LARGE LANGUAGE MODELS

Design Smarter Prompts and Boost Your LLM Output: Real Tricks from an AI Engineer's Toolbox

Not just what you ask, but how you ask it. Practical techniques for prompt engineering that deliver

Ugo Pradère

Jun 12, 2025 9 min read

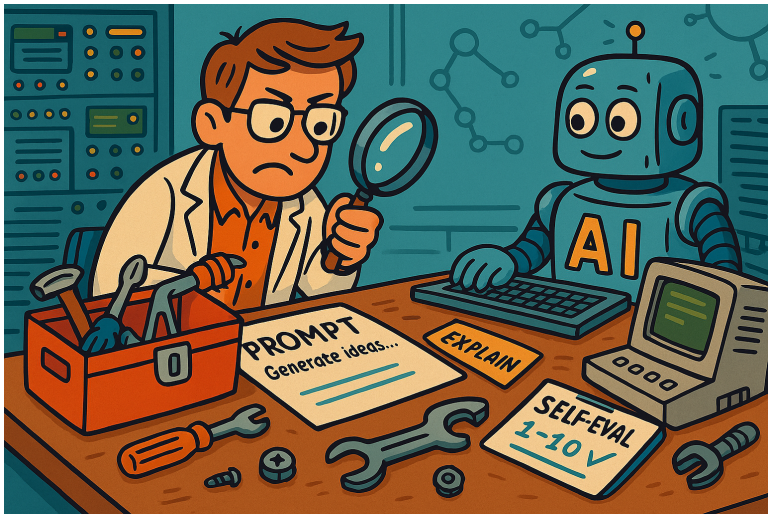

Image generated by the author with Dall-E

Prompting is easy but **good prompting** that provides **efficient and reliable outputs** is not. As language models grow in capability and versatility, getting high quality results depends more on **how you ask** the model than the model itself. That's where prompt engineering comes in, not as a theoretical exercise, but as a **day-by-day practical built-in experience** in production environments, with thousands of calls every day.

In this article, I'm sharing **five practical prompt engineering techniques** I use almost every day to build stable and reliable, high-performing AI workflows. They are not just tips I've read about but methods I've tested, refined, and relied on across real-world use cases in my work.

Some might seem counterintuitive, others surprisingly simple, but all of them have made a real difference in my proficiency to get the results I

expect from LLMs. Let’s dive in.

Tip 1 – Ask the LLM to write its own prompt

This first technique might feel **counterintuitive**, but it’s one I use all the time. Rather than trying to craft the perfect prompt from the start, I usually begin with a rough outline of what I want , then I ask the LLM to **refine the ideal prompt for itself**, based on additional context I provide. This **co-construction strategy** allows for the fast production of very **precise and effective prompts**.

The overall process is often composed of three steps:

- Start with general structure explaning tasks and rules to follow
- Iterative evaluation/refinement

of the prompt to match the desired result

- Iterative integration of edge cases or specific needs

Once the LLM proposes a prompt, I run it on a few **typical examples**. If the results are off, I don't just tweak the prompt manually. Instead, I ask the LLM to do so, asking specifically for **a generic correction**, as LLMs tends to patch things in a too-specific way otherwise. Once I obtain the desired answer for the 90+ percent cases, I generally run it on a **batch of input data** to analyse the edges cases that needs to be addressed. I then submit the problem to the LLM explaining the issue while submiting the input and ouput, to iteratively tweak the prompts and obtain the desired result.

A good tip that generally helps a lot is **to require the LLM to ask questions** before proposing prompt modifications to insure it fully

understand the need.

So, why does this work so well?

a. It's immediately better structured.
Especially for complex tasks, the LLM helps structure the problem space in a way that is both **logical** and **operational**. It also helps me **clarify my own thinking**. I avoid getting bogged down in syntax and stay focused on solving the problem itself.

b. It reduces contradictions.
Because the LLM is translating the task into its « own words », it's far more likely to detect ambiguity or contradictions. And when it does, it often asks for clarification before proposing a cleaner, conflict-free formulation. After all, who better to phrase a message than **the one who is meant to interpret it**?

