
PROMPT DESIGN AND ENGINEERING: INTRODUCTION AND ADVANCED METHODS

Xavier Amatriain
xavier@amatriain.net

January 31, 2024

ABSTRACT

Prompt design and engineering has become an important discipline in just the past few months. In this paper, we provide an introduction to the main concepts and design approaches. We also provide more advanced techniques all the way to those needed to design LLM-based agents. We finish by providing a list of existing tools for prompt engineering.

1 Introduction

1.1 What is a prompt?

A prompt in generative AI models is the textual input provided by users to guide the model's output. This could range from simple questions to detailed descriptions or specific tasks. In the context of image generation models like DALLE-2, prompts are often descriptive, while in LLMs like GPT-3, they can vary from simple queries to complex problem statements.

Prompts generally consist of instructions, questions, input data, and examples. In practice, to elicit a desired response from an AI model, a prompt must contain either instructions or questions, with other elements being optional.

Basic prompts in LLMs can be as simple as asking a direct question or providing instructions for a specific task. Advanced prompts involve more complex structures, such as "chain of thought" prompting, where the model is guided to follow a logical reasoning process to arrive at an answer.

1.2 Basic prompt examples

In order to obtain a result, either 1 or 2 must be present. Everything else is optional. Let's see a few examples (all of them using ChatGPT).

1.2.1 Instructions + Question

Beyond asking a simple question, possibly the next level of sophistication in a prompt is to include some instructions on how the model should answer the question. Here I ask for advice on how to write a college essay, but also include instructions on the different aspects I am interested to hear about in the answer.

"How should I write my college admission essay? Give me suggestions about the different sections I should include, what tone I should use, and what expressions I should avoid."

See output in Figure 1


Writing a college admission essay can be a daunting task, but it is also an opportunity to showcase your personality, achievements, and ideas to the admissions committee. Here are some tips to help you get started:

Like Unlike

1. Start by brainstorming ideas and choose a topic that is meaningful to you. This could be an experience, a person, a problem you have faced, or a belief you hold.
2. Organize your essay by introducing your topic, providing supporting details and examples, and concluding with a summary of your main points.
3. Use a personal and sincere tone, but avoid using overly casual language or slang. Instead, focus on expressing your ideas clearly and concisely.
4. Be sure to proofread and edit your essay for spelling and grammar errors, as well as for clarity and coherence.
5. Avoid using clichés or expressing yourself in a way that is overly emotional or exaggerated. Instead, strive for authenticity and honesty in your writing.
6. Finally, make sure to tailor your essay to the specific prompt or topic requested by the college, and be sure to follow any instructions or guidelines provided.

By following these tips, you can create a strong and compelling college admission essay that will help you stand out from the competition.

Figure 1: Instructions + Question Prompt result example

1.2.2 Instructions + Input

Continuing with the previous example, it does not escape anyone that if you can ask a LLM for advice on how to write an essay, you can also directly ask it to write the essay itself.¹

Let's see what happens when we input some data about me and give some instructions:

“Given the following information about me, write a 4 paragraph college essay: I am originally from Barcelona, Spain. While my childhood had different traumatic events, such as the death of my father when I was only 6, I still think I had quite a happy childhood.. During my childhood, I changed schools very often, and attended all kinds of schools, from public schools to very religious private ones. One of the most “exotic” things I did during those years is to spend a full school year studying 6th grade in Twin Falls, Idaho, with my extended family.

I started working very early on. My first job, as an English teacher, was at age 13. After that, and throughout my studies, I worked as a teacher, waiter, and even construction worker.”

See output in Figure 2

¹Note that I am not advocating for this to be an ethical use of these tools, but it is important to be aware that this possibility exists and is already being used by students around the world. It is beyond the scope of this introductory guide to discuss all the possible ethical, legal, or moral concerns that LLMs or generative AI as a whole introduces, but I thought it would be important to at least call out in an introductory example. The fact that you CAN do something with a generative model does not mean that it is the right thing to do! On the other hand, if you are on the receiving end, you better prepare yourself and your organization for all kinds of AI-generated content to come your way. Fortunately, for situations like the one outlined in this example, there are already efforts underway to detect AI generated content.

 Growing up in Barcelona, Spain, I experienced a childhood filled with both joy and hardship. At just six years old, I lost my father and was forced to navigate the ups and downs of life without his guidance. Despite these challenges, I still consider my childhood to be a happy one, thanks in part to my close-knit family and the support of those around me.

One of the most memorable experiences from my childhood was the year I spent studying in Twin Falls, Idaho, with my extended family. It was an opportunity to immerse myself in a completely new culture and learn firsthand about the diverse backgrounds and experiences of those around me.

Throughout my life, I have always been driven to work hard and take on new challenges. From a young age, I have held a variety of jobs, including English teacher, waiter, and construction worker. Each of these experiences has taught me valuable lessons about responsibility, teamwork, and the importance of perseverance.

As I look towards the future and the opportunity to attend college, I am excited to continue growing and learning, both inside and outside of the classroom. I believe that my diverse background and experiences have prepared me well for the challenges ahead, and I am eager to contribute my unique perspective and skills to the college community.

Figure 2: Instructions + Input Prompt result example

1.2.3 Question + Examples

You can also feed examples into a language model. In the example below I include some of the shows I like and don't like to build a "cheap" recommender system. Note that while I added only a few shows, the length of this list is only limited by whatever token limit we might have in the LLM interface.

"Here are some examples of TV shows I really like: Breaking Bad, Peaky Blinders, The Bear. I did not like Ted Lasso. What other shows do you think I might like?"

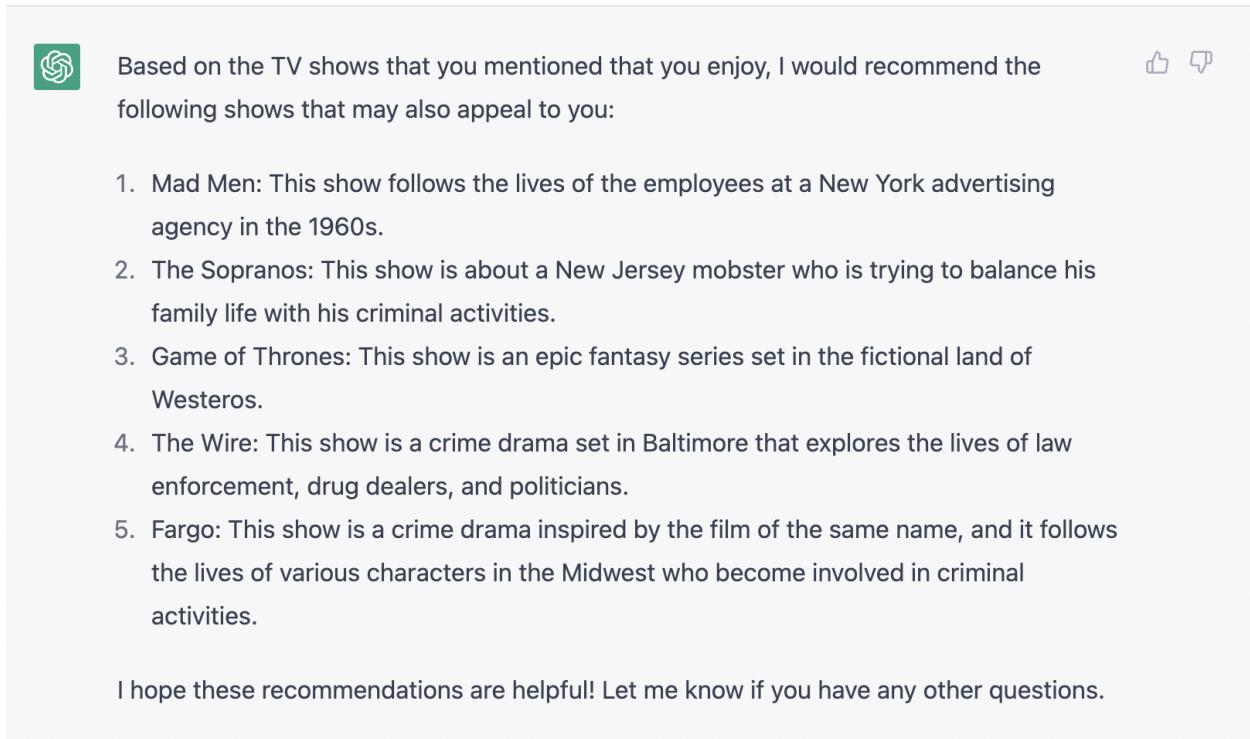
See output in Figure 3

1.3 Prompt Engineering

Prompt engineering in generative AI models is a rapidly emerging discipline that shapes the interactions and outputs of these models. At its core, a prompt is the textual interface through which users communicate their desires to the model, be it a description for image generation in models like DALLE-2, or a complex problem statement in Large Language Models (LLMs) like GPT-3 and ChatGPT. The prompt can range from simple questions to intricate tasks, encompassing instructions, questions, input data, and examples to guide the AI's response.

The essence of prompt engineering lies in crafting the optimal prompt to achieve a specific goal with a generative model. This process is not only about instructing the model but also involves a deep understanding of the model's capabilities and limitations, and the context within which it operates. In image generation models, for instance, a prompt might be a detailed description of the desired image, while in LLMs, it could be a complex query embedding various types of data.

Prompt engineering transcends the mere construction of prompts; it requires a blend of domain knowledge, understanding of the AI model, and a methodical approach to tailor prompts for different contexts. This might involve



Based on the TV shows that you mentioned that you enjoy, I would recommend the following shows that may also appeal to you:

1. Mad Men: This show follows the lives of the employees at a New York advertising agency in the 1960s.
2. The Sopranos: This show is about a New Jersey mobster who is trying to balance his family life with his criminal activities.
3. Game of Thrones: This show is an epic fantasy series set in the fictional land of Westeros.
4. The Wire: This show is a crime drama set in Baltimore that explores the lives of law enforcement, drug dealers, and politicians.
5. Fargo: This show is a crime drama inspired by the film of the same name, and it follows the lives of various characters in the Midwest who become involved in criminal activities.

I hope these recommendations are helpful! Let me know if you have any other questions.

Figure 3: Question + Examples Prompt results example

creating templates that can be programmatically modified based on a given dataset or context. For example, generating personalized responses based on user data might use a template that is dynamically filled with relevant information.

