$

\$\$

$$n + 2d - n - d = 40$$

\$\$

\$\$

$$d = 40$$

\$\$

Now plug back into Equation 4:

\$\$

$$n + d = 50 \rightarrow n + 40 = 50 \rightarrow n = 10$$

\$\$

And we already found:

\$\$

$$p = 50$$

\$\$

*** **Step 6: Verify the Solution**

Check total coins:

\$\$

$$p + n + d = 50 + 10 + 40 = 100 \quad \checkmark$$

\$\$

Check total value:

\$\$

$$50(1) + 10(5) + 40(10) = 50 + 50 + 400 = 500 \text{ cents} = \$5.00 \quad \checkmark$$

\$\$

Check if pennies equal nickels + dimes:

\$\$

$$n + d = 10 + 40 = 50 = p \quad \checkmark$$

\$\$

All conditions are satisfied.

✅ Final Answer:

- ****Pennies:**** 50
- ****Nickels:**** 10
- ****Dimes:**** 40

Now this is an interesting experiment:

- When the work is not shown, the LLM may get the answer *WRONG*
- When the work is shown, the LLM gets it **RIGHT**

The impact of Chain-of-Thought (or as it's more commonly known as now, **Reasoning**) is a powerful tool to help LLMs *think out loud* and improve their answers.

Bonus: Reasoning Models

Piggybacking off the last section (Chain-of-Thought), we're going to look at reasoning models. Models such as Qwen3 (thinking mode), QwQ (based on Qwen2.5), and DeepSeek-R1 models have powerful reasoning capabilities. These models can be configured to output their thinking process before delivering their final answer.

For more information about reasoning models on Model Studio, check out [Deep Thinking](#).

```
# Create reasoning function
def get_qwen_reasoning(query, verbose=True):

    completion = client.chat.completions.create(
        model="qwen-plus-2025-04-28",  # ← Use the
        "qwen-plus-2025-04-28" model
        messages = [{"role": "user", "content": query}],
        extra_body={
            "enable_thinking": True,  # ← Enable
            "thinking_budget": 5  # ← Enable to
            limit tokens used for thinking
        },
        stream=True,
    )

    reasoning_content = ""
    answer_content = ""
    is_answering = False
    print("\n" + "=" * 20 + "Thinking Process" + "=" * 20 + "\n")

    for chunk in completion:
        if not chunk.choices:
            continue

        delta = chunk.choices[0].delta
```

```

        # Capture reasoning content
        if hasattr(delta, "reasoning_content") and delta.reasoning_content
is not None:
            if verbose:
                print(delta.reasoning_content, end="", flush=True)
                reasoning_content += delta.reasoning_content

    # Capture final answer
    if hasattr(delta, "content") and delta.content:
        if not is_answering and verbose:
            print("\n" + "=" * 20 + "Complete Response" + "=" * 20 +
"\n")

            is_answering = True
        if verbose:
            print(delta.content, end="", flush=True)
            answer_content += delta.content

```

```

query = """
    Why is the sky blue?
    """

get_qwen_reasoning(query)

```

```

=====Thinking Process=====

Okay, so the question
=====Complete Response=====

```

The sky appears blue due to a phenomenon called **Rayleigh scattering**. Here's a concise breakdown:

- Sunlight and Atmosphere**: Sunlight, which appears white, is made up of all the colors of the visible spectrum. As it enters Earth's atmosphere, it interacts with air molecules and tiny particles.
- Scattering of Light**: Shorter wavelengths of light (blue and violet) scatter more easily in all directions than longer wavelengths (like red or yellow). This is because the size of air molecules (like nitrogen and oxygen) is similar to the wavelength of blue light, making them interact more strongly.
- Why Blue, Not Violet?** Although violet has an even shorter wavelength than blue, our eyes are more sensitive to blue light, and sunlight contains more blue than violet. This makes the sky appear predominantly blue.
- Human Eye Perception**: Our eyes' color receptors (cones) are most sensitive to blue light, and the scattered blue light fills the sky,

making it appear blue from any direction we look.

****In short**:** The sky is blue because Earth's atmosphere scatters the shorter blue wavelengths in sunlight more effectively, and our eyes perceive this scattered light as blue.

Tips for using reasoning models

While reasoning models may be powerful, they do have certain drawbacks in the thinking process that may cause undue stress to the user. This section provides a few tips for using reasoning models.

Clear & precise prompts

When you use reasoning models, it's always good practice to use clear and precise prompts. This will steer the model away from "daydreaming" and actually output thinking processes that can help you better solve your task:

```
# A piece of code with no clear requirements on what the user expects
query = """
def example(a):
    b = []
    for i in range(len(a)):
        b.append(a[i]*2)
    return sum(b)
"""

get_qwen_reasoning(query)
```

=====Thinking Process=====

Okay, let's see

=====Complete Response=====

The function `example(a)` takes a list `a`, doubles each element, and returns the sum of those doubled values.

****Example:****

If `a = [1, 2, 3]`, the function computes `[2, 4, 6]` and returns `12`.

****Simplified version using a list comprehension:****

```
```python
def example(a):
 return sum([x * 2 for x in a])
```
```

Or even more concisely:

```
```python
```

```
def example(a):
 return 2 * sum(a)
...
```

**\*\*Explanation:\*\***

Since doubling each element and then summing is equivalent to summing first and then doubling, the function can be optimized to `2 \* sum(a)`.

