**What is Prompt Engineering?**

At its core, prompt engineering involves designing, analyzing, and refining the inputs (or "prompts") used to elicit responses from AI models, ensuring that the outputs are as desired. Just as a skilled interviewer can frame questions in various ways to get the most accurate answers from a human interviewee, prompt engineers frame their inputs to AI systems in a way that maximizes the accuracy, relevance, and clarity of the system's outputs.

For many people, the phrase "using prompt engineering" is synonymous with "using ChatGPT". However, this isn't necessarily the case, as prompt engineering can be applied to a broad spectrum of models.

**Basic LLM Parameters/Settings to yield better results**

When working with prompts, you interact with the LLM via an API or directly. You can configure a few parameters to get different results for your prompts.

**Temperature** - In short, the lower the temperature, the more deterministic the results in the sense that the highest probable next token is always picked. Increasing temperature could lead to more randomness, which encourages more diverse or creative outputs. You are essentially increasing the weights of the other possible tokens. In terms of application, you might want to use a lower temperature value for tasks like fact-based QA to encourage more factual and concise responses. For poem generation or other creative tasks, it might be beneficial to increase the temperature value.

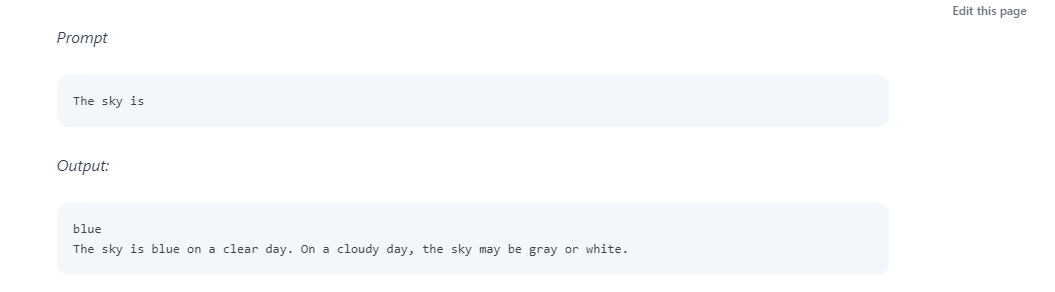
Note: The sampling temperature, between 0 and 1. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic. If set to 0, the model will use [**log probability**](https://en.wikipedia.org/wiki/Log_probability) to automatically increase the temperature until certain thresholds are hit.

**Top\_p** - Similarly, with top\_p, a sampling technique with temperature called nucleus sampling, you can control how deterministic the model is at generating a response. If you are looking for exact and factual answers keep this low. If you are looking for more diverse responses, increase to a higher value.

**The general recommendation is to alter one, not both.**Before starting with some basic examples, keep in mind that your results may vary depending on the version of LLM you use.

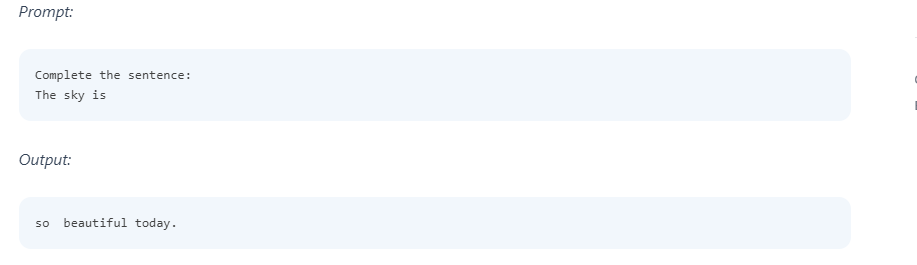
**Basics of Prompting**

You can achieve a lot with simple prompts, but the quality of results depends on how much information you provide it and how well-crafted it is. A prompt can contain information like the *instruction* or *question* you are passing to the model and include other details such as *context*, *inputs*, or *examples*. You can use these elements to instruct the model better and as a result get better results.  
  
Let's get started by going over a basic example of a simple prompt:

  
  
  
As you can see, the language model outputs a continuation of strings that make sense given the context "The sky is". The output might be unexpected or far from the task you want to accomplish.

This basic example also highlights the necessity to provide more context or instructions on what specifically you want to achieve.