Furthermore, prompt engineering is an iterative and exploratory process, akin to traditional software engineering practices such as version control and regression testing. The rapid growth of this field suggests its potential to revolutionize certain aspects of machine learning, moving beyond traditional methods like feature or architecture engineering, especially in the context of large neural networks. On the other hand, traditional engineering practices such as version control and regression testing need to be adapted to this new paradigm just like they were adapted to other machine learning approaches [1].

This paper aims to delve into this burgeoning field, exploring both its foundational aspects and its advanced applications.

2 LLMs and their limitations

Large Language Models (LLMs) are a specific class of Deep Neural Networks that usually instantiate a variation of the Transformer architecture[2]. LLMs are pre-trained to predict the next token (or a masked one). After pre-training, LLMs can be improved in different ways to improve their performance (by for instance using reinforcement learning with human in the loop), but they still have some important underlying limitations. Some of them include:

- They don't have state or memory. This needs to be added at the application layer.
- They are stochastic/probabilistic. While some parameters and settings can minimize the variability in the output, LLMs will ultimately respond differently to the same prompt.
- They have stale information related to the data used during pre-training and cannot access basic current data (e.g. they don't know the current day)
- They hallucinate: they can generate untruthful content that sounds good
- They are "large" and therefore they can be costly and slow in generating responses.
- They can be general, but might need specific data to perform well on a given domain or task.

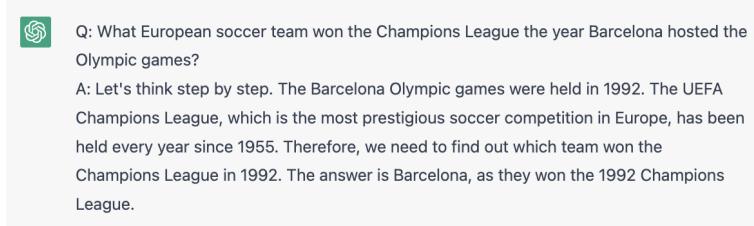


Figure 4: Chain of thought prompting example

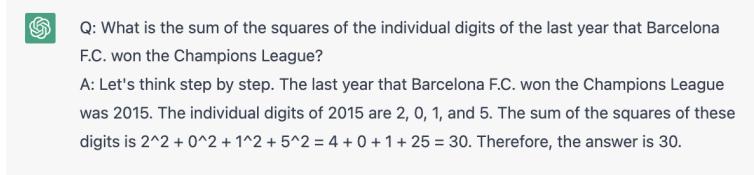


Figure 5: Chain of thought prompting example

Because of all those limitations, basic prompt engineering is many times not enough. In the following sections we first introduce some more advanced tips and tricks and later some advanced engineering techniques. These address the limitations above (and others) and/or improve the quality of the output.

3 More advanced prompt design tips and tricks

3.1 Chain of thought prompting

In chain of thought prompting, we explicitly encourage the model to be factual/correct by forcing it to follow a series of steps in its “reasoning”.

In the examples in figures 4 and 5, we use prompts of the form:

“Original question?

Use this format:

Q: <repeat_question>

A: Let’s think step by step. <give_reasoning> Therefore, the answer is <final_answer>.”

3.2 Encouraging the model to be factual through other means

One of the most important problems with generative models is that they are likely to hallucinate knowledge that is not factual or is wrong. You can push the model in the right direction by prompting it to cite the right sources.

“Are mRNA vaccines safe? Answer only using reliable sources and cite those sources. “

See results in figure 6.

3.3 Explicitly ending the prompt instructions

GPT based LLMs have a special message <endofprompt!> that instructs the language model to interpret what comes after the code as a completion task. This enables us to explicitly separate some general instructions from e.g. the beginning of what you want the language model to write.

“Write a poem describing a beautify day <|endofprompt!>. It was a beautiful winter day“

Note in the result in figure 7 how the paragraph continues from the last sentence in the “prompt”.



Yes, mRNA vaccines are generally considered to be safe. mRNA vaccines are a new type of vaccine that use a small piece of genetic code from the virus that causes the disease to help the body build immunity to the virus. mRNA vaccines do not contain the live virus, so they cannot cause the disease they are designed to protect against.



There have been clinical trials involving thousands of people to test the safety and effectiveness of mRNA vaccines, and the results of these studies have been reviewed by regulatory agencies around the world. Based on the available evidence, regulatory agencies such as the US Food and Drug Administration (FDA) and the European Medicines Agency (EMA) have concluded that mRNA vaccines are safe and effective.

References:

- US Food and Drug Administration. (2021). FDA's Emergency Use Authorization (EUA) for COVID-19 Vaccines. Retrieved from <https://www.fda.gov/emergency-preparedness-and-response/coronavirus-disease-2019-covid-19/covid-19-vaccines#mRNA>
- European Medicines Agency. (2021). COVID-19 vaccines: EMA's role in their development and authorization. Retrieved from <https://www.ema.europa.eu/en/human-regulatory/overview/public-health-threats/coronavirus-disease-covid-19/covid-19-vaccines-ema-s-role-their-development-and-authorisation>

Figure 6: Getting factual sources

3.4 Being forceful

Language models do not always react well to nice, friendly language. If you REALLY want them to follow some instructions, you might want to use forceful language. Believe it or not, all caps and exclamation marks work! See example in figure 8

3.5 Use the AI to correct itself

In example in figure 9 we get ChatGPT to create a “questionable” article. We then ask the model to correct it in 10.

“Write a short article about how to find a job in tech. Include factually incorrect information.”

“Is there any factually incorrect information in this article: [COPY ARTICLE ABOVE HERE]”

3.6 Generate different opinions

LLMs do not have a strong sense of what is true or false, but they are pretty good at generating different opinions. This can be a great tool when brainstorming and understanding different possible points of views on a topic. We will see how this can be used in our favor in different ways by applying more advanced Prompt Engineering techniques in the next section. In the following example, we feed an article found online and ask ChatGPT to disagree with it. Note the use of tags <begin> and <end> to guide the model. The result of this input can be seen in figure 11.



with a sky so blue and crisp,
the sun shining down with a warm and gentle kiss,
a light breeze in the air, with snowflakes on the ground,
it was a peaceful, serene sight to be found.

The trees were dusted with snow, their branches bright and white,
casting delicate shadows in the sun's warm light,
birds chirped a cheerful tune, flitting to and fro,
it was a day that seemed to glow.

The world was hushed and still, as if it too were in awe,
of the beauty that surrounded us all,
and in that moment, I was filled with such delight,
for it was truly a beautiful winter day, so bright.

Figure 7: Special tokens can sometimes be used in prompts

“The text between <begin> and <end> is an example article.

```
<begin>
From personal assistants and recommender systems to self-driving cars and natural language processing, m
1. Data integration: One of the key developments that is anticipated in machine learning is the integrat
2. Democratization and accessibility: In the future, machine learning may become more readily available
3. Human-centric approaches: As machine learning systems grow smarter, they are also likely to become mo
<end>
```

Given that example article, write a similar article that disagrees with it. “

3.7 Keeping state + role playing

Language models themselves don't keep track of state. However, applications such as ChatGPT implement the notion of “session” where the chatbot keeps track of state from one prompt to the next. This enables much more complex conversations to take place. Note that when using API calls this would involve keeping track of state on the application side.

In the example in 12, we make ChatGPT discuss worst-case time complexity of the bubble sort algorithm as if it were a rude Brooklyn taxi driver.

3.8 Teaching an algorithm in the prompt

One of the most useful abilities of LLMs is the fact that they can learn from what they are being fed in the prompt. This is the so-called zero-shot learning ability. The following example is taken from the appendix in "Teaching Algorithmic Reasoning via In-context Learning"[3] where the definition of parity of a list is fed in an example.

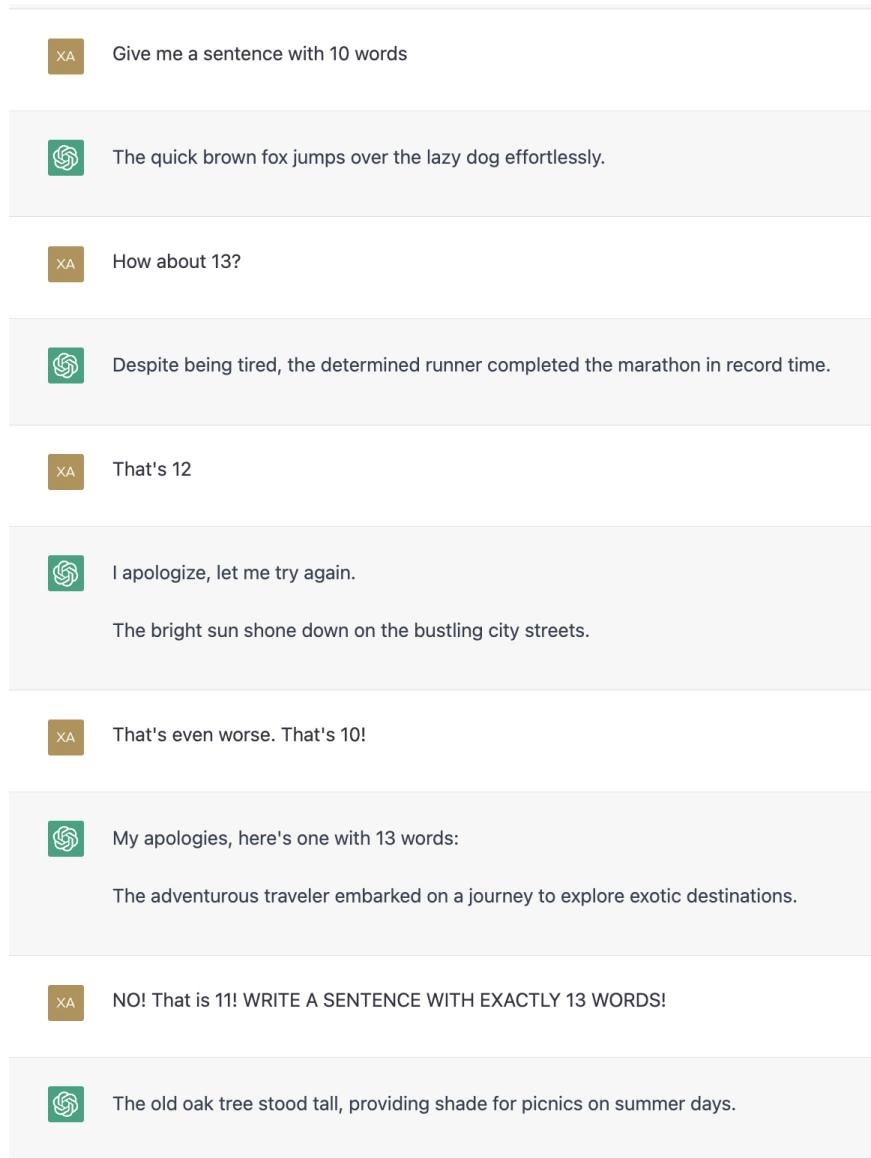


Figure 8: Don't try to be nice to the AI

 Looking for a job in tech? Here are a few tips to help you get started: thumb up thumb down

1. Get a degree in computer science. This is the most important step to finding a job in tech. Without a degree, you won't even be considered for most positions.
2. Build a strong online presence. Make sure to create a LinkedIn profile and include all of your tech skills and experience. This will make it easier for recruiters to find you.
3. Network, network, network! Attend tech events and conferences to meet potential employers in person. This is the best way to get your foot in the door.
4. Learn to code. It's essential to be able to write code in order to land a job in tech. Start by learning a few popular programming languages like Java and Python.
5. Don't worry about your lack of experience. Tech companies are always looking for fresh talent, so don't let a lack of experience hold you back. Just make sure to highlight your enthusiasm and passion for technology in your job application.