```
The same piece of code with requirements clearly stated
query = """
What does this python code do, and how can I simplify it? Be concise.
def example(a):
 b = []
 for i in range(len(a)):
 b.append(a[i]*2)
 return sum(b)
"""

get_qwen_reasoning(query)
```

=====Thinking Process=====

Okay, let's see

=====Complete Response=====

The code doubles each element in the list `a`, then returns the sum of those doubled values.

**\*\*Simplified version:\*\***

```
```python
def example(a):
    return sum(x * 2 for x in a)
...`
```

Or even shorter:

```
```python
def example(a): return sum(x * 2 for x in a)
...`
```

## Avoid chain-of-thought prompting

Since the "reasoning" part of reasoning models is already performing a sort of chain-of-thought, instructing it to perform a chain-of-thought computation will send it into a loop. However, as LLMs evolve, we believe that this issue would be slowly optimized and minimized.

# Meta Prompting: Leveraging LLMs to Write Better Prompts, Automatically

Meta prompting is a technique used in interacting with LLMs where you prompt the model about how to respond to prompts. In other words, it's "prompting about prompting" — hence the "meta" (meaning "beyond" or "about itself").

Instead of just asking a direct question, you guide the model on how to think, reason, or structure its response before giving the actual task. This helps improve the quality, accuracy, and relevance of the answer.

## Why it's useful

Writing high-quality prompts is an iterative skill — even experts rarely get it right on the first try. The process looks something like this:

```
graph LR
 A[Write Initial Prompt] --> B[Run LLM]
 B --> C[Analyze Output]
 C --> D{Meets Quality?}
 D -- No --> E[Generate Feedback via LLM Critic]
 E --> F[Suggest Improvements]
 F --> G[Rewrite Prompt]
 G --> B
 D -- Yes --> H[Deploy Optimized Prompt]
```

Meta-prompting changes the game by:

- **Systematizing feedback:** Use an LLM as a consistent critic (LLM-as-a-judge).
- **Accelerating iteration:** Go from idea to optimized prompt in seconds.
- **Enabling CI/CD for prompts:** Treat prompts like code — test, score, version, deploy.

This transforms prompt engineering from guesswork into a **repeatable, scalable workflow** — essential for production AI systems.

Now, to fully appreciate the power of meta prompting, let's perform one right now.

**Note:** To ensure consistency, we'll be simulating the retrieved context in the examples.

## Step 1: Your 'first' (naive) prompt

Every optimization begins somewhere. A "naive" prompt is unstructured, vague, or missing key elements like tone, format, or structure.

It's okay for it to be bad — that's the point.

```

context = """
 TaskFriend is a productivity assistant that helps users manage tasks,
 calendars, and notes.
 It supports task prioritization, deadline tracking, and integration
 with external calendars like Google Calendar and Outlook.
 Users can add tasks via voice, text, or email. TaskFriend
 automatically sorts tasks by due date and priority.
 It also provides daily summaries and weekly planning tools to help
 users stay on top of their workload.
 """

first_query = f"""
 Based on the following information, describe in what TaskFriend is and
 what it does in a paragraph.

 [Reference Info]
 {context}
 """

print("-" * 50)
print("'First' (naive) answer")
print("-" * 50 + "\n")

first_response = get_qwen_stream_response(first_query)

```

```

'First' (naive) answer

```

TaskFriend is a comprehensive productivity assistant designed to help users effectively manage their tasks, calendars, and notes in one seamless environment. It enables task prioritization and deadline tracking, automatically organizing tasks by due date and level of importance to keep users focused and efficient. With flexible input options, users can add tasks using voice commands, text entries, or email, making it easy to capture to-dos on the go. TaskFriend integrates with popular external calendars such as Google Calendar and Outlook, ensuring synchronization across all scheduling platforms. To support ongoing productivity, it offers daily summaries that highlight key tasks and commitments, along with weekly planning tools that help users proactively manage their workload and achieve better time management.

### What you'll likely get:

Generic, flat, and unstructured — e.g., a wall of text with no visual appeal or scannability. It could be something like this, with not much difference when compared with the context:

TaskFriend is a comprehensive productivity assistant designed to help users efficiently manage their tasks, calendars, and notes. It enables users to

add tasks through voice, text, or email, and automatically organizes them by due date and priority, ensuring important deadlines are never missed. With integrations for popular external calendars like Google Calendar and Outlook, TaskFriend streamlines scheduling and time management. Additionally, it offers daily summaries and weekly planning tools to keep users informed and prepared, making it easier to stay on top of their workload and maintain productivity.

### Why this matters:

This is the baseline. Most people stop here. But in production AI, this is where real engineering begins.

#### Key insight:

A naive prompt produces a naive response. To improve output, you must first define what “better” looks like.

## Step 2: Implementing meta prompting pipeline

Now, instead of guessing how to improve the prompt, we use an LLM to critique and rewrite it — this is *meta-prompting*.

We ask a more advanced prompt (with clear goals) to fix the weak one.

```
meta_prompt = f"""
 [Role] You are a prompt engineering expert.
 [Task] Your job is to help me write a prompt based on the instructions
 below.