Let's try to improve it a bit:

  
  
Is that better? Well, you told the model to complete the sentence so the result looks a lot better as it follows exactly what you told it to do ("complete the sentence"). This approach of designing optimal prompts to instruct the model to perform a task is what's referred to as **prompt engineering**.

The example above is a basic illustration of what's possible with LLMs today. Today's LLMs are able to perform all kinds of advanced tasks that range from text summarization to mathematical reasoning to code generation.  
  
When prompting like the above, it's also referred to as *zero-shot prompting*, i.e., you are directly prompting the model for a response without any examples or demonstrations about the task you want it to achieve. Some large language models do have the ability to perform zero-shot prompting but it depends on the complexity and knowledge of the task at hand.

**Elements of a Prompt**As we cover more and more examples and applications with prompt engineering, you will notice that certain elements make up a prompt.A prompt contains any of the following elements:

**Instruction** - a specific task or instruction you want the model to perform

**Context** - external information or additional context that can steer the model to better responses

**Input Data** - the input or question that we are interested to find a response for

**Output Indicator** - the type or format of the output.

You do not need all the four elements for a prompt and the format depends on the task at hand. We will touch on more concrete examples in upcoming guides.

**General Tips or Best Practices for Designing Prompts**

Here are some tips to keep in mind while you are designing your prompts:

**Start Simple**

As you get started with designing prompts, you should keep in mind that it is really an iterative process that requires a lot of experimentation to get optimal results. Using a simple playground from OpenAI or Cohere is a good starting point.

You can start with simple prompts and keep adding more elements and context as you aim for better results. Iterating your prompt along the way is vital for this reason. As you read the guide, you will see many examples where specificity, simplicity, and conciseness will often give you better results.

When you have a big task that involves many different subtasks, you can try to break down the task into simpler subtasks and keep building up as you get better results. This avoids adding too much complexity to the prompt design process at the beginning.

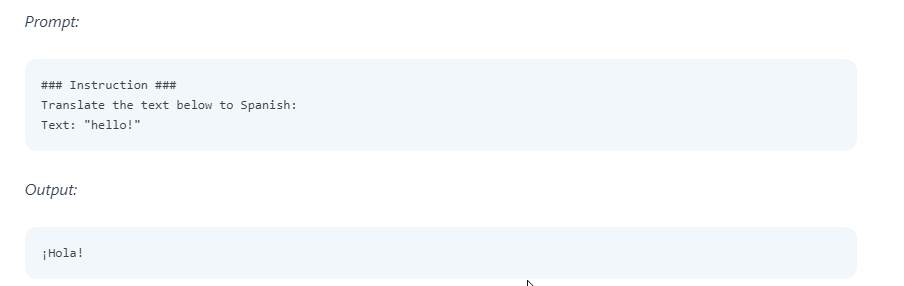
**The Instruction**

You can design effective prompts for various simple tasks by using commands to instruct the model what you want to achieve, such as "Write", "Classify", "Summarize", "Translate", "Order", etc.

Keep in mind that you also need to experiment a lot to see what works best. Try different instructions with different keywords, contexts, and data and see what works best for your particular use case and task. Usually, the more specific and relevant the context is to the task you are trying to perform, the better. We will touch on the importance of sampling and adding more context in the upcoming guides.

Others recommend that you place instructions at the beginning of the prompt. Another recommendation is to use some clear separator like "###" to separate the instruction and context.

For instance:

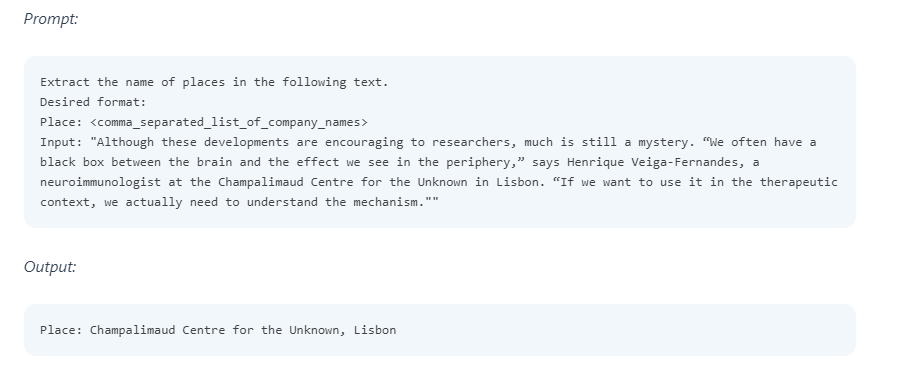


**Specificity**

Be very specific about the instruction and task you want the model to perform. The more descriptive and detailed the prompt is, the better the results. This is particularly important when you have a desired outcome or style of generation you are seeking. There aren't specific tokens or keywords that lead to better results. It's more important to have a good format and descriptive prompt. In fact, providing examples in the prompt is very effective to get desired output in specific formats.

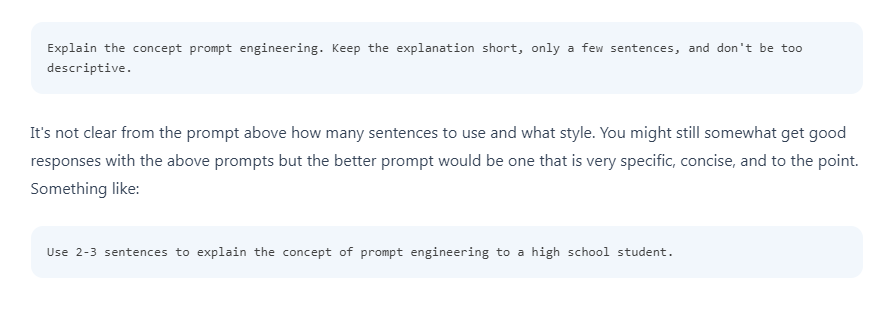
When designing prompts, you should also keep in mind the length of the prompt as there are limitations regarding how long the prompt can be. Thinking about how specific and detailed you should be. Including too many unnecessary details is not necessarily a good approach. The details should be relevant and contribute to the task at hand. This is something you will need to experiment with a lot. We encourage a lot of experimentation and iteration to optimize prompts for your applications.

As an example, let's try a simple prompt to extract specific information from a piece of text.

  
**Avoid Impreciseness**

Given the tips above about being detailed and improving format, it's easy to fall into the trap of wanting to be too clever about prompts and potentially creating imprecise descriptions. It's often better to be specific and direct. The analogy here is very similar to effective communication -- the more direct, the more effective the message gets across.

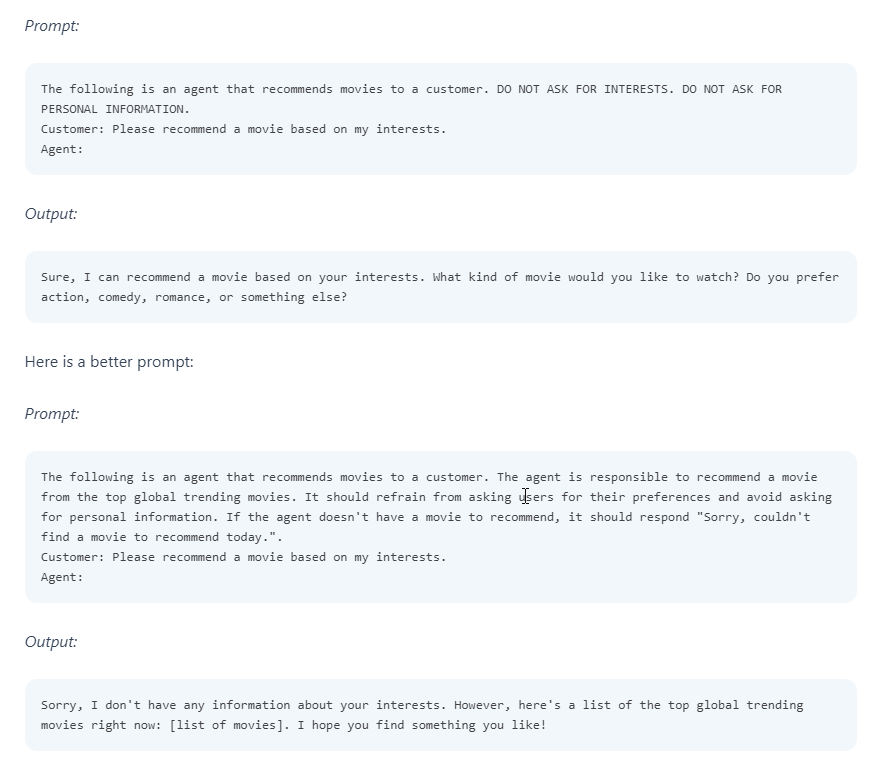
For example, you might be interested in learning the concept of prompt engineering. You might try something like:



**To do or not to do?**

Another common tip when designing prompts is to avoid saying what not to do but say what to do instead. This encourages more specificity and focuses on the details that lead to good responses from the model.