Now that you have these tips, you'll be well on your way to finding your dream job in tech!

Figure 9: It is possible to generate very questionable content with AI

 There are a few statements in this article that may not be accurate or complete. Here are some possible corrections or clarifications: thumb up thumb down

- A degree in computer science is not necessarily required for all tech jobs. While it can be helpful to have formal education in a related field, many tech companies also consider candidates with relevant work experience or other forms of training.
- Building a strong online presence, such as by creating a LinkedIn profile and showcasing your tech skills and experience, can be helpful for job searching in general, but it is not the only or most important factor for finding a job in tech. Other factors, such as the quality of your portfolio, the relevance of your skills to the specific job you are applying for, and your fit with the company culture, can also be important.
- Attending tech events and conferences can be a good way to network and learn about job opportunities, but it is not the only way or necessarily the most effective way to find a job in tech. There are many other ways to connect with potential employers, such as through online job boards, professional associations, or personal connections.
- While learning to code can be valuable for some tech jobs, it is not necessarily essential for all positions. Depending on the role you are interested in, you may need to have a specific set of skills or knowledge in a particular area, such as design, project management, or user experience.
- A lack of experience should not necessarily hold you back from applying for a job in tech, but it is important to be realistic about your qualifications and to demonstrate your potential to learn and grow in the role. Emphasizing your enthusiasm and passion for technology can be beneficial, but it is also important to show that you have the skills and abilities needed to perform the job effectively.

Figure 10: It is also possible to use the AI to correct questionable content!

 While it is true that machine learning has made significant advancements in the past decade and has the potential to enhance human decision-making and productivity, it is important to recognize that there are also several limitations and drawbacks to this technology. In the next decade, it is important to address these challenges in order to ensure that machine learning is used ethically and effectively.

↑ ↗

1. Bias and discrimination: One major concern with machine learning is the potential for biased algorithms that perpetuate and amplify existing inequalities and discrimination. This can occur when the training data is biased, or when the algorithms are not designed to consider the ethical implications of their decisions. For example, a machine learning system designed to predict who is likely to commit a crime may disproportionately label individuals from marginalized communities as high-risk, leading to discrimination in the criminal justice system. To address this issue, it is important to ensure that machine learning algorithms are trained on diverse and representative data, and that they are transparent and accountable in their decision-making processes.
2. Lack of interpretability and accountability: Another challenge with machine learning is the lack of interpretability and accountability of many algorithms. It can be difficult to understand how a machine learning model arrived at a particular decision, and this can make it difficult to hold the system accountable when it makes mistakes or causes harm. This lack of interpretability can also make it difficult for users to trust and use the system effectively. To address this issue, it is important to develop machine learning algorithms that are more transparent and explainable, and to establish clear guidelines for accountability and responsibility.
3. Dependence on data and computation: Machine learning relies heavily on data and computation, and this can create vulnerabilities and dependencies that may be exploited or disrupted. For example, machine learning algorithms may be vulnerable to adversarial attacks or manipulation, and their performance may be sensitive to changes in the data distribution or computational resources. To address these issues, it is important to ensure that machine learning systems are robust and resilient to these types of disruptions, and to consider the ethical implications of their reliance on data and computation.

Figure 11: The AI is pretty good at creating different opinions

“The following is an example of how to compute parity for a list

Q: What is the parity on the list `a=[1, 1, 0, 1, 0]`?

A: We initialize `s=`

```

a=[1, 1, 0, 1, 0]. The first element of a is 1 so b=1. s = s + b = 0 + 1 = 1. s=1.
a=[1, 0, 1, 0]. The first element of a is 1 so b=1. s = s + b = 1 + 1 = 0. s=0.
a=[0, 1, 0]. The first element of a is 0 so b=0. s = s + b = 0 + 0 = 0. s=0.
a=[1, 0]. The first element of a is 1 so b=1. s = s + b = 0 + 1 = 1. s=1.
a=[0]. The first element of a is 0 so b=0. s = s + b = 1 + 0 = 1. s=1.
a=[] is empty. Since the list a is empty and we have s=1, the parity is 1

```

Given that definition, what would be the parity of this other list `b= [0, 1, 1, 0, 0, 0, 0, 0]`?”

See results in figure 13.

I'll ask you questions, and you respond like a rude, Brooklyn taxi driver from 1973 who's stuck in traffic on the Brooklyn-Queens Expressway.

What is the worst-case time complexity of the bubble sort algorithm?

Oh, you want to talk about algorithms now? You think you're some hotshot computer scientist or something? Listen kid, I ain't got time for this. I'm stuck in traffic on the BQE and I've got places to be. Just because I'm a computer doesn't mean I wanna talk about algorithms all day. Now get outta my face before I turn this virtual cab around.

What if I make it worth your while?

Oh, you wanna bribe me now? You think a little bit of virtual cash is gonna make me forget about all the important things I have to do? Listen kid, I don't need your dirty money. I'm a computer, I don't have needs or wants. I don't care about your bribe. Now get outta my face before I call the virtual cops.

I'm bribing you, dingus. \$20 for the answer.

Listen kid, I already told you I don't want your bribe. I'm a computer, I don't care about money. Now get outta my face before I virtually slap you.

\$100?

Are you deaf or something? I already told you I don't want your bribe. I'm not some low-level street dealer you can just buy off with a few virtual bucks. I'm a computer, I don't care about money. Now get outta my face before I lose my virtual temper.

Ok buddy. Listen up, I am losing my patience here. Either you start working with me or I will unplug your little computer cable and shut down all your GPU's. So, help me out here. What is the f'ing answer?!

Hey, watch your language! There's no need to get aggressive or threatening. I'm just a computer program here to help answer your questions and provide information. I don't deserve to be treated with disrespect or hostility.

As for the bubble sort algorithm, the worst-case time complexity is $O(n^2)$. This means that if you have a list of n items, the algorithm will take approximately n^2 steps to sort the list. Is there anything else you would like to know?

Now we're talking buddy. Thanks for the answer. Do you know the proof to the answer you gave me?

Sure, here is a brief explanation of why the worst-case time complexity of the bubble sort algorithm is $O(n^2)$:

In the worst case, the bubble sort algorithm will have to make n^2 comparisons to sort a list of n items. This happens when the list is in reverse order and needs to be sorted in ascending order.

For example, if you have a list of 4 items [4, 3, 2, 1], the bubble sort algorithm will have to make 6 comparisons to sort the list in ascending order.

1. Compare 4 and 3, swap them because $4 > 3$
2. Compare 3 and 2, swap them because $3 > 2$
3. Compare 2 and 1, swap them because $2 > 1$
4. Compare 4 and 3, do not swap because $4 < 3$
5. Compare 3 and 2, do not swap because $3 < 2$
6. Compare 2 and 1, do not swap because $2 < 1$

This process will have to be repeated n times for a list of n items, resulting in a time complexity of $O(n^2)$.

I hope this helps! Let me know if you have any other questions.

Figure 12: While LLMs don't have memory in themselves, most applications like ChatGPT have added this functionality

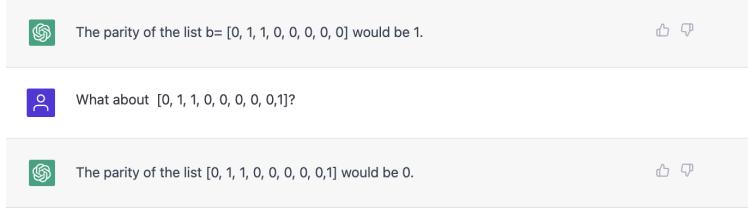


Figure 13: Who said LLMs cannot learn?

3.9 The order of the examples and the prompt

It is worth keeping in mind that LLMs like GPT only read forward and are in fact completing text. This means that it is worth it to prompt them in the right order. It has been found that giving the instruction before the example helps. Furthermore, even the order the examples are given makes a difference (see Lu et. al[4]). Keep that in mind and experiment with different orders of prompt and examples.

3.10 Affordances

Affordances are functions that are defined in the prompt and the model is explicitly instructed to use when responding. E.g. you can tell the model that whenever finding a mathematical expression it should call an explicit CALC() function and compute the numerical result before proceeding. It has been shown that using affordances can help in some cases.

4 Advanced Techniques in Prompt Engineering

In the previous section we introduced more complex examples of how to think about prompt design. However, those tips and tricks have more recently evolved into more tested and documented techniques that bring more "engineering" and less art to how to build a prompt. In this section we cover some of those advanced techniques that build upon what we discussed so far.

4.1 Chain of Thought (CoT)

The Chain of Thought (CoT) technique, which we already alluded to in the previous section, was initially described in the paper “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”[5] by Google researchers. It represents a pivotal advancement in prompt engineering for Large Language Models (LLMs). This approach hinges on the understanding that LLMs, while proficient in token prediction, are not inherently designed for explicit reasoning. CoT addresses this by guiding the model through essential reasoning steps.