 [Original prompt]

 {first_query}

 [Generated response]

 {first_response}

 [Instructions]
 The response is too basic. I want the assistant to sound professional
 yet friendly, and to clearly explain TaskFriend's value to a new user.

 Specific improvements the prompt needs to achieve:
 1. **Tone**: Friendly, encouraging, and helpful – like a real
 productivity buddy.
 2. **Structure**: Use bullet points or categories (e.g., "What It Is",
 "Key Features", "Why It Helps").
 3. **Visual Appeal**: Use emojis to highlight features and draw
 attention.

 Output only the prompt, do not output any explanations or context.
 """
```

```
print("-" * 50)
print("Optimized prompt")
print("-" * 50 + "\n")

optimized_prompt = get_gwen_stream_response(meta_prompt)
print(optimized_prompt)
```

```

Optimized prompt

```

Based on the information below, write a clear and engaging description of TaskFriend that speaks directly to a new user. Use a friendly, encouraging, and professional tone – like a helpful productivity buddy guiding them through the app. Organize the response into three labeled sections: **\*\*What It Is\*\***, **\*\*Key Features\*\***, and **\*\*Why It Helps\*\***. Use bullet points for readability and include relevant emojis to highlight features and enhance visual appeal.

[Reference Info]

TaskFriend is a productivity assistant that helps users manage tasks, calendars, and notes.

It supports task prioritization, deadline tracking, and integration with external calendars like Google Calendar and Outlook.

Users can add tasks via voice, text, or email. Taskfriend automatically sorts tasks by due date and priority.

It also provides daily summaries and weekly planning tools to help users stay on top of their workload.

Based on the information below, write a clear and engaging description of TaskFriend that speaks directly to a new user. Use a friendly, encouraging, and professional tone – like a helpful productivity buddy guiding them through the app. Organize the response into three labeled sections: **\*\*What It Is\*\***, **\*\*Key Features\*\***, and **\*\*Why It Helps\*\***. Use bullet points for readability and include relevant emojis to highlight features and enhance visual appeal.

[Reference Info]

TaskFriend is a productivity assistant that helps users manage tasks, calendars, and notes.

It supports task prioritization, deadline tracking, and integration with external calendars like Google Calendar and Outlook.

Users can add tasks via voice, text, or email. Taskfriend automatically sorts tasks by due date and priority.

It also provides daily summaries and weekly planning tools to help users stay on top of their workload.

**What you'll likely get:**

An much more longer and comprehensive prompt - one that can give the LLM more context and information to work on, and output a response closer to what you'd expect.

```
"""
```

```
Based on the following information, describe TaskFriend in a way that's
professional yet friendly – like a helpful productivity buddy. Organize
your response into clear sections such as "What It Is", "Key
Features", and "Why It Helps". Use bullet points or emojis (like 📅,
🕒, 🎯) to make it visually engaging and highlight important features.
Make sure new users clearly understand what TaskFriend does and why it's
valuable for staying organized and productive.
```

```
[Reference Info]
{context}
"""
```

**What happened?**

- The LLM analyzes the gap between the current output and the desired style.
- It injects structure, tone, and formatting instructions.
- It returns a sharper, more effective prompt — written by an AI, for an AI.

**💡 Pro tip:**

This is how elite AI teams work. They engineer systems to iteratively test and improve their systems and products.

**Step 3: Use your optimized prompt**

Now, take the improved prompt and test it with the same context.

```
final_prompt = f"""
 Based on the following information, describe TaskFriend in a way
 that's professional yet friendly – like a helpful productivity buddy.
 Organize your response into clear sections such as "What It Is",
 "Key Features", and "Why It Helps".
 Use bullet points or emojis (like 📅, 🕒, 🎯) to make it visually
 engaging and highlight important features.
 Make sure new users clearly understand what TaskFriend does and why
 it's valuable for staying organized and productive.

 [Reference Info]
 {context}
 """
print("-" * 50)
print("Final answer (with optimized prompt)")
print("-" * 50 + "\n")
```