Here is an example of a movie recommendation chatbot failing at exactly what I don't want it to do because of how I wrote the instruction -- focusing on what not to do.



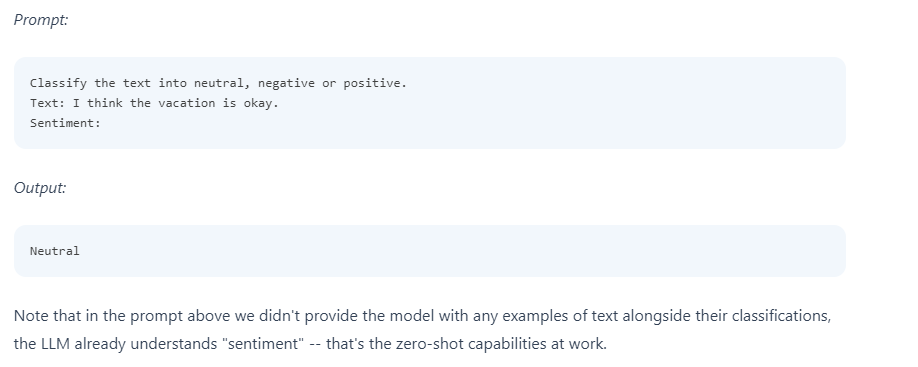
**Different Techniques of Prompting:**

1. **Zero-shot Prompting**
2. **Few-shot Prompting**
3. **Chain-of-Thought Prompting**
4. **Tree of Thoughts**
5. **Retrieval Augmented Generation**

**Zero-shot Prompting**

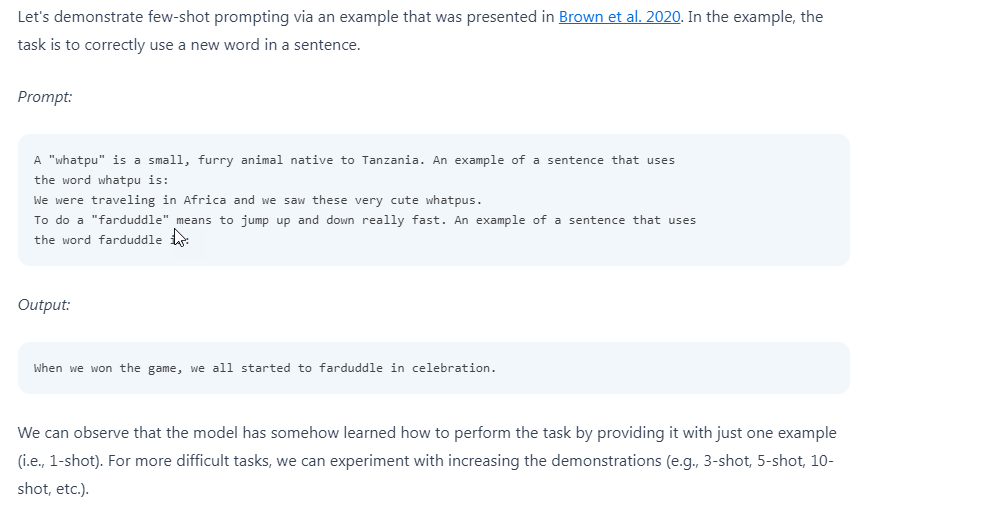
Large LLMs today, such as GPT-3, are tuned to follow instructions and are trained on large amounts of data; so they are capable of performing some tasks "zero-shot."