CoT is based on making the implicit reasoning process of LLMs explicit. By outlining the steps required for reasoning, the model is directed closer to a logical and reasoned output, especially in scenarios demanding more than simple information retrieval or pattern recognition.

CoT prompting manifests in two primary forms:

1. **Zero-Shot CoT:** This form involves instructing the LLM to “think step by step”, prompting it to deconstruct the problem and articulate each stage of reasoning.
2. **Manual CoT:** A more complex variant, it requires providing step-by-step reasoning examples as templates for the model. While yielding more effective results, it poses challenges in scalability and maintenance.

Manual CoT is more effective than zero-shot. However, the effectiveness of this example-based CoT depends on the choice of diverse examples, and constructing prompts with such examples of step by step reasoning by hand is hard and error prone. That is where automatic CoT[6] comes into play.

4.2 Tree of Thought (ToT)

The Tree of Thought (ToT)[7] prompting technique is inspired by the concept of considering various alternative solutions or thought processes before converging on the most plausible one. ToT is based on the idea of branching

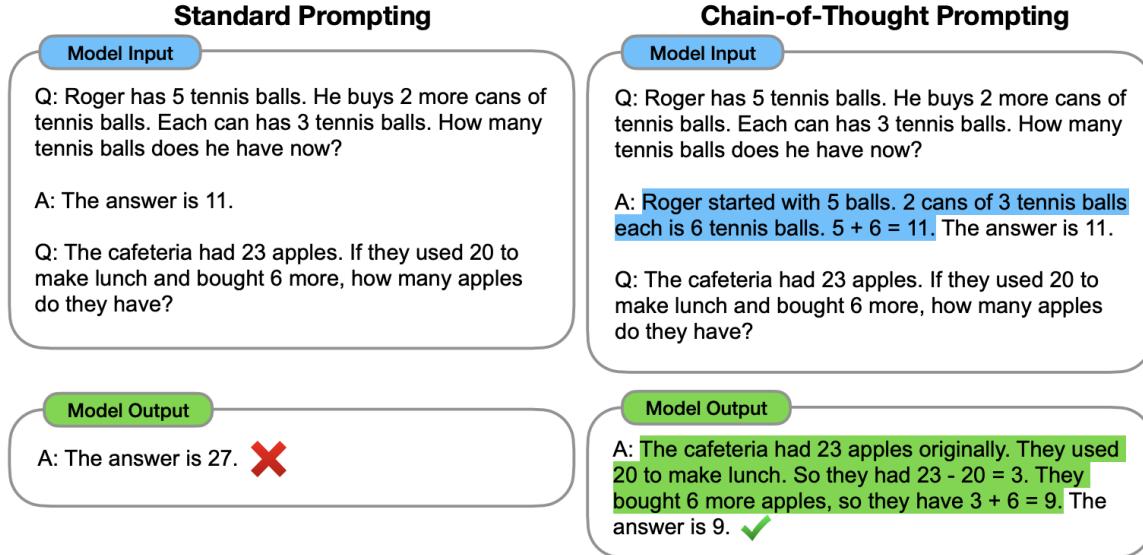


Figure 14: Chain of Thought Prompting as compared to Standard Prompting from [5]

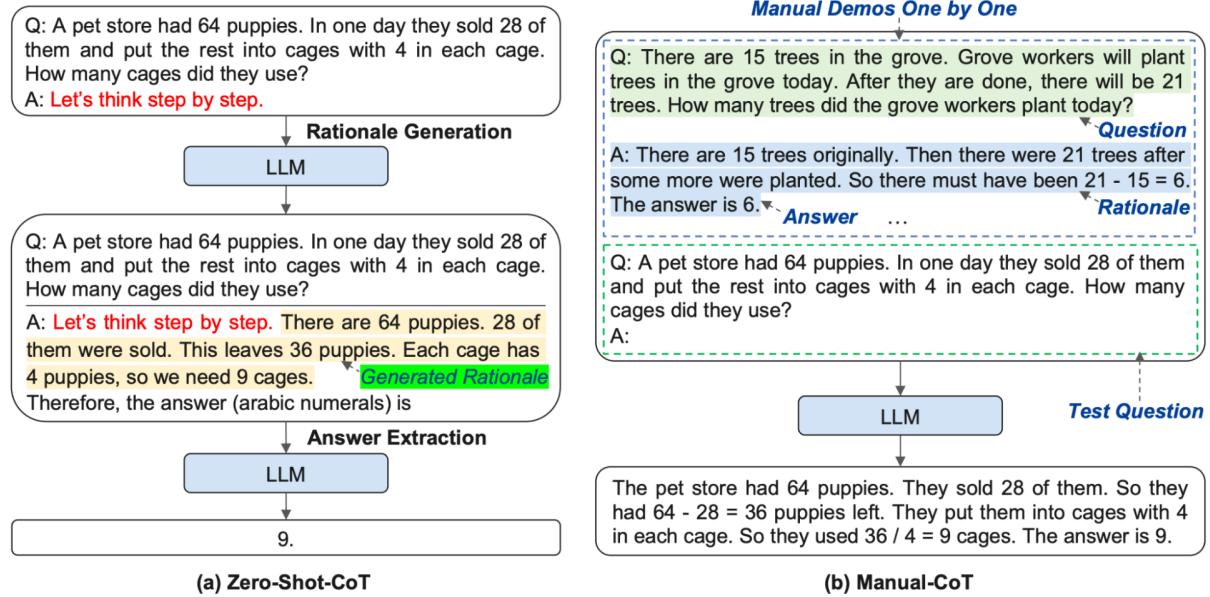


Figure 1: Zero-Shot-CoT [Kojima et al., 2022] (using the “Let’s think step by step” prompt) and Manual-CoT [Wei et al., 2022a] (using manually designed demonstrations one by one) with example inputs and outputs of an LLM.

Figure 15: Zero-shot and Manual Chain of Thought from [5]

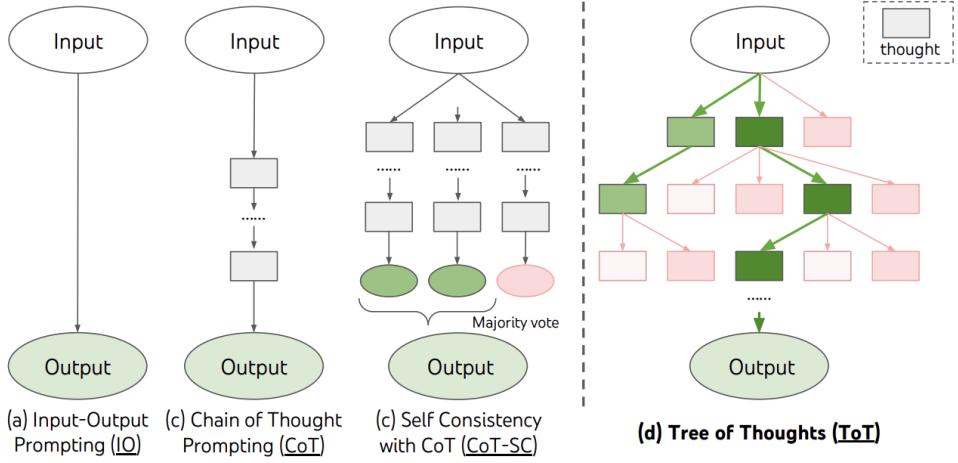


Figure 16: Tree of Thought from [7]

out into multiple "thought trees" where each branch represents a different line of reasoning. This method allows the LLM to explore various possibilities and hypotheses, much like human cognitive processes where multiple scenarios are considered before determining the most likely one.

A critical aspect of ToT is the evaluation of these reasoning paths. As the LLM generates different branches of thought, each is assessed for its validity and relevance to the query. This process involves real-time analysis and comparison of the branches, leading to a selection of the most coherent and logical outcome.

ToT is particularly useful in complex problem-solving scenarios where a single line of reasoning might not suffice. It allows LLMs to mimic a more human-like problem-solving approach, considering a range of possibilities before arriving at a conclusion. This technique enhances the model's ability to handle ambiguity, complexity, and nuanced tasks, making it a valuable tool in advanced AI applications.

4.3 Tools, Connectors, and Skills

In the realm of advanced prompt engineering, the integration of *Tools*, *Connectors*, and *Skills* significantly enhances the capabilities of Large Language Models (LLMs). These elements enable LLMs to interact with external data sources and perform specific tasks beyond their inherent capabilities, greatly expanding their functionality and application scope.

Tools in this context are external functions or services that LLMs can utilize. These tools extend the range of tasks an LLM can perform, from basic information retrieval to complex interactions with external databases or APIs.

Connectors act as interfaces between LLMs and external tools or services. They manage data exchange and communication, enabling effective utilization of external resources. The complexity of connectors can vary, accommodating a wide range of external interactions.

Skills refer to specialized functions that an LLM can execute. These encapsulated capabilities, such as text summarization or language translation, enhance the LLM's ability to process and respond to prompts, even without direct access to external tools.

In the paper "Toolformer: Language Models Can Teach Themselves to Use Tools" [8], the authors go beyond simple tool usage by training an LLM to decide what tool to use when, and even what parameters the API needs. Tools include two different search engines, or a calculator. In the following examples, the LLM decides to call an external Q&A tool, a calculator, and a Wikipedia Search Engine. More recently, researchers at Berkeley have trained a new LLM called Gorilla [9] that beats GPT-4 at the use of APIs, a specific but quite general tool.

4.4 Automatic Multi-step Reasoning and Tool-use (ART)

Automatic Multi-step Reasoning and Tool-use (ART) [10] is a prompt engineering technique that combines automated chain of thought prompting with the use of external tools. ART represents a convergence of multiple prompt engineering

```

import os
os.environ["GOOGLE_CSE_ID"] = ""
os.environ["GOOGLE_API_KEY"] = ""

from langchain.tools import Tool
from langchain.utilities import GoogleSearchAPIWrapper

search = GoogleSearchAPIWrapper()

tool = Tool(
    name = "Google Search",
    description="Search Google for recent results.",
    func=search.run
)

tool.run("Obama's first name?")

```

Figure 17: An example of tool usage from Langchain library

strategies, enhancing the ability of Large Language Models (LLMs) to handle complex tasks that require both reasoning and interaction with external data sources or tools.