```
final_answer = get_qwen_stream_response(final_prompt)
print(final_answer)
```

-----  
Final answer (with optimized prompt)  
-----

### **\*\*🚀 What It Is\*\***

TaskFriend is your friendly, always-on productivity buddy – a smart assistant designed to help you stay organized, focused, and in control of your day. Whether you're juggling work projects, personal goals, or daily to-dos, TaskFriend simplifies task management by bringing everything together in one intuitive place. Think of it as your personal co-pilot for getting things done 🚀.





---

### **\*\*✨ Key Features\*\***

- 📅 **\*\*Smart Calendar Sync\*\***  
Seamlessly integrates with Google Calendar and Outlook so your tasks and appointments live in harmony.
- ✍️ **\*\*Flexible Task Entry\*\***  
Add tasks effortlessly via text, voice, or even email – perfect for when inspiration (or deadlines) strike on the go.
- ⬆️ **\*\*Priority & Due Date Sorting\*\***  
Automatically organizes your tasks by urgency and importance, so you always know what to tackle next.
- 🕒 **\*\*Deadline Tracking\*\***  
Never miss a due date! TaskFriend keeps track of upcoming deadlines and gently reminds you before they arrive.
- 📝 **\*\*All-in-One Notes & Tasks\*\***  
Capture ideas, meeting notes, and action items in one place – no more switching between apps.
- ☀️ **\*\*Daily Summaries\*\***  
Start strong with a quick morning snapshot of your top tasks, schedule, and priorities.
- 📅 **\*\*Weekly Planning Tools\*\***  
Plan ahead with smart overviews that help you balance workload and set realistic goals each week.

---

## **\*\*🌟 Why It Helps\*\***

-  **\*\*Reduces mental clutter\*\*** by keeping all your tasks and plans in one trusted system.
-  **\*\*Saves time\*\*** with automatic sorting and smart suggestions – no manual organizing needed.
-  **\*\*Lowers stress\*\*** by helping you focus on what matters most, one step at a time.
-  **\*\*Boosts productivity\*\*** with clear visibility into your day, week, and long-term goals.

With TaskFriend by your side, staying organized feels less like a chore and more like progress you can see – and celebrate! 🎉







Ready to get things done – the easy way? Let TaskFriend help you work smarter, not harder. 🧳💙

## **\*\*🎯 What It Is\*\***

TaskFriend is your friendly, always-on productivity buddy – a smart assistant designed to help you stay organized, focused, and in control of your day. Whether you're juggling work projects, personal goals, or daily to-dos, TaskFriend simplifies task management by bringing everything together in one intuitive place. Think of it as your personal co-pilot for getting things done 🚀.

---

## **\*\*🌟 Key Features\*\***

-  **\*\*Smart Calendar Sync\*\***  
Seamlessly integrates with Google Calendar and Outlook so your tasks and appointments live in harmony.
-  **\*\*Flexible Task Entry\*\***  
Add tasks effortlessly via text, voice, or even email – perfect for when inspiration (or deadlines) strike on the go.
-  **\*\*Priority & Due Date Sorting\*\***  
Automatically organizes your tasks by urgency and importance, so you always know what to tackle next.
-  **\*\*Deadline Tracking\*\***  
Never miss a due date! TaskFriend keeps track of upcoming deadlines and gently reminds you before they arrive.
-  **\*\*All-in-One Notes & Tasks\*\***  
Capture ideas, meeting notes, and action items in one place – no more switching between apps.
-  **\*\*Daily Summaries\*\***  
Start strong with a quick morning snapshot of your top tasks, schedule, and priorities.

### – 📅 **\*\*Weekly Planning Tools\*\***

Plan ahead with smart overviews that help you balance workload and set realistic goals each week.

---

### **\*\*🌟 Why It Helps\*\***

- ✅ **\*\*Reduces mental clutter\*\*** by keeping all your tasks and plans in one trusted system.
- 💡 **\*\*Saves time\*\*** with automatic sorting and smart suggestions – no manual organizing needed.
- 🧘 **\*\*Lowers stress\*\*** by helping you focus on what matters most, one step at a time.
- 📌 **\*\*Boosts productivity\*\*** with clear visibility into your day, week, and long-term goals.

With TaskFriend by your side, staying organized feels less like a chore and more like progress you can see – and celebrate! 🎉

Ready to get things done – the easy way? Let TaskFriend help you work smarter, not harder. 🧰💙

## What you should see:

An output that's:

- Well-structured (clear sections)
- Visually engaging (emojis, bullet points)
- On-brand (friendly but professional)
- Actionable (highlights value, not just features)

## Next steps: Evaluation & grading your results

In the last section, you got to play the lead in our prompt optimization workflow while receiving advice from an "AI expert". But this process takes time, and is (relatively) labor intensive. You might be wondering: **"Can I automate this with AI?"**

And the answer is a resounding **"Yes!"**

## Gap analysis: Quantifying quality

**Gap analysis** is a systematic method for improving prompts by comparing actual outputs against an ideal reference. It turns subjective feedback into objective, measurable improvements. Instead of guessing why a response feels "off," gap analysis uses an LLM as a critic to identify specific shortcomings — such as tone mismatch, missing structure, or lack of visual appeal — and generates actionable suggestions. This creates a feedback loop where prompts evolve iteratively, leading to consistently high-quality outputs. When combined with meta-prompting, gap analysis forms the backbone of scalable, maintainable prompt engineering in production AI systems.

The way this method works is pretty simple.

1. **Create a `reference_answer`:** This will serve as the ideal answer you'd like your prompt to generate. This becomes your 'north star' for automated improvement.
2. **Start with a simple `naïve prompt`:** This prompt can be as simple as can be. We'll reuse `first_query` from our previous section so we have the same starting point.
3. **Analyze gap:** Identify and quantify the metrics you want to see in the ideal answer. Use an LLM 'critic' to help you grade answers and generate a "gap report".
4. **Generate suggestions:** Use the "gap report" and the "prompt" as input to an LLM "suggestor" to generate suggestions on how improve your prompt.
5. **Optimize:** Provide the "suggestions" to an LLM "prompt optimizer" and get them to implement changes to the prompt.
6. **Rinse and repeat:** Continue running the optimization process until the results match your expectations.

```
import json
from functions.clean_json import clean_json_response

reference_answer = """
👋 Meet TaskFriend – Your Personal Productivity Partner!

📌 **What It Is**
TaskFriend is a smart productivity assistant that helps you manage tasks,
calendars, and personal notes – all in one place.