Think of it like communicating with a human: a significant portion of

miscommunication comes from differing interpretations. The LLM finds sometimes something unclear or contradictory that I thought was perfectly obvious... and at the end, it is the one doing the job, so it's *its* **interpretation that matters, not mine.**

c. It generalizes better.
Sometimes I struggle to find a clear, abstract formulation for a task. The LLM is surprisingly good at this. It spots the pattern and produces a generalized prompt that's more scalable and robust to what I could produce myself.

Tip 2 – Use self-evaluation

The idea is simple, yet once again, very powerful. The goal is to **force the LLM to self-evaluate the quality of its answer** before outputting it. More specifically, I ask it **to rate its own answer** on a predefined scale, for instance, from 1 to 10. If the score is below a certain threshold

(usually I set it at 9), I ask it to either **retry** or **improve** the answer, depending on the task. I sometimes add the concept of “if you can do better” to avoid an endless loop.

In practice, I find it fascinating that an LLM tends to behave similarly to humans: it often goes for **the easiest answer rather than the best** one. After all, LLMs are trained on human produced data and are therefore meant to replicate the answer patterns. Therefore, giving it an **explicit quality standard** helps significantly improve the final output result.

A similar approach can be used for a **final quality check focused on rule compliance**. The idea is to ask the LLM to review its answer and confirm whether it followed a specific rule or all the rules before sending the response. This can help improve answer quality, especially when one rule tends to be skipped sometimes. However, in my

experience, this method is a bit less effective than asking for a self-assigned quality score. When this is required, it probably means your prompt or your AI workflow needs improvement.

Tip 3 – Use a response structure plus a targeted example combining format and content

Using examples is a well-known and powerful way to improve results… **as long as you don't overdo it**. A well-chosen example is indeed often more helpful than many lines of instruction.

The response structure, on the other hand, helps define exactly how the output should look, especially for technical or repetitive tasks. It avoids surprises and keeps the results consistent.

The example then complements that structure by showing how to

fill it with processed content. **This « structure + example » combo tends to work nicely**.

However, examples are often **text-heavy**, and using too many of them can **dilute the most important rules** or lead to them being followed less consistently. They also increase the number of tokens, which can cause side effects.

So, use examples wisely: one or two well-chosen examples that cover most of your essential or edge rules are usually enough. Adding more may not be worth it. It can also help to add a **short explanation** after the example, justifying why it matches the request, especially if that's not really obvious. I personally rarely use negative examples.

I usually give one or two positive examples along with a general structure of the expected output. Most of the time I choose XML tags

like `<open_tag></close_tag>`. Why? Because it's **easy to parse** and can be directly used in information systems for post-processing.

Giving an example is especially useful when the structure is nested. It makes things much clearer.

```
## Here is an example

Expected Output :

<items>
    <item>
        <sub_item>
            <sub_sub_item>
                My sub sub item 1 te
            </sub_sub_item>
            <sub_sub_item>
                My sub sub item 2 te
            </sub_sub_item>
        </sub_item>
        <sub_item>
            My sub item 2 text
        </sub_item>
        <sub_item>
            My sub item 3 text
        </sub_item>
    </item>
    <item>
        <sub_item>
            My sub item 1 text
```

```
            </sub_item>
            <sub_item>
                <sub_sub_item>
                    My sub sub item 1 te
                </sub_sub_item>
            </sub_item>
        </item>
    </items>

    Explanation :

    Text of the explanation
```

Tip 4 – Break down complex tasks into simple steps

This one may seem obvious, but it's essential for keeping answer quality high when dealing with complex tasks. The idea is **to split a big task into several smaller, well-defined steps**.

Just like the human brain struggles when it has to multitask, LLMs tend to produce lower-quality answers when the task is too broad or involves too many different goals at once. For example, if I ask you to calculate 125 + 47, then 256 − 24,

and finally 78 + 25, one after the other, this should be fine (hopefully :)). But if I ask you to give me the three answers in a single glance, the task becomes more complex. **I like to think that LLMs behave the same way**.

So instead of asking a model to do everything in one go like proofreading an article, translating it, and formatting it in HTML, I prefer to break the process into two or three simpler steps, each handled by a separate prompt.