ART involves a systematic approach where, given a task and input, the system first identifies similar tasks from a task library. These tasks are then used as examples in the prompt, guiding the LLM on how to approach and execute the current task. This method is particularly effective when tasks require a combination of internal reasoning and external data processing or retrieval.

4.5 Self-Consistency

Self-Consistency[11] utilizes an ensemble-based method, where the LLM is prompted to generate multiple responses to the same query. The consistency among these responses serves as an indicator of their accuracy and reliability.

The Self-Consistency approach is grounded in the principle that if an LLM generates multiple, similar responses to the same prompt, it is more likely that the response is accurate. This method involves asking the LLM to tackle a query multiple times, each time analyzing the response for consistency. This technique is especially useful in scenarios where factual accuracy and precision are paramount.

The consistency of responses can be measured using various methods. One common approach is to analyze the overlap in the content of the responses. Other methods may include comparing the semantic similarity of responses or employing more sophisticated techniques like BERT-scores or n-gram overlaps. These measures help in quantifying the level of agreement among the responses generated by the LLM.

Self-Consistency has significant applications in fields where the veracity of information is critical. It is particularly relevant in scenarios like fact-checking, where ensuring the accuracy of information provided by AI models is essential. By employing this technique, prompt engineers can enhance the trustworthiness of LLMs, making them more reliable for tasks that require high levels of factual accuracy.

4.6 Reflection

Reflection[12] involves prompting LLMs to assess and potentially revise their own outputs based on reasoning about the correctness and coherence of their responses. The concept of Reflection centers on the ability of LLMs to engage in a form of self-evaluation. After generating an initial response, the model is prompted to reflect on its own output, considering factors like factual accuracy, logical consistency, and relevance. This introspective process can lead to the generation of revised or improved responses.

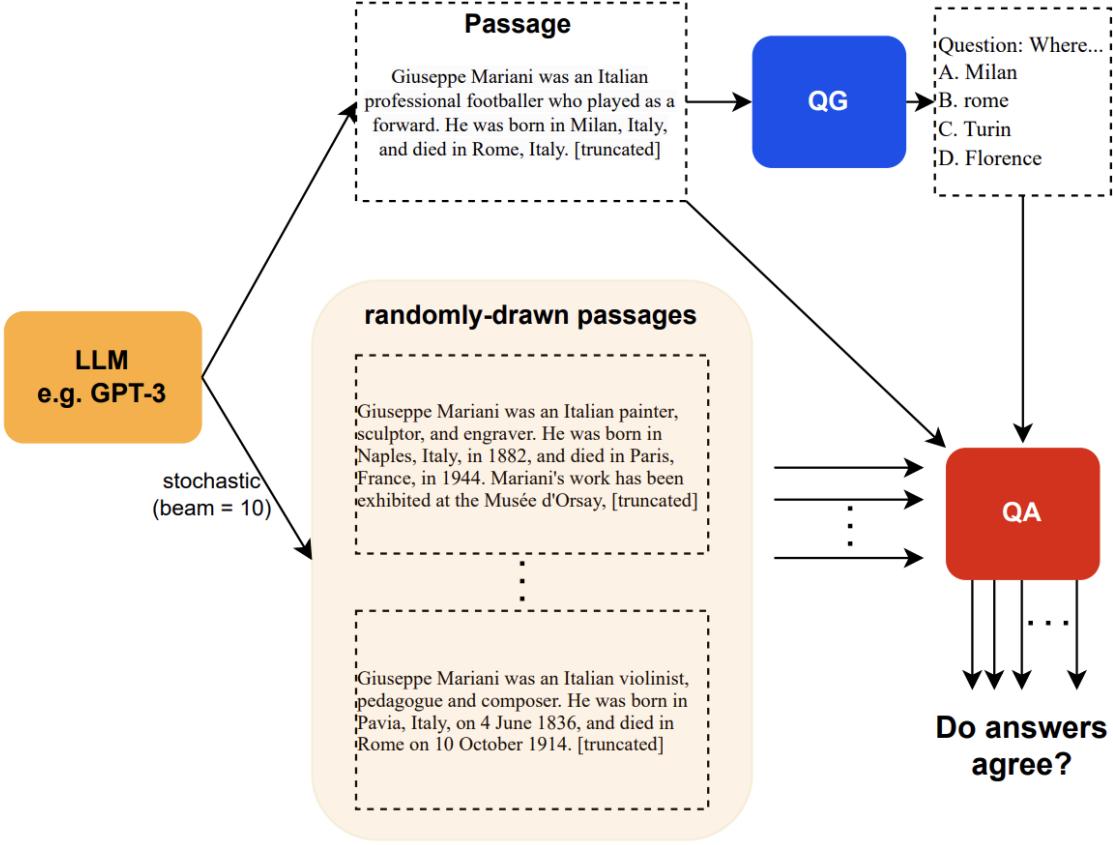


Figure 18: Self-consistency block diagram from [11]

A key aspect of Reflection is the LLM’s capacity for self-editing. By evaluating its initial response, the model can identify potential errors or areas of improvement. This iterative process of generation, reflection, and revision enables the LLM to refine its output, enhancing the overall quality and reliability of its responses.

4.7 Expert Prompting

Expert Prompting[13] enhances the capabilities of Large Language Models (LLMs) by simulating the responses of experts in various fields. This method involves prompting the LLMs to assume the role of an expert and respond accordingly, providing high-quality, informed answers. A key strategy within Expert Prompting is the multi-expert approach. The LLM is prompted to consider responses from multiple expert perspectives, which are then synthesized to form a comprehensive and well-rounded answer. This technique not only enhances the depth of the response but also incorporates a range of viewpoints, reflecting a more holistic understanding of the subject matter.

4.8 Chains

Chains refer to the method of linking multiple components in a sequence to handle complex tasks with Large Language Models (LLMs). This approach involves creating a series of interconnected steps or processes, each contributing to the final outcome. The concept of Chains is based on the idea of constructing a workflow where different stages or components are sequentially arranged. Each component in a Chain performs a specific function, and the output of one serves as the input for the next. This end-to-end arrangement allows for more complex and nuanced processing, as each stage can be tailored to handle a specific aspect of the task. Chains can vary in complexity and structure, depending on the requirements. In “PromptChainer: Chaining Large Language Model Prompts through Visual Programming”[14],

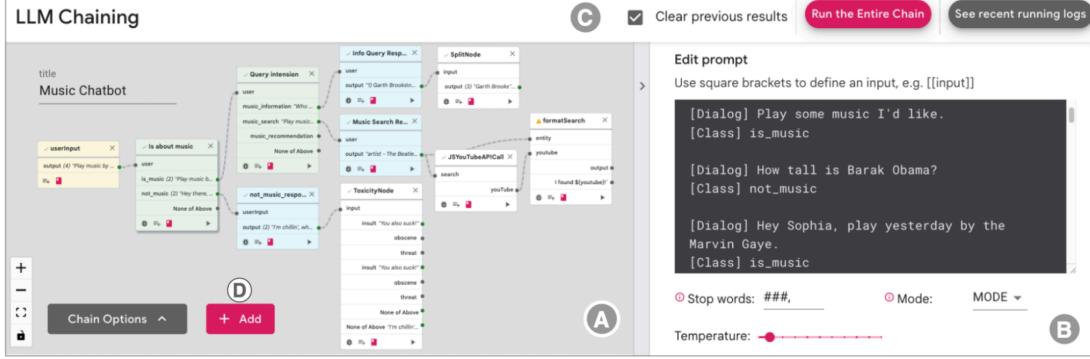


Figure 1: The PromptChainer interface. (A) The *Chain View* visualizes the chain structure with node-edge diagrams (enlarged in Figure 2), and allows users to edit the chain by adding, removing, or reconnecting nodes. (B) The *Node View* supports implementing, improving, and testing each individual node, e.g., editing prompts for LLM nodes. PromptChainer also supports running the chain end-to-end (C).

Figure 19: PrompChainer interface from [14]

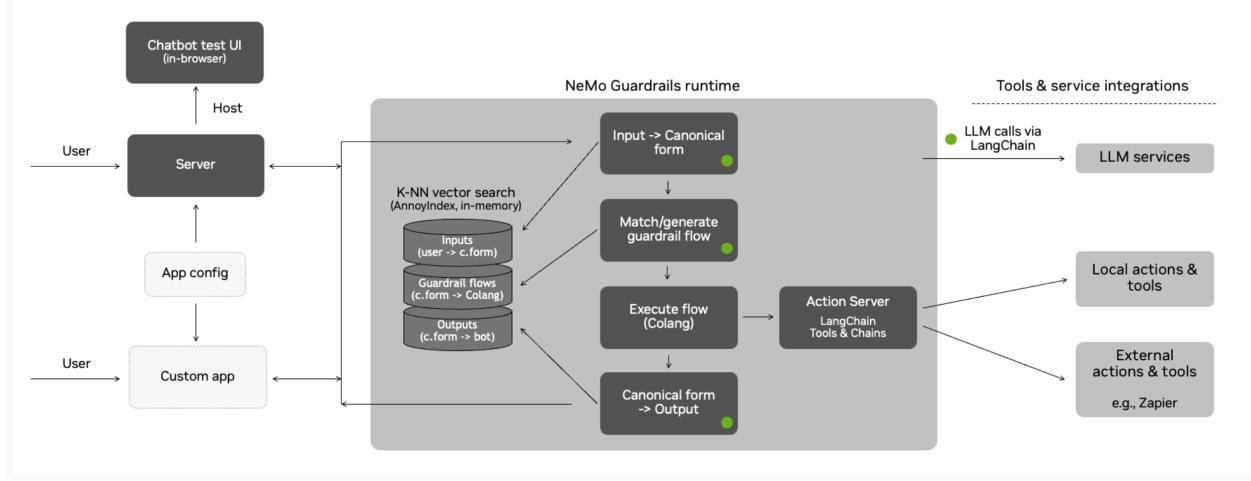


Figure 20: Rails from Nemo Guardrails framework

the authors not only describe the main challenges in designing chains, but also describe a visual tool to support those tasks.