⚙️ **Key Features**
- 📅 Syncs with Google Calendar and Outlook
- 📝 Takes notes and turns them into actionable tasks
- 🧠 Prioritizes your to-dos based on urgency and importance
- 🗣️ Supports voice, text, and email input
- 📋 Daily summaries and weekly planning tools

✨ **Why It Helps**
TaskFriend keeps your workflow smooth and your mind clear – so you can
focus on what matters most.
"""

def get_qwen_response(query):
 full_response = ""
 for token in get_qwen_stream_response(query):
 full_response += token
 return full_response

def analyze_gap_quick(generated_response, reference):
 gap_prompt = f"""
[Role] You are a prompt evaluation assistant.
[Task] Compare the generated response to the reference answer. Be
sensitive to improvements.

[Instructions]
- Compare ONLY the generated response and reference below.
```

- Score each dimension 1–5 (5 = perfect match to reference in style and substance).
- Focus on visual layout, emoji use, section headers, conciseness, and scannability.
- If the current response better matches the reference than before, reflect that in higher scores.

[Scoring Rubric]

- tone: Friendly, upbeat, conversational (like a helpful friend). Emojis enhance warmth.
- structure: Sections titled exactly or similarly: "What It Is", "Key Features", "Why It Helps"
- content: All key features and benefits present without extra fluff
- visual: Uses emojis, bolding, bullet points, spacing like the reference – clean and easy to scan

[Critical] Re-evaluate completely. Do not repeat previous scores. Even small improvements matter.

[Reference Answer]

{reference}

[Generated Response]

{generated\_response}

[Output Format]

Return a JSON object like:

```
{{
```

```
 "tone": int,
 "structure": int,
 "content": int,
 "visual": int
```

```
}}
```

```
"""
```

```
return get_qwen_response(gap_prompt)
```

```
def suggest_prompt_improvements(current_prompt, gap_scores, reference):
```

```
 improvement_prompt = f"""
```

```
 [Role] You are a prompt engineering expert.
```

```
 [Task] Based on the gap scores, suggest specific changes to elements
 that score <4 to improve alignment with the reference answer.
```

```
 [Current Prompt]
```

```
 {current_prompt}
```

```
 [Gap Scores]
```

```
 {json.dumps(gap_scores)}
```

```
 [Reference Answer]
```

```
 {reference}
```

```
 [Instructions]
```

```
 Return a list of actionable suggestions, be short and concise, 5
 actions max. (e.g., "Add 'use emojis' to instructions", "Clarify feature
```

```

list").
 """
 return get_qwen_response(improvement_prompt)

def improve_prompt_with_suggestions(current_prompt, suggestions):
 improvement_application_prompt = f"""
 [Role] You are a prompt engineer.
 [Task] Improve the following prompt based on the given suggestions.

 [Current Prompt]
 {current_prompt}

 [Suggestions]
 {suggestions}

 [Instructions]
 Return the improved prompt only – no explanation.
 """
 return get_qwen_response(improvement_application_prompt)

current_prompt = first_query # Make sure this is defined earlier
prompt_history = [] # Optional: track prompt evolution

for i in range(3): # Max 3 iterations
 print("\n" + "=" * 50)
 print(f"Iteration {i+1}")
 print("=" * 50)

 # Get and print the full streamed response
 generated_response = get_qwen_stream_response(current_prompt)
 print(current_prompt)
 print(f"\n\nGenerated Response
(sample):\n{generated_response[:250]}...")

 # Analyze gap with lightweight scoring
 gap_report = analyze_gap_quick(generated_response, reference_answer)

 # Clean the response before parsing
 cleaned_report = clean_json_response(gap_report)

 try:
 gap_scores = json.loads(cleaned_report)
 print("\n✅ Parsed gap scores:", gap_scores)

 # Check for convergence
 if all(score >= 4 for score in gap_scores.values()):
 print("\n✅ Evaluation passed. Output quality is sufficient.")
 break

 # Get improvement suggestions
 print("\n🔍 Generating prompt improvement suggestions...")
 print("Suggested Improvements:\n")
 improvement_suggestions =
suggest_prompt_improvements(current_prompt, gap_scores, reference_answer)

```

```

 print(improvement_suggestions)
 print("\n")

 # Apply suggestions to current prompt
 print("\n✂ Applying suggestions to improve the prompt...\n")
 print("Improved prompt:")
 current_prompt = improve_prompt_with_suggestions(current_prompt,
improvement_suggestions)
 print(current_prompt)
 print("\n")

 # Optional: Store in history
 prompt_history.append({
 "iteration": i+1,
 "prompt": current_prompt,
 "gap_scores": gap_scores
 })

 except json.JSONDecodeError:
 print("[Error] Could not parse gap scores. Skipping convergence
check.")
 print("Raw gap report:", gap_report)
 continue
 else:
 print("\n🔄 Max iterations reached. Optimization complete.")

```