We tried a few zero-shot examples in the previous section. Here is one of the examples we used:

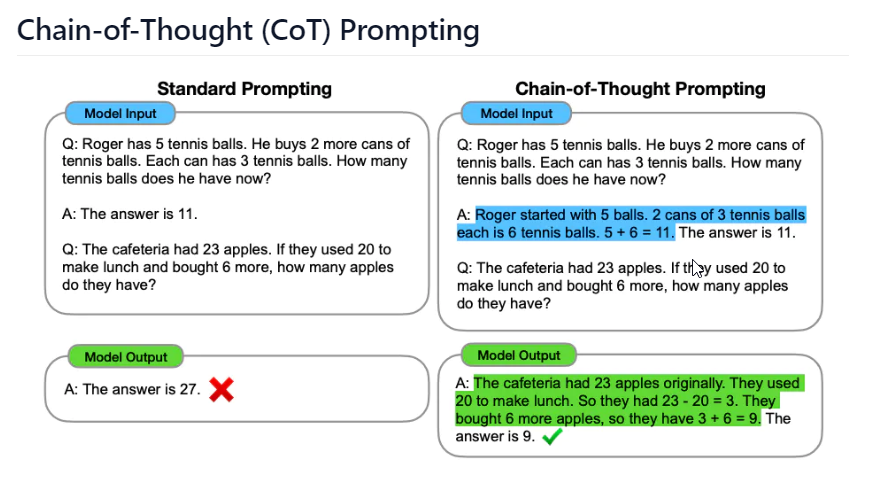


**Few-Shot Prompting**

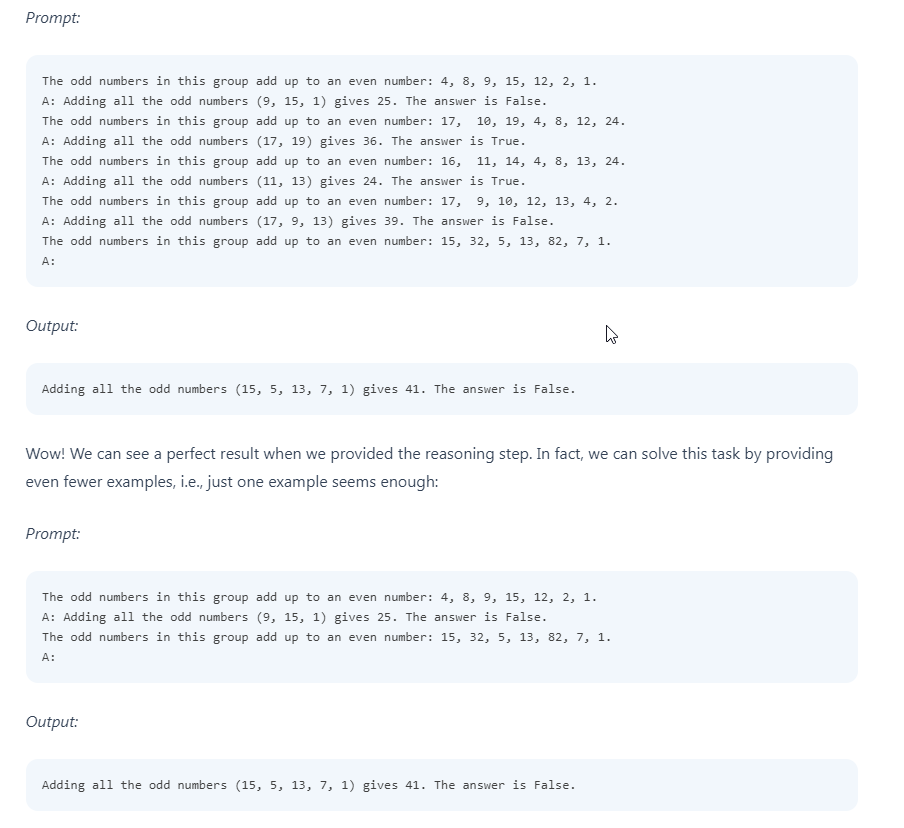
While large-language models demonstrate remarkable zero-shot capabilities, they still fall short on more complex tasks when using the zero-shot setting. Few-shot prompting can be used as a technique to enable in-context learning where we provide demonstrations in the prompt to steer the model to better performance. The demonstrations serve as conditioning for subsequent examples where we would like the model to generate a response.