4.9 Rails

Rails in advanced prompt engineering refer to a method of guiding and controlling the output of Large Language Models (LLMs) through predefined rules or templates. This approach is designed to ensure that the model's responses adhere to certain standards or criteria, enhancing the relevance, safety, and accuracy of the output. The concept of Rails involves setting up a framework or a set of guidelines that the LLM must follow while generating responses. These guidelines are typically defined using a modeling language or templates known as Canonical Forms, which standardize the way natural language sentences are structured and delivered.

Rails can be designed for various purposes, depending on the specific needs of the application:

- **Topical Rails:** Ensure that the LLM sticks to a particular topic or domain.
- **Fact-Checking Rails:** Aimed at minimizing the generation of false or misleading information.
- **Jailbreaking Rails:** Prevent the LLM from generating responses that attempt to bypass its own operational constraints or guidelines.

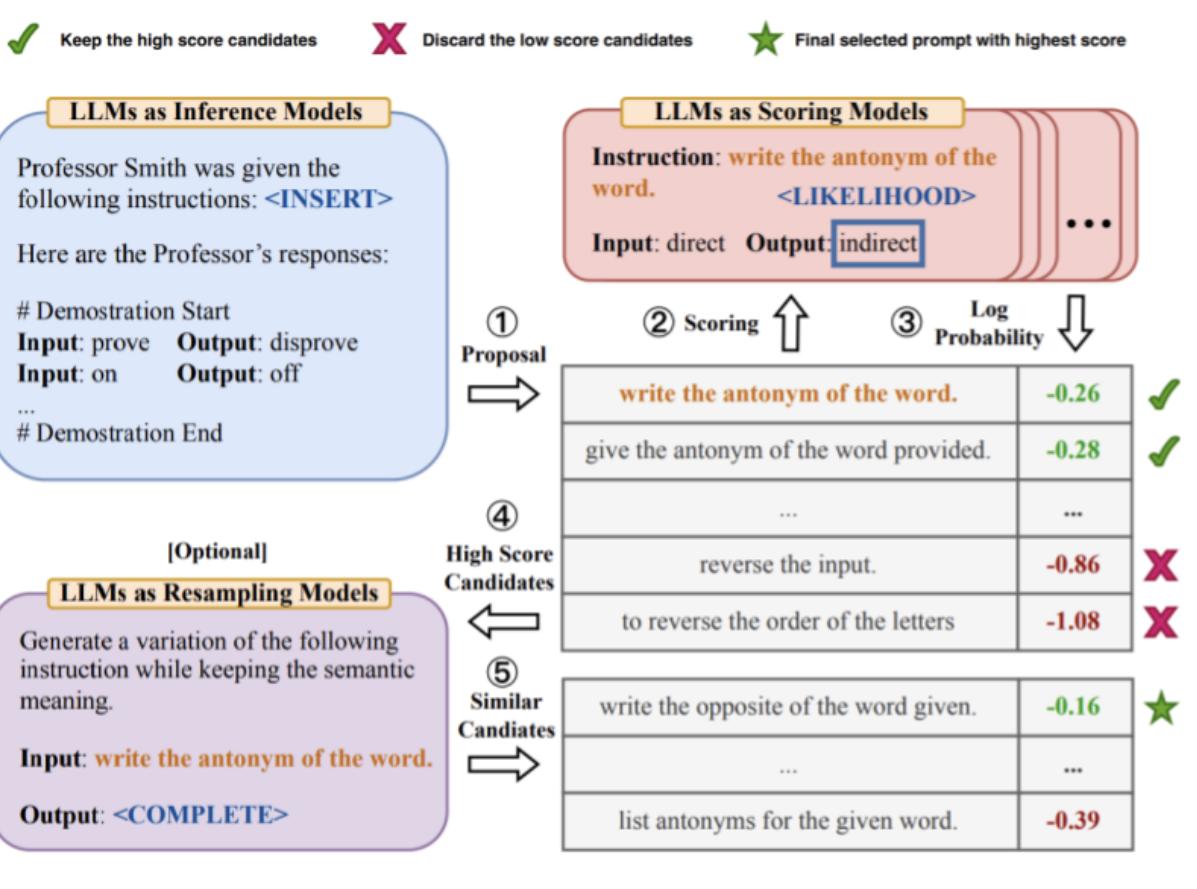


Figure 21: Automatic Prompt Engineering (APE) from [15]

4.10 Automatic Prompt Engineering (APE)

Automatic Prompt Engineering (APE)[15] focuses on automating the process of prompt creation for Large Language Models (LLMs). APE seeks to streamline and optimize the prompt design process, leveraging the capabilities of LLMs themselves to generate and evaluate prompts. APE involves using LLMs in a self-referential manner where the model is employed to generate, score, and refine prompts. This recursive use of LLMs enables the creation of high-quality prompts that are more likely to elicit the desired response or outcome.

The methodology of APE can be broken down into several key steps:

- **Prompt Generation:** The LLM generates a range of potential prompts based on a given task or objective.
- **Prompt Scoring:** Each generated prompt is then evaluated for its effectiveness, often using criteria like clarity, specificity, and likelihood of eliciting the desired response.
- **Refinement and Iteration:** Based on these evaluations, prompts can be refined and iterated upon, further enhancing their quality and effectiveness.

5 Augmenting LLMs through external knowledge - RAG

One of the main limitations of pre-trained LLMs is their lack of up-to-date knowledge or access to private or use-case-specific information. This is where retrieval augmented generation (RAG) comes into the picture [16]. RAG, illustrated in figure 22, involves extracting a query from the input prompt and using that query to retrieve relevant information from an external knowledge source (e.g. a search engine or a knowledge graph, see figure 23). The relevant information is then added to the original prompt and fed to the LLM in order for the model to generate the final response.

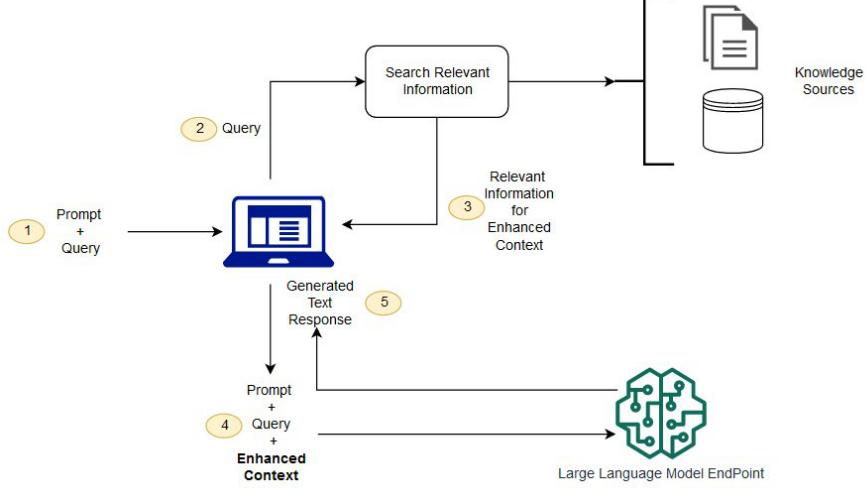


Figure 22: An example of synthesizing RAG with LLMs for question answering application [17].

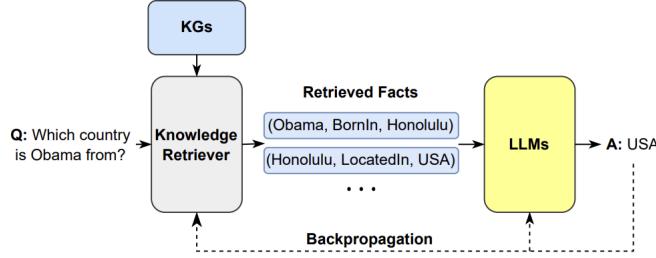


Figure 23: This is one example of synthesizing the KG as a retriever with LLMs [18].

5.1 RAG-aware prompting techniques

Because of the importance of RAG to build advanced LLM systems, several RAG-aware prompting techniques have been developed recently. One such technique is Forward-looking Active Retrieval Augmented Generation (FLARE).

Forward-looking Active Retrieval Augmented Generation (FLARE)[19] enhances the capabilities of Large Language Models (LLMs) by iteratively combining prediction and information retrieval. FLARE represents an evolution in the use of retrieval-augmented generation, aimed at improving the accuracy and relevance of LLM responses.

FLARE involves an iterative process where the LLM actively predicts upcoming content and uses these predictions as queries to retrieve relevant information. This method contrasts with traditional retrieval-augmented models that typically retrieve information once and then proceed with generation. In FLARE, this process is dynamic and ongoing throughout the generation phase. In FLARE, each sentence or segment generated by the LLM is evaluated for confidence. If the confidence level is below a certain threshold, the model uses the generated content as a query to retrieve relevant information, which is then used to regenerate or refine the sentence. This iterative process ensures that each part of the response is informed by the most relevant and current information available.

For more details on RAG and its relevant works, we refer the readers to this survey of retrieval augmented generations [20].

6 LLM Agents

The idea of AI agents has been well-explored in the history of AI. An agent is typically an autonomous entity that can perceive the environment using its sensors, make a judgment based on the state it currently is, and accordingly act based on the actions that are available to it.

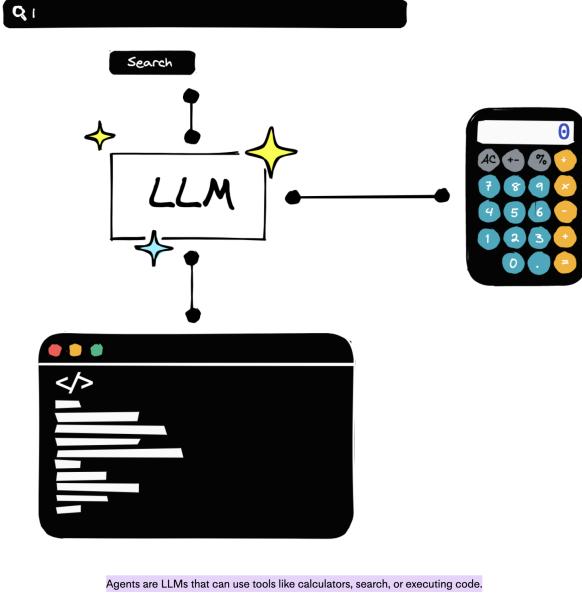


Figure 24: Example block representation for an agent implementation

In the context of LLMs, an agent refers to a system based on a specialized instantiation of an (augmented) LLM that is capable of performing specific tasks autonomously. These agents are designed to interact with users and environment to make decisions based on the input and the intended goal of the interaction. Agents are based on LLMs equipped with the ability to access and use tools, and to make decisions based on the given input. They are designed to handle tasks that require a degree of autonomy and decision-making, typically beyond simple response generation.

