Department for
Science, Innovation
& Technology

Government
Digital Service

# AI Insights

## Prompt Engineering

## Querying a large language model

Asking a question of a chatbot or Large Language Model (LLM), is quite like an intelligent search.  Instead of entering simple search terms into your browser, and getting a large number of website hits, we can instead ask questions of an LLM and get coherent responses across a surprisingly wide range of subjects.

Most people have interacted directly with LLMs in the form of website chatbots, AI assistants, or coding assistants.  We ask questions and get answers.  Normally, we can ask follow-up questions to guide the LLM in the direction we would like to investigate.

Even without such direct communication there is a strong likelihood that you have interacted with an LLM, albeit indirectly, simply by using everyday services and applications online.

When these services and applications communicate on your behalf with backend LLMs, they do so by wrapping queries, data and instructions inside a prompt.  This prompt is then sent to the LLM where it is processed and a response is returned.

## What is a prompt?

While, generally speaking, a prompt is any request made of a large language model, typically what we mean as a prompt is a carefully designed set of instructions and context that tells an AI system how to perform a specific task or respond to a particular type of input.

A prompt is like a recipe for retrieving what we want from an LLM-based system.  The data, examples, and instructions we provide are like lists of ingredients.  The methodology we want the system to follow, and the steps we may provide help to create a finished dish, that is: the outcome we expect from the LLM.

## What is prompt engineering?

Prompt engineering is the process we follow in order to ensure that prompts are constructed in a safe and efficient manner.

To complete our recipe metaphor, prompt engineering is like having a conversation with a chef. The prompt is our request - it includes the dish we want and how we want it prepared. Just as a vague request like 'make something tasty' might result in unexpected dishes, an unclear request may lead to unexpected responses. The more specific we are - 'Please make a vegetarian lasagna with spinach and mushrooms, with the pasta cooked al dente' - the more likely we are to get exactly what you want.

We're asking an AI to process information, analyse data, invoke some behaviour or create content. And just like a chef needs clear, well-organised instructions to prepare a dish correctly, an AI needs well-structured prompts to deliver the results we're looking for.

### Safety First

Safety comes first at all times.  One of the most important responsibilities of successful Prompt Engineering is to ensure that queries which are sent to the Large Language Model do no harm.

Prompt Engineering helps to ensure that queries are sanitised, for example:

- They are not attempting to bypass or remove safety measures
- They contain no nefarious commands or instructions,
- They are not attempting to invoke unauthorised behaviour
- They are not attempting to access unauthorised data

In practice this list is far more comprehensive, and is used in concert with many other safety features which provide guardrails around prompts and also around the responses returned.

Please take a look at our Prompt Risks Spotlight and LLM Safeguarding Insight for more information

Prompt engineering is used to direct the return of relevant responses for our queries.  If we are accessing a system to examine a user's order history, we would like to constrain the system away from anything irrelevant, or from returning irrelevant information in the response.

This is done through a variety of methods.  We typically include instructions in the prompt which help to constrain and to direct the LLM towards the target domain.  We can also tell the LLM how to respond if an answer cannot be found.  This is vital to avoid *hallucinations* in the response pipeline.

## What are hallucinations?

We hear a lot about confabulation or hallucinations in LLM-based systems.  This deserves some explanation.  For example, If we ask someone about a painting that's slightly out of their view - they can see parts of it clearly, but other parts are out-of-focus. When describing the blurry parts, they might fill in details that seem logical but aren't actually there. AI systems can do something similar - when asked about topics where their knowledge is incomplete or uncertain, they might generate responses that sound plausible and match learned patterns, but aren't actually true.

Without getting too technical, this arises because LLMs are mostly *probabilistic* models. They have learned their features, weights, and clusters from large swathes of data. They predict or project a response from these based on the input request.  The response is the model's estimate, or *best guess*, of the *most likely answer* based on what was asked and how it was asked, and how the model was initially trained and refined.  Sometimes that *best guess* is wrong, especially in specialised areas or if the prompt is vague or obscure.

There have been some well-documented cases of people blindly relying on the responses from LLMs.  We don't need to go into them here, save to say that the less refined the prompt, the absence of protective measures, and the more specialised the domain, the greater the likelihood of a blurred picture to derive from and therefore the increased potential for hallucination in the output.  It is our responsibility to check these responses and refine our prompts accordingly.

We also very often provide data in the prompt, and constrain the LLM to use the data we provide while generating the response.  This is widely used in commercial and public sector systems, as we mostly wish to make data and services available to our users.