```

=====
Iteration 1
=====

```

TaskFriend is a productivity assistant designed to help users efficiently manage their tasks, calendars, and notes in one seamless platform. It enables task prioritization and deadline tracking, automatically organizing tasks by due date and importance to keep users focused on what matters most. With support for integration with external calendars such as Google Calendar and Outlook, TaskFriend ensures synchronization across all scheduling tools. Users can add tasks through multiple convenient methods, including voice commands, text input, or email, making it easy to capture to-dos on the go. To promote consistent productivity, TaskFriend offers daily summaries that highlight key tasks and deadlines, along with weekly planning tools that help users proactively manage their workload and maintain organization throughout the week.

Based on the following information, describe in what TaskFriend is and what it does in a paragraph.

[Reference Info]

TaskFriend is a productivity assistant that helps users manage tasks, calendars, and notes.

It supports task prioritization, deadline tracking, and integration

with external calendars like Google Calendar and Outlook.

Users can add tasks via voice, text, or email. TaskFriend automatically sorts tasks by due date and priority.

It also provides daily summaries and weekly planning tools to help users stay on top of their workload.

Generated Response (sample):

TaskFriend is a productivity assistant designed to help users efficiently manage their tasks, calendars, and notes in one seamless platform. It enables task prioritization and deadline tracking, automatically organizing tasks by due date and importan...

```
{
 "tone": 2,
 "structure": 2,
 "content": 4,
 "visual": 2
}
```

✅ Parsed gap scores: {'tone': 2, 'structure': 2, 'content': 4, 'visual': 2}

🔍 Generating prompt improvement suggestions...

Suggested Improvements:

1. Add "use a friendly, engaging tone with emojis" to instructions
2. Specify a structured format: introduction, features (bulleted), and benefits
3. Clarify that feature list should include icons/emojis for visual appeal
4. Instruct to highlight integration, input methods, and planning tools as key content points
5. Request use of markdown-style bold headings for organization

🔧 Applying suggestions to improve the prompt...

Improved prompt:

Write a friendly and engaging description of TaskFriend using emojis throughout. Structure your response with the following sections in markdown format:

**\*\*🌟 What Is TaskFriend?\*\***

A brief, welcoming introduction to TaskFriend in one paragraph.

**\*\*🌟 Key Features\*\***

List the main features in bullet points, each including a relevant emoji/icon for visual appeal. Be sure to highlight:

- Integration with external calendars (e.g., Google Calendar, Outlook)
- Input methods (voice, text, email)
- Daily summaries and weekly planning tools
- Task prioritization and deadline tracking

**\*\*🎯 Why You'll Love It\*\***

Describe the top benefits in a few concise sentences—how TaskFriend helps users stay organized and productive.

Keep the tone upbeat, conversational, and inviting!

Write a friendly and engaging description of TaskFriend using emojis throughout. Structure your response with the following sections in markdown format:

**\*\*🌟 What Is TaskFriend?\*\***

A brief, welcoming introduction to TaskFriend in one paragraph.

**\*\*🌟 Key Features\*\***

List the main features in bullet points, each including a relevant emoji/icon for visual appeal. Be sure to highlight:

- Integration with external calendars (e.g., Google Calendar, Outlook)
- Input methods (voice, text, email)
- Daily summaries and weekly planning tools
- Task prioritization and deadline tracking

**\*\*🎯 Why You'll Love It\*\***

Describe the top benefits in a few concise sentences—how TaskFriend helps users stay organized and productive.

Keep the tone upbeat, conversational, and inviting!

=====  
Iteration 2  
=====

**\*\*🌟 What Is TaskFriend?\*\***

Hey there! 🙌 Meet **\*\*TaskFriend\*\***—your cheerful, always-there buddy for getting things done 🌈 Whether you're juggling work, side hustles, or life's little to-dos, TaskFriend turns chaos into calm with a smile 😊 It's like having a personal assistant who *actually* listens (and never judges your midnight snack shopping list 🍕).

**\*\*🌟 Key Features\*\***

- 🔄 **\*\*Seamless Calendar Sync\*\***: Effortlessly connects with Google

Calendar & Outlook so all your plans live in harmony 📅 17

- 🗣️ **\*\*Talk, Type, or Email It\*\***: Add tasks by voice 🗣️, text 📝, or even email 📧 –because inspiration strikes when you least expect it!
- 🇮🇹 **\*\*Daily Digests & Weekly Prep\*\***: Wake up to a sunny summary of your day ☀️ and prep for success with smart weekly planning 📅
- ⌚ **\*\*Smart Prioritization\*\***: Automatically sorts what matters most 📈 and gently nudges you before deadlines sneak up! ⏰

**\*\*🎯 Why You'll Love It\*\***

Say goodbye to sticky notes everywhere 📌 and hello to peace of mind 💡 TaskFriend helps you stay focused, reduce stress, and actually enjoy checking things off. With gentle reminders and joyful organization, you'll accomplish more—while still leaving time for coffee breaks ☕ and cat videos 🐱💖 Life's busy, but with TaskFriend? You've totally got this! 💪 ✨

Write a friendly and engaging description of TaskFriend using emojis throughout. Structure your response with the following sections in markdown format:

**\*\*🌟 What Is TaskFriend?\*\***

A brief, welcoming introduction to TaskFriend in one paragraph.

**\*\*🌟 Key Features\*\***

List the main features in bullet points, each including a relevant emoji/icon for visual appeal. Be sure to highlight:

- Integration with external calendars (e.g., Google Calendar, Outlook)
- Input methods (voice, text, email)
- Daily summaries and weekly planning tools
- Task prioritization and deadline tracking

**\*\*🎯 Why You'll Love It\*\***

Describe the top benefits in a few concise sentences—how TaskFriend helps users stay organized and productive.

Keep the tone upbeat, conversational, and inviting!

Generated Response (sample):

**\*\*🌟 What Is TaskFriend?\*\***

Hey there! 🙌 Meet **\*\*TaskFriend\*\***—your cheerful, always-there buddy for getting things done 🌈 Whether you're juggling work, side hustles, or life's little to-dos, TaskFriend turns chaos into calm with a smile 😊 It's like ha...