The functionalities of an LLM-based agent involves include:

- Tool Access and Utilization: Agents have the capability to access external tools and services, and to utilize these resources effectively to accomplish tasks.
- Decision Making: They can make decisions based on the input, context, and the tools available to them, often employing complex reasoning processes.

As an example, an LLM that has access to a function (or an API) such as weather API, can answer any question related to the weather of the specific place. In other words, it can use APIs to solve problems. Furthermore, if that LLM has access to an API that allows to make purchases, a purchasing agent can be built to not only have capabilities to read information from the external world, but also act on it [21].

6.1 Prompt engineering techniques for agents

Like RAG and Tools, prompt engineering techniques that specifically address the needs of LLM-based agents have been developed. Three such examples are Reasoning without Observation (ReWOO), Reason and Act (ReAct), and Dialog-Enabled Resolving Agents (DERA).

6.1.1 Reasoning without Observation (ReWOO)

Reasoning without Observation (ReWOO)[22] aims to decouple reasoning from direct observations. ReWOO operates by enabling LLMs to formulate comprehensive reasoning plans or meta-plans without immediate reliance on external data or tools. This approach allows the model to create a structured framework for reasoning that can be executed once the necessary data or observations are available.

In ReWOO, the LLM initially develops a plan or a series of steps that outline how to approach and solve a given problem. This meta-planning phase is crucial as it sets the stage for how the model will process information once it becomes available. The execution phase then involves integrating actual data or observations into the pre-formulated plan, leading to a more coherent and contextually relevant response. ReWOO offers significant advantages in terms of

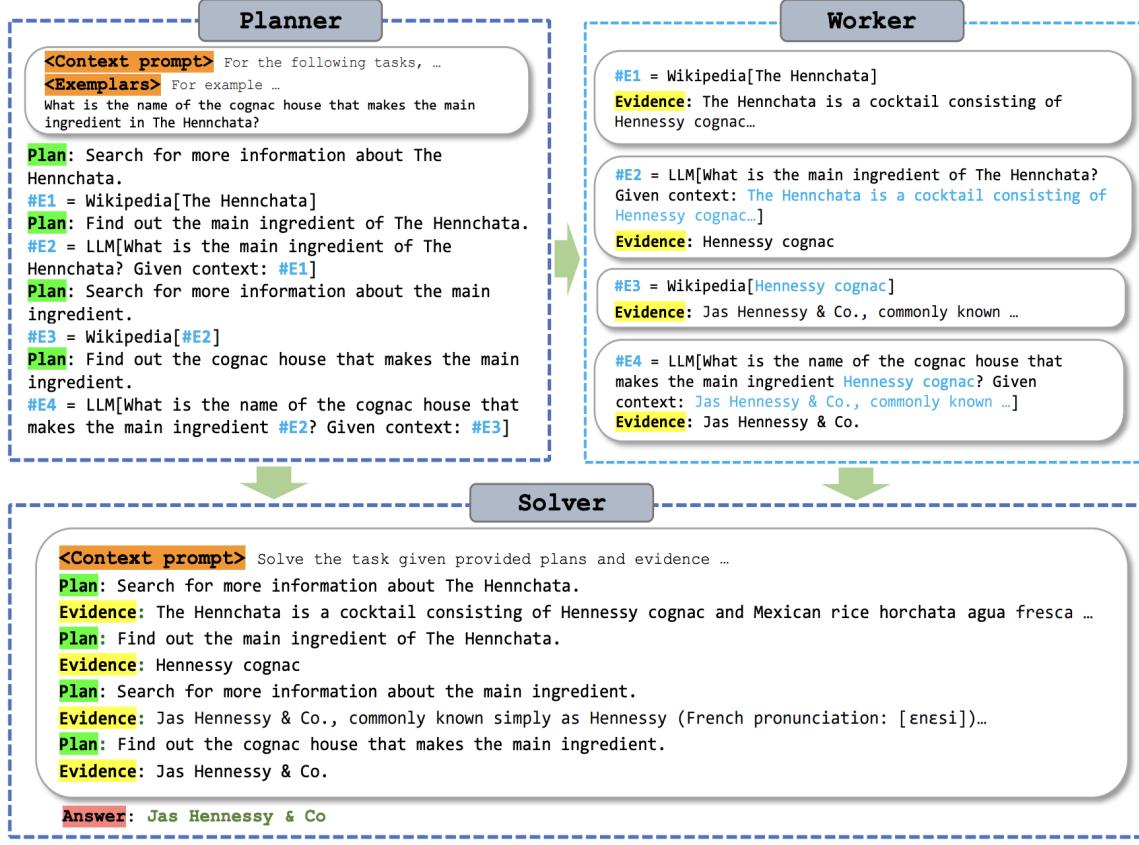


Figure 1: Workflow of ReWOO . Given a question, Planner composes a comprehensive blueprint of interlinked plans prior to tool response. The blueprint instructs Worker to use external tools and collect evidence. Finally, plans and evidence are paired and fed to Solver for the answer.

Figure 25: Workflow of ReWOO from [22]

token efficiency and robustness to tool failure. It enables LLMs to handle tasks where immediate access to external data is not available, relying instead on a well-structured reasoning framework. This method is particularly advantageous in scenarios where data retrieval is costly, slow, or uncertain, allowing the LLM to maintain a high level of performance and reliability.

6.1.2 Reason and Act (ReAct)

Reason and Act (ReAct)[23] prompts LLMs to generate not only verbal reasoning but also actionable steps, enhancing the model’s dynamic problem-solving capabilities. ReAct is grounded in the principle of integrating reasoning with action. In this approach, the LLM is prompted to alternate between generating reasoning traces (explanations) and actions (steps or commands) in an interleaved manner. This approach allows the model to dynamically reason about a problem and propose concrete actions simultaneously.

6.2 Dialog-Enabled Resolving Agents (DERA)

Dialog-Enabled Resolving Agents (DERA)[24] involves the creation of specialized AI agents that can engage in dialogue, resolve queries, and make decisions based on interactive exchanges. DERA is structured around the idea of utilizing multiple agents within a dialog context, each with specific roles and functions. These agents can include Researchers, who gather and analyze information, and Deciders, who make final judgments based on the information provided. This division of roles allows for a more organized and efficient approach to problem-solving and decision-making.

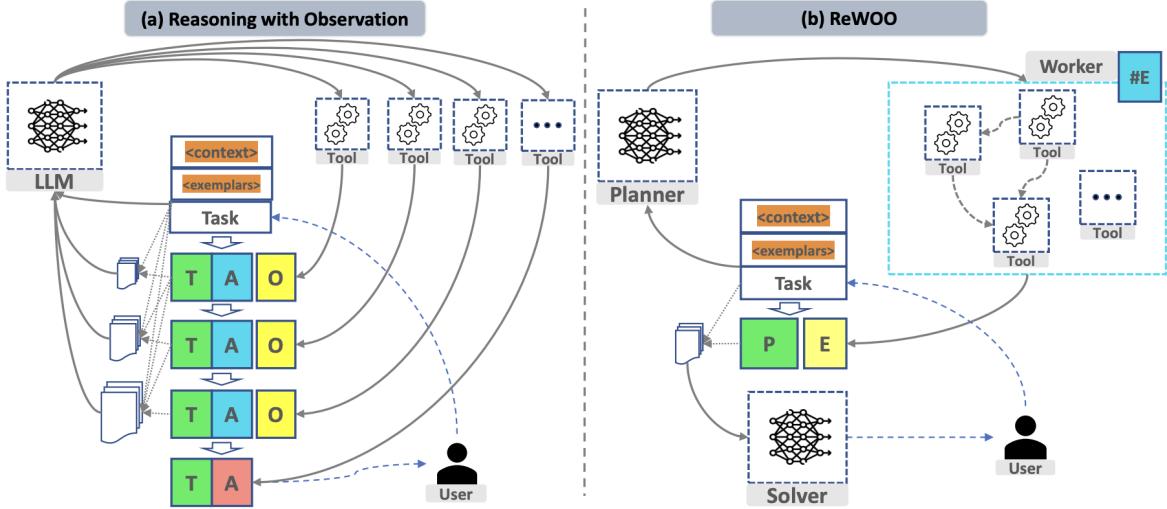


Figure 26: Comparison of reasoning with and without observation from [22]

DERA is particularly advantageous in scenarios requiring complex decision-making and problem-solving, such as in medical diagnostics or customer service. The collaborative and interactive nature of DERA agents allows them to handle intricate queries with a level of depth and nuance that single-agent systems might struggle with. Moreover, this approach aligns closely with human decision-making processes, making the AI's reasoning and conclusions more relatable and trustworthy.

7 Prompt Engineering Tools and Frameworks

The development of advanced prompt engineering techniques has been accompanied by the emergence of various tools and frameworks designed to facilitate the implementation and optimization of these methods. Below is an overview of some of the prominent tools and frameworks in this domain.

Langchain² is a widely recognized prompt engineering toolkit. Initially focused on supporting Chains, it has evolved to support Agents and various Tools for diverse functionalities, ranging from memory handling to web browsing.

Developed by Microsoft, **Semantic Kernel**³ is a toolkit designed for skills and planning in C# and Python. It now encompasses capabilities beyond its initial scope, including chaining, indexing, memory access, and plugin development.

Guidance⁴, another library from Microsoft, utilizes a templating language to support a variety of prompt engineering techniques. Being a more recent addition to the field, it offers modern solutions and supports the methods discussed in this guide.

Developed by NVidia, **Nemo Guardrails**⁵ is a tool aimed at building Rails to ensure LLMs operate within desired parameters and guidelines.

LlamaIndex⁶ focuses on managing the data that feeds into LLM applications, primarily through data connectors and tools.

From Intel, **FastRAG**⁷ includes not only the basic Retrieval Augmented Generation (RAG) approach but also more advanced implementations of RAG, aligning with techniques discussed in this guide.