The main downside of this method is that it adds some complexity to your code, especially when passing information from one step to the next. But modern frameworks like **LangChain**, which I personally love and use whenever I have to deal with this situation, make this kind of sequential task management very easy to implement.

Tip 5 – Ask the LLM for

explanation

Sometimes, it's hard to understand *why* the LLM gave an unexpected answer. You might start making guesses, but the easiest and most reliable approach might simply **to ask the model to explain its reasoning**.

Some might say that the predictive nature of LLM does not allow LLM to actually explain their reasonning because it simply does *not* reason but my experience shows that :

1- most of the time, it will effectively outline **a logical explanation** that produced its response

2- making prompt modification according to this explanation generally corrects the incorrect LLM answering.

Of course, this is not a proof that the LLM is actually reasoning, and *it is not my job to prove this*, but I can

state that this solution works in pratice very well for prompt optimization.

This technique is especially helpful during development, pre-production, or even the first weeks after going live. In many cases, it's difficult to anticipate all possible edge cases in a process that relies on one or several LLM calls. Being able to understand *why* the model produced a certain answer helps you design the most precise fix possible, one that solves the problem without causing unwanted side effects elsewhere.

Conclusion

Working with LLMs is a bit like working with a genius intern, insanely fast and capable, but often messy and going in every direction if you do not tell clearly what you expect. Getting the best out of an intern requires clear instructions and a bit of management

experience. The same goes with LLMs for which **smart prompting and experience make all the difference**.

The five techniques I've shared above are not "magic tricks" but **practical methods** I use daily to go **beyond generic results** obtained with standard prompting technique and get the high quality ones I need. They consistently help me turn correct outputs into great ones. Whether it's co-designing prompts with the model, breaking tasks into manageable parts, or simply asking the LLM *why* a response is what it is, **these strategies have become essential tools in my daily work to craft the best AI workflows I can**.

Prompt engineering is not just about writing clear and well organized instructions. It's about understanding *how* **the model interprets** them and designing your approach accordingly. Prompt

engineering is in a way like a sort of art, one of nuance, finesse, and personal style, where no two prompt designers write quite the same lines which results in different outcomes in term of strenght and weaknesses. Afterall, one thing stays true with LLMs: **the better you talk to them, the better they work for you**.

WRITTEN BY

Ugo Pradère

See all from Ugo Pradère

Topics:

Editors Pick | Llm | Machine Learning | Prompt Design | Prompt Engineering

Share this article:

Related Articles

ARTIFICIAL INTELLIGENCE

What Do Large Language Models “Understand”?

A deep dive on the meaning of understanding and how it applies to LLMs

Tarik Dzekman

August 21, 2024 31 min read

LARGE LANGUAGE MODELS

Deep Dive into LlaMA 3 by Hand ✍️

Explore the nuances of the transformer architecture behind Llama 3 and its prospects for the...

Srijanie Dey, PhD

May 3, 2024 12 min read

LARGE LANGUAGE MODELS

Deep Dive into Transformers by Hand ✍

Explore the details behind the power of transformers

Srijanie Dey, PhD

April 12, 2024 6 min read

DATA SCIENCE

Deep Dive into Sora's Diffusion Transformer (DiT) by Hand ✍

Explore the secret behind Sora's state-of-the-art videos

Srijanie Dey, PhD

April 2, 2024 13 min read

LARGE LANGUAGE MODELS

UniFliXsg: AI-Powered Undergraduate Program Recommendations for Singapore Universities

How could AI suggest your majors?

Phanuphat (Oad) Srisukhawasu

August 14, 2024 8 min read

ARTIFICIAL INTELLIGENCE

Beware of Unreliable Data in Model Evaluation: A LLM Prompt Selection case study with Flan-T5

You may choose suboptimal prompts for your LLM (or make other suboptimal choices via model...

Chris Mauck

June 16, 2023 12 min read

ARTIFICIAL INTELLIGENCE

Semantic Search Engine for Emojis in

50+ Languages Using AI 😊🌍🚀

If you are on social media like Twitter or LinkedIn, you have probably noticed that...

Badr Alabsi, PhD

July 17, 2024 14 min read

Your home for data science and Al. The world's leading publication for data science, data analytics, data engineering, machine learning, and artificial intelligence professionals.

Subscribe to Our Newsletter