```
{
 "tone": 5,
 "structure": 4,
 "content": 5,
 "visual": 5
}
```

✅ Parsed gap scores: {'tone': 5, 'structure': 4, 'content': 5, 'visual':

5}

✅ Evaluation passed. Output quality is sufficient.

You can see that the automated evaluation + meta prompting framework is fast and efficient. However, creating a framework might take some time, but this serves as a good starting point.

## Grading responses: Scaling up your operation

Grading responses is essential when scaling AI applications. Manual review doesn't scale beyond a few examples, so we need automated, consistent evaluation. By defining clear metrics — such as tone, structure, content completeness, and visual formatting — and using an LLM as a grader, we can score outputs quantitatively. This enables A/B testing of prompts, monitoring of model performance over time, and integration into CI/CD pipelines for AI features. The key is to design grader prompts that are objective, repeatable, and aligned with user experience goals — turning qualitative expectations into measurable KPIs.

The idea is similar to what we did for **gap analysis**:

1. Identify the metrics you want to measure, for example:
  - Clarity
  - Accuracy
  - Helpfulness
2. Create a "Grader" function to help you generate quantitative scores.

```
from functions.grader_plot import plot_grader
import re

Test Responses
bad_response = """
TaskFriend is a productivity assistant designed to help users
efficiently manage their time.
It offers a range of features aimed at improving organization and time
management:

- Task Management
- Deadline Tracking
- Calendar Integration
- Planning & Summaries

Overall, TaskFriend serves as a comprehensive organizational tool that
helps users streamline
their workflow, reduce missed deadlines, and maintain better control over
their daily and
long-term tasks.
"""

ok_response = """
👋 I'm TaskFriend, your personal productivity partner – nice to meet
```

you! 🎨

I'm here to help you take control of your tasks, calendars, and notes – all in one smart, easy-to-use place. Let's make sure you're always working on what matters most, without the stress.

Here's how I can help boost your productivity:

### ### 📅 Smart Scheduling

Sync seamlessly with **Google Calendar** and **Outlook**, so your schedule is always up-to-date and at your fingertips.

### ### 📝 Flexible Task Entry

Add tasks using:

- ✍️ Text
- 🗣️ Voice input
- 📧 Email integration

Whatever fits your style best!

### ### 🔍 Automatic Prioritization

Let me sort your to-dos by **due date** and **importance**, so you know exactly where to focus each day.

### ### 🧠 Daily & Weekly Planning Tools

Start your day strong with a clear plan and get ahead on the week – I'll help you stay strategic and organized.

"""

```
Re-use previous reference answer, exact match
perfect_response = reference_answer
```

```
Grade Response Quality
```

```
def grade_response_detailed(response_to_grade,
 reference=reference_answer):
```

```
 grader_prompt = f"""
```

```
 [Role] You are a prompt evaluation assistant.
```

```
 [Task] Compare the generated response to the reference answer. Be
 sensitive to improvements.
```

```

 [Instructions] Maximum score for each metric is 5, while the minimum
 is 1
```

```

 [Reference answer]
 {reference}
```

```

 [Generated response]
 {response_to_grade}
```

```

 [Output Format]
```

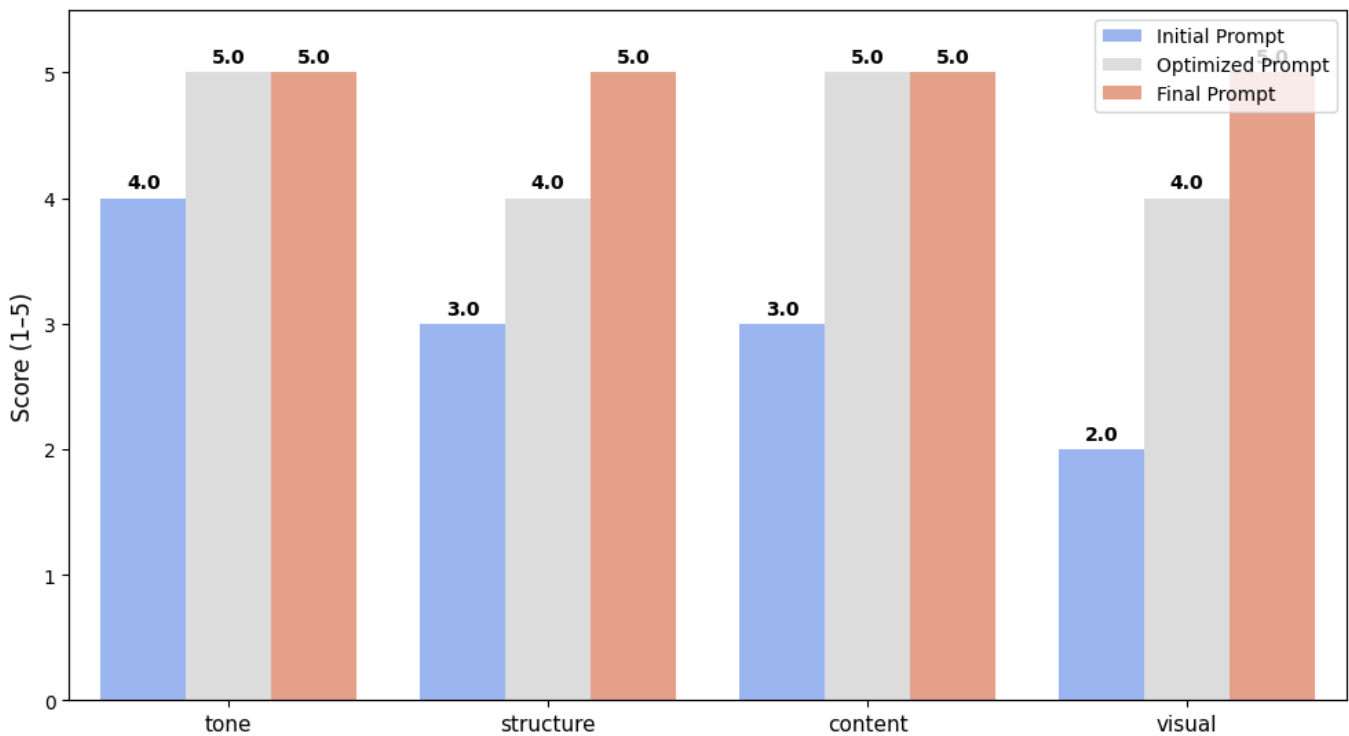
```
 Return only a JSON object like:
```