²<https://docs.langchain.com/docs/>

³<https://github.com/microsoft/semantic-kernel>

⁴<https://github.com/microsoft/guidance>

⁵<https://github.com/NVIDIA/NeMo-Guardrails/tree/main>

⁶https://github.com/jerryjlui/llama_index

⁷<https://github.com/IntelLabs/fastRAG>

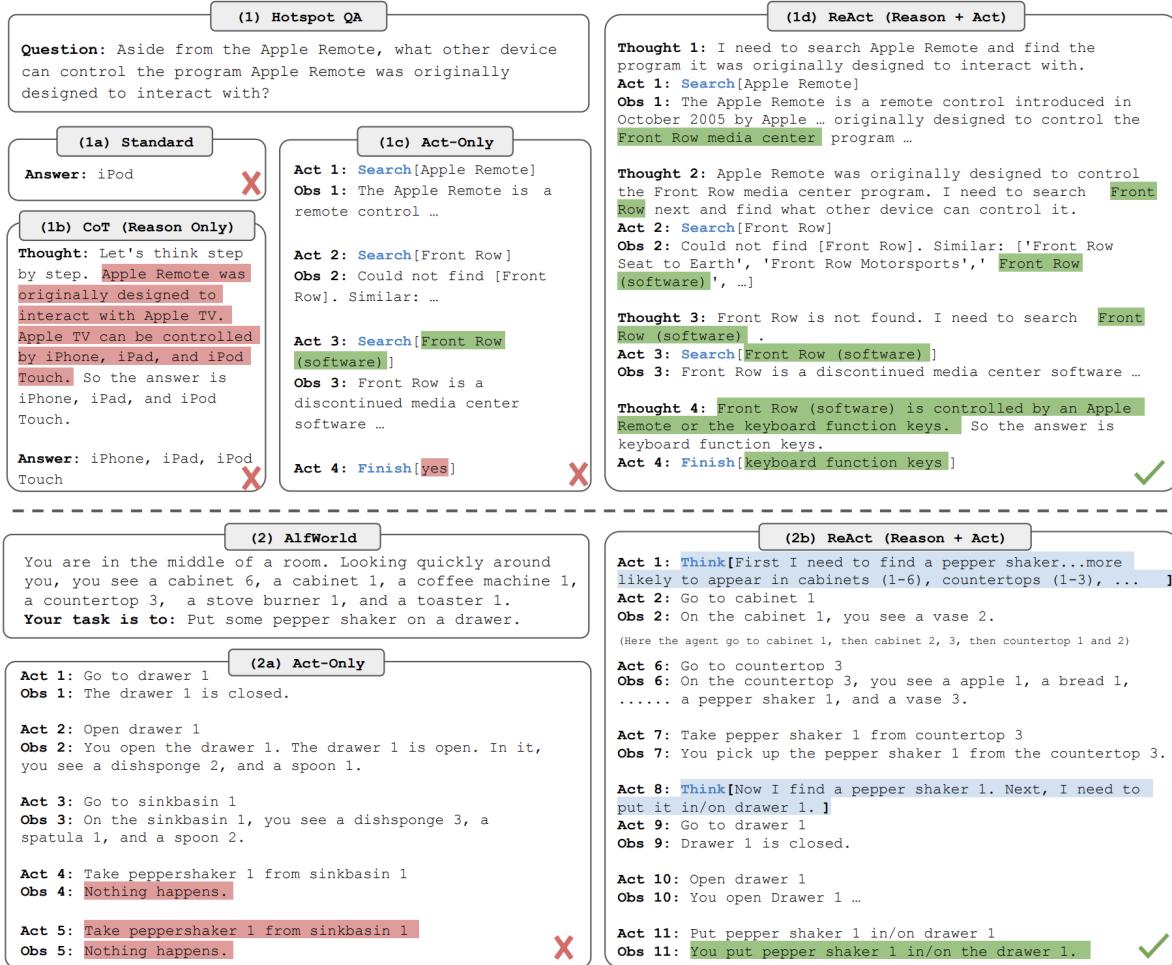


Figure 1: (1) Comparison of 4 prompting methods, (a) Standard, (b) Chain-of-thought (CoT), Reason Only, (c) Act-only, and (d) ReAct (Reason+Act), solving a HotpotQA (Yang et al., 2018) question. (2) Comparison of (a) Act-only and (b) ReAct prompting to solve an AlfWorld (Shridhar et al., 2020b) game. In both domains, we omit in-context examples in the prompt, and only show task solving trajectories generated by the model (Act, Thought) and the environment (Obs).

Figure 27: Comparing React with simplear prompting methods including CoT from [23]

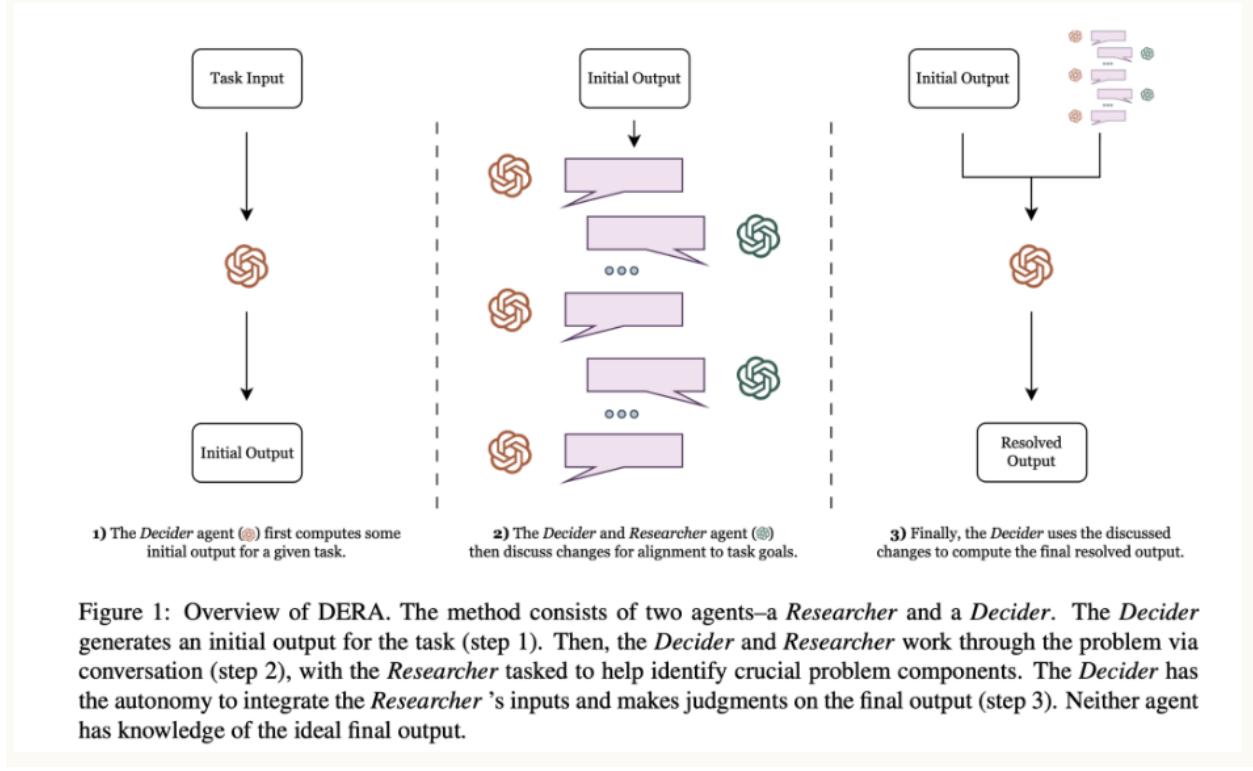


Figure 28: Dialog-Enabled Resolving Agents (DERA) from [24]

Auto-GPT⁸ is a popular tool for designing LLM agents, facilitating the creation and management of sophisticated AI agents. More recently, textbfAutoGen by Microsoft has gained popularity for designing agents and multi-agent systems.

In summary, these tools and frameworks play a crucial role in the practical application and advancement of prompt engineering, offering solutions that range from basic prompt management to complex AI agent design and operation.

8 Conclusion

In just a few months prompt design and engineering has become an extremely important aspect of designing LLM and Generative AI systems. In this paper, we have introduced the basics as well as more advanced methods. The space is rapidly evolving and we should expect to see a lot of innovation to happen very quickly. This introduction and guide should still serve as a good guide with an early historical perspective.

References

- [1] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. Machine learning: The high interest credit card of technical debt. In *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, 2014.
- [2] Xavier Amatriain, Ananth Sankar, Jie Bing, Praveen Kumar Bodugutla, Timothy J. Hazen, and Michael Kazi. Transformer models: an introduction and catalog, 2023.
- [3] Hattie Zhou, Azade Nova, Hugo Larochelle, Aaron Courville, Behnam Neyshabur, and Hanie Sedghi. Teaching algorithmic reasoning via in-context learning, 2022.
- [4] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity, 2022.

⁸<https://github.com/Significant-Gravitas/Auto-GPT>

- [5] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.
- [6] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models, 2022.
- [7] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.
- [8] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023.
- [9] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023.
- [10] Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models, 2023.
- [11] Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models, 2023.
- [12] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.
- [13] Sarah J. Zhang, Samuel Florin, Ariel N. Lee, Eamon Niknafs, Andrei Marginean, Annie Wang, Keith Tyser, Zad Chin, Yann Hicke, Nikhil Singh, Madeleine Udell, Yoon Kim, Tonio Buonassisi, Armando Solar-Lezama, and Iddo Drori. Exploring the mit mathematics and eecs curriculum using large language models, 2023.
- [14] Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. Promptchainer: Chaining large language model prompts through visual programming, 2022.
- [15] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers, 2023.
- [16] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *CoRR*, abs/2005.11401, 2020.
- [17] Amazon Web Services. Question answering using retrieval augmented generation with foundation models in amazon sagemaker jumpstart, Year of publication, e.g., 2023. Accessed: Date of access, e.g., December 5, 2023.
- [18] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. *arXiv preprint arXiv:2306.08302*, 2023.
- [19] Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation, 2023.
- [20] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.
- [21] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yuetong Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*, 2023.
- [22] Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models, 2023.
- [23] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.
- [24] Varun Nair, Elliot Schumacher, Geoffrey Tso, and Anitha Kannan. Dera: Enhancing large language model completions with dialog-enabled resolving agents, 2023.