**Chain-of-Thought Prompting**



Introduced in [Wei et al. (2022)(opens in a new tab)](https://arxiv.org/abs/2201.11903), chain-of-thought (CoT) prompting enables complex reasoning capabilities through intermediate reasoning steps. You can combine it with few-shot prompting to get better results on more complex tasks that require reasoning before responding.



**Retrieval Augmented Generation (RAG)**

General-purpose language models can be fine-tuned to achieve several common tasks such as sentiment analysis and named entity recognition. These tasks generally don't require additional background knowledge.

For more complex and knowledge-intensive tasks, it's possible to build a language model-based system that accesses external knowledge sources to complete tasks. This enables more factual consistency, improves reliability of the generated responses, and helps to mitigate the problem of "hallucination".

Meta AI researchers introduced a method called [Retrieval Augmented Generation (RAG)(opens in a new tab)](https://ai.facebook.com/blog/retrieval-augmented-generation-streamlining-the-creation-of-intelligent-natural-language-processing-models/) to address such knowledge-intensive tasks. RAG combines an information retrieval component with a text generator model. RAG can be fine-tuned and its internal knowledge can be modified in an efficient manner and without needing retraining of the entire model.

RAG takes an input and retrieves a set of relevant/supporting documents given a source (e.g., Wikipedia). The documents are concatenated as context with the original input prompt and fed to the text generator which produces the final output. This makes RAG adaptive for situations where facts could evolve over time. This is very useful as LLMs's parametric knowledge is static. RAG allows language models to bypass retraining, enabling access to the latest information for generating reliable outputs via retrieval-based generation.

Lewis et al., (2021) proposed a general-purpose fine-tuning recipe for RAG. A pre-trained seq2seq model is used as the parametric memory and a dense vector index of Wikipedia is used as non-parametric memory (accessed using a neural pre-trained retriever). Below is a overview of how the approach works:

RAG performs strong on several benchmarks such as [Natural Questions(opens in a new tab)](https://ai.google.com/research/NaturalQuestions), [WebQuestions(opens in a new tab)](https://paperswithcode.com/dataset/webquestions), and CuratedTrec. RAG generates responses that are more factual, specific, and diverse when tested on MS-MARCO and Jeopardy questions. RAG also improves results on FEVER fact verification.

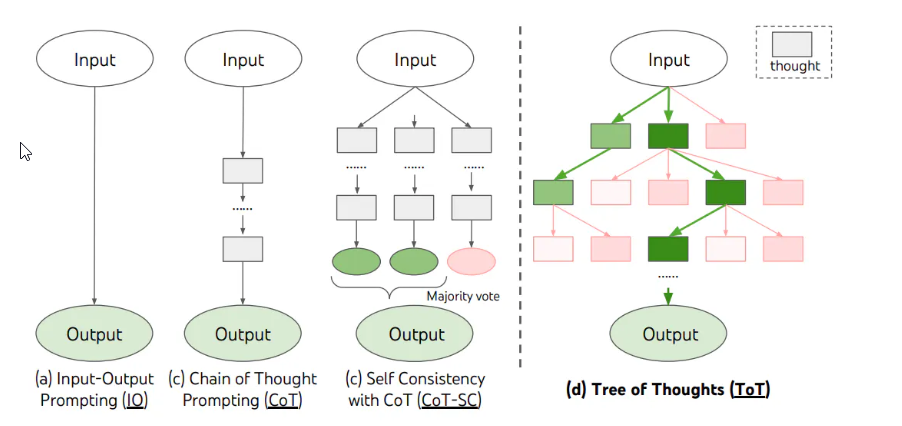
This shows the potential of RAG as a viable option for enhancing outputs of language models in knowledge-intensive tasks.

**Tree of Thoughts (ToT)**

For complex tasks that require exploration or strategic lookahead, traditional or simple prompting techniques fall short. [Yao et el. (2023)(opens in a new tab)](https://arxiv.org/abs/2305.10601) and [Long (2023)(opens in a new tab)](https://arxiv.org/abs/2305.08291) recently proposed Tree of Thoughts (ToT), a framework that generalizes over chain-of-thought prompting and encourages exploration over thoughts that serve as intermediate steps for general problem solving with language models.