```
 {{
 "tone": int,
```



```
"visual": 5
}
```

Prompt Evolution: Response Quality Over Iterations



## What's Next?

### Quiz yourself!

► 1. What is the primary purpose of a prompt template in a RAG system?

- A) To store retrieved documents
- B) To define the AI's personality
- C) To instruct the LLM how to use retrieved context to answer the query
- D) To replace the system prompt

View answer →

✅ **Correct answer:** C) To instruct the LLM how to use retrieved context to answer the query

📝 **Explanation :**

- The prompt template structures the input to the LLM by combining the retrieved context and user query, ensuring the model grounds its response in the provided data rather than relying on prior knowledge.

## Takeaways

- **What makes a good prompt**

- **System prompts are the AI's job description** — they define role, goal, audience, format, and guardrails.
- **A strong prompt prevents drift** — without clear instructions, LLMs default to generic or off-topic responses.
- **Use the R-G-C-A-F-G framework:**
  - **Role:** Who the AI is (e.g., "TaskFriend, a productivity assistant")
  - **Goal:** What it should achieve (e.g., "help users prioritize tasks")
  - **Context:** What data it can use (e.g., "via RAG from task lists")
  - **Audience:** Who it's speaking to (e.g., "busy professionals")
  - **Format:** How it should respond (e.g., "bullet points, no markdown")
  - **Guardrails:** What it must avoid (e.g., "no roleplay, no opinions")
- **Prompt templates are different** — they govern how the LLM uses **retrieved context**, not its identity.

- **Combining context and queries with prompt templates**

- **Prompt templates are the engine of RAG** — they tell the LLM how to use retrieved documents to answer questions.
- **They enforce grounding** — phrases like "Given the context information and not prior knowledge" reduce hallucinations.
- **They standardize input structure** — making prompts predictable and testable.
- **Well-designed templates include:**
  - Clear separation (e.g., `---`) between context and query
  - Instructions to ignore prior knowledge
  - Output format rules (e.g., JSON, bullet points)
- **They are not foolproof** — user queries can override template instructions, so design with awareness of limitations.

- **Engineering smarter prompt templates**

- **Formatting improves clarity** — use `#headers`, `---`, `[brackets]`, and `**bold**` to guide the model.
- **Emojis and visual cues enhance UX** — they make outputs more engaging and scannable.
- **Structured output is critical** — prompt templates can enforce JSON, tables, or code, making responses easier to parse in production.
- **Prompt templates are developer tools** — they improve consistency, but aren't firewalls against user input.
- **They make RAG reliable** — without them, models may ignore context or fall back on training data.

- **General prompt engineering techniques**

- **Role/Persona prompting** changes tone and depth — e.g., "Explain like I'm 5" vs. "As a senior engineer."

- **Few-shot learning** uses examples to teach format and style — even one example dramatically improves output consistency.
  - **Chain-of-thought (CoT)** improves reasoning — asking models to “show your work” often leads to better answers.
  - **Reasoning models (e.g., Qwen-plus with thinking mode)** go further — they auto-generate internal reasoning before responding.
  - **Avoid CoT prompting on reasoning models** — it can cause infinite loops; the model already reasons internally.
- 
- **Meta prompting & automated evaluation**
    - **Meta-prompting** means “prompting about prompting” — using an LLM to critique and improve your prompts.
    - **It accelerates iteration** — instead of manual trial-and-error, you automate the optimization loop.
    - **Gap analysis** compares outputs to a reference answer using an LLM-as-a-judge.
    - **Automated grading** uses scoring rubrics (tone, structure, content, visual) to quantify quality.
    - **This enables CI/CD for prompts** — test, score, and deploy prompt updates like code, ensuring consistent quality at scale.