Robust prompt engineering capabilities are as important as the selection of the AI model itself. It is one of the key mechanisms for translating requirements into reliable AI behaviours and ensuring AI systems remain controllable, predictable, and aligned with our objectives.

## How are prompts used in software systems?

In addition to people querying LLMs through prompts, computer systems also use prompts behind the scenes.  These prompts are done in an almost identical way to human interaction.  The software system will extend a prompt that has been tested and evaluated, usually by inserting some specific data, send it to the LLM, and parse the response.

Computer systems using LLMs have seen a dramatic increase in popularity as LLMs introduce novel techniques for manipulating data and behaviour.  This functionality was either unavailable before, or so cumbersome to create that the cost was prohibitive with limited, and often brittle, results.

These LLM-based systems must also protect themselves from abuse, and have specialised methods to support this.  They are also subject to the same Prompt Engineering requirements and constraints as humans.

### Expanding software capabilities through LLMs

LLMs have transformed software development. Instead of programming every possible scenario into rigid code paths, developers can rely on LLMs to interpret user requests, handle unexpected queries, or even manage tasks that previously required human judgement. For instance, a system could use an LLM to process natural language instructions—such as “Find all recent customer orders with issues in shipment”—and then decide the best way to fulfill that request within the application’s framework. With the help of prompt engineering, the LLM can be guided to focus only on relevant data or to follow certain rules, ensuring that its responses are both accurate and appropriate.

This new model of software interaction unlocks capabilities that were once too cumbersome or costly to implement. Tasks that previously demanded complex coding, manual oversight, or rule-based logic can now be achieved with minimal additional effort. This not only speeds up development but also makes applications more adaptive to change. If new requirements arise, developers can often adjust the underlying prompts rather than rebuilding entire systems, resulting in more fluid and responsive software.

## LLMs in agentic systems

One of the most exciting developments in this space is the use of LLMs to manage software agents. These agents are composed of specialised tools or functions within a broader application—each responsible for a specific type of task or service.

Unlike traditional software components that interact through rigid programming interfaces, these agents can describe themselves and their capabilities in natural language. For example, an agent might describe itself to the LLM by saying, “I can fetch and analyse customer order data,” or “I can translate text from English to French.”

When a user makes a request or when the system itself decides it needs a certain capability, the LLM examines the natural language descriptions of each available agent. Based on the provided prompt and context, the LLM then determines which agent’s services are most suitable to fulfill that request.

Essentially, the LLM acts as an orchestrator, using natural language instructions and descriptions to pick the right tool for the job. This approach makes it much simpler for different aspects of the system to talk to each other while also being more flexible, adaptive, and easier to maintain.

## LLM frameworks

In the early days of LLM integration, prompts were often just long strings of text: instructions, conditions, and bits of data all mashed together in a single, somewhat fragile script. This approach worked well enough for simple tasks, but has a tendency to become unwieldy. Large text prompts are prone to errors and inconsistencies. Changing a single detail often means carefully reviewing and editing large sections of text.

Developers and users alike quickly realised that as LLM-based systems became more important, they needed to move beyond this somewhat haphazard method.

Today, we are moving toward more structured and robust approaches. Prompt engineering is beginning to adopt best practices from traditional software engineering—such as organising instructions into smaller, modular parts; applying testing and evaluation frameworks; and implementing metrics to measure performance and reliability. Instead of an LLM prompt being a static block of text that never changes, it can be refined, tested, and improved over time.

For example, as these systems evolve, we use profiles or templates that specify what kind of information the LLM should rely on, what style of response it should provide, and how to handle uncertain inputs. These profiles can be evaluated against test scenarios—like a developer checking whether the LLM responds correctly when certain data is missing.

They can be monitored for consistency—ensuring that the LLM provides similar answers for similar questions over time. They can also be monitored for performance—tracking how quickly the LLM responds and how often it needs to consult an external agent.

Through these techniques, LLM-based applications become more predictable and trustworthy. Developers gain confidence that their systems will behave as intended, and users get more reliable, accurate results. Over time, as the field matures, we can expect these agentic frameworks and prompt engineering practices to continue improving.

The result is a new generation of intelligent applications that can adapt to evolving requirements, integrate with multiple specialised agents, and maintain high standards of safety, consistency, and performance - while remaining accessible through the medium of natural language.

## Conclusion

Prompts are not just a new interface for humans to communicate with intelligent systems; they are also a powerful tool that software can use internally to make decisions, marshal resources, and deliver enhanced capabilities. By blending the flexibility of natural language with the rigor of engineering practices, modern software developers and organisations are creating dynamic, responsive, and safe systems that leverage the strengths of LLMs. This evolution promises to reshape how we build, manage, and interact with applications in the years to come.