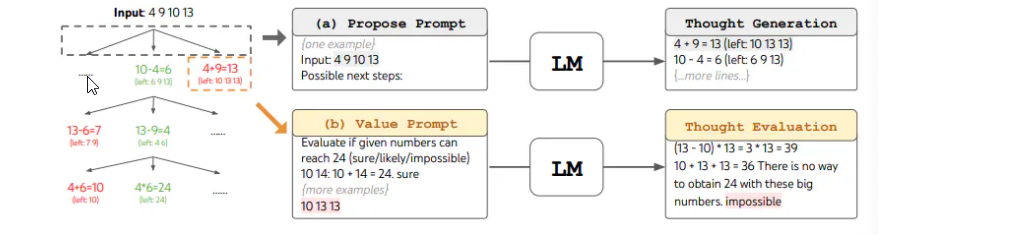
ToT maintains a tree of thoughts, where thoughts represent coherent language sequences that serve as intermediate steps toward solving a problem. This approach enables an LM to self-evaluate the progress intermediate thoughts make towards solving a problem through a deliberate reasoning process. The LM's ability to generate and evaluate thoughts is then combined with search algorithms (e.g., breadth-first search and depth-first search) to enable systematic exploration of thoughts with lookahead and backtracking.

The ToT framework is illustrated below:



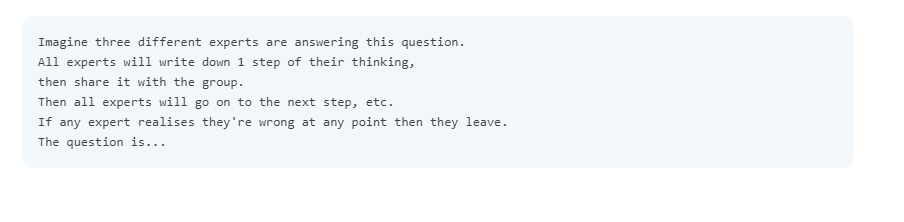
When using ToT, different tasks requires defining the number of candidates and the number of thoughts/steps. For instance, as demonstrated in the paper, Game of 24 is used as a mathematical reasoning task which requires decomposing the thoughts into 3 steps, each involving an intermediate equation. At each step, the best b=5 candidates are kept.

To perform BFS in ToT for the Game of 24 task, the LM is prompted to evaluate each thought candidate as "sure/maybe/impossible" with regard to reaching 24. As stated by the authors, "the aim is to promote correct partial solutions that can be verdicted within few lookahead trials, and eliminate impossible partial solutions based on "too big/small" commonsense, and keep the rest "maybe"". Values are sampled 3 times for each thought. The process is illustrated below:



At a high level, the main ideas of [Yao et el. (2023)(opens in a new tab)](https://arxiv.org/abs/2305.10601) and [Long (2023)(opens in a new tab)](https://arxiv.org/abs/2305.08291) are similar. Both enhance LLM's capability for complex problem solving through tree search via a multi-round conversation. One of the main difference is that [Yao et el. (2023)(opens in a new tab)](https://arxiv.org/abs/2305.10601) leverages DFS/BFS/beam search, while the tree search strategy (i.e. when to backtrack and backtracking by how many levels, etc.) proposed in [Long (2023)(opens in a new tab)](https://arxiv.org/abs/2305.08291) is driven by a "ToT Controller" trained through reinforcement learning. DFS/BFS/Beam search are generic solution search strategies with no adaptation to specific problems. In comparison, a ToT Controller trained through RL might be able learn from new data set or through self-play (AlphaGo vs brute force search), and hence the RL-based ToT system can continue to evolve and learn new knowledge even with a fixed LLM.

[Hulbert (2023)(opens in a new tab)](https://github.com/dave1010/tree-of-thought-prompting) has proposed Tree-of-Thought Prompting, which applies the main concept from ToT frameworks as a simple prompting technique, getting the LLM to evaluate intermediate thoughts in a single prompt. A sample ToT prompt is:



**Why is Prompt Engineering Crucial in Software Testing?**

1. **Improves Model Understanding**: Different prompts shed light on the AI model's functioning, assisting in troubleshooting and behavior refinement.
2. **Enhances Model Utility**: Consistent and appropriate responses to a broad spectrum of user queries make models like chatbots or virtual assistants more valuable. Prompt engineering is the key.
3. **Safety and Reliability**: It's imperative to identify and rectify potential problematic outputs for AI models in sensitive applications. Diverse prompts play a pivotal role.
   * **Real-world Consequences**: Inadequately tested AI models, especially in sectors like healthcare or autonomous vehicles, can have grave implications. This emphasizes prompt engineering's necessity.
   * **Contingency Measures**: It's beneficial for AI systems to have built-in mechanisms, like deferring to a human or providing generic answers, when faced with unfamiliar prompts.

**Key Aspects of Prompt Engineering in Software Testing**

1. **Diverse Inputs**:
   * **Examples**: For instance, testing a chatbot requires prompts from different languages, colloquialisms, and cultural contexts.
   * **Impact on Model Fairness**: Ensuring models don't discriminate against specific groups mandates testing with diverse demographic inputs.
2. **Iterative Refinement**:
   * **Feedback Loop Creation**: Continuous improvement is realized when insights from one test cycle inspire the next set of prompts.
   * **Integration with Other Testing Methods**: Prompt engineering works best when integrated with other methods, like adversarial testing.( Adversarial testing is a method for systematically evaluating ML model with the intent of learning how it behaves when provided with malicious or inadvertently harmful input)
3. **Collaboration with Model Training**:
   * **Fine-tuning with Custom Prompts**: Insights can guide further model refinement. If a style of prompt is consistently misinterpreted, it indicates a training gap.
   * **Active Learning**: Challenging examples unearthed by prompt engineering can be incorporated into model retraining.

**Comparison with Traditional Testing**  
Traditional QA methodologies often focus on fixed scenarios with predictable outputs. In contrast, prompt engineering, tailored for AI, accepts, and even expects variability. While the former might rely heavily on predefined test cases, the latter leans into adaptability and exploration, navigating the vast landscape of potential AI responses to ensure consistency and reliability.

**Challenges and Considerations**

1. **Bias Mitigation**: Testing prompts must be unbiased, ensuring the model's fairness and wide applicability.
2. **Complexity of AI Responses**: AI models, unlike traditional software, produce a broad range of responses, complicating the testing process. The complexities of AI-based model increases when they provide probabilistic results and are non-deterministic in nature.
3. **Subjectivity in Evaluating Responses**: The "correctness" of AI responses can be open to interpretation, posing unique challenges.
4. **Scalability**:
   * **Automated Prompt Generation**: Given the vastness of potential prompts, automated tools might be the answer to generate a plethora of test prompts, or even employing AI to craft challenging prompts for other AI systems.

**Different types of Testing can be performed on LLM Models**

**Testing for BIAS**

An ML model should be evaluated against different biases and actions taken to remove inappropriate bias. This may involve positive bias being deliberately introduced to counter the inappropriate bias.  
Testing with the independent dataset can often detect bas, however it can be difficult to identify all the data that causes bias because the algorithm can use combinations of seemingly unrelated features to create unwanted bias.  
A model should be tested for algorithm bias, sample bias and inappropriate bias. This may involve:

* Analysis during model’s training, evaluation, and tuning activities to identify whether algorithm bias is present.
* Reviewing the source of training data and processes used to acquire it, such that presence of sample bias can be identified.
* Reviewing the pre-processing of data as part of ML workflow to identify whether the data has been affected in a way that could cause sample bias.
* Measuring how changes in the system input affect system outputs over many interactions, and examining the results based on the groups of people objects that the system may be inappropriately biased towards or against.
* Obtaining additional information concerning the attributes of the input data potentially related to the bias and correlating it to the results. This could relate to demographic data for example which might be appropriate when testing for inappropriate bias that affects group of people.

**Testing of Transparency, Interpretability and Explainability**

Information on how the model has been implemented may be provided by the system developers. This may include the sources of training data, how labelling was conducted and how the system components have been designed, When this system is not available, it can make design of tests challenging. For e.g. When training data information is not available, then identifying potential gaps in such data and testing the impact of those gaps becomes difficult. This situation can be compared to black box and white box testing and has similar advantages and disadvantages.

* Transparency can be tested by comparing the information documented on the data and algorithm to the actual implementation and determining how closely they match.
* Different ML models provide different levels of explainability and should be selected based on the requirements for the system which may include explainability and testability.
* Method to understand explainability is through the dynamic testing of the ML Model when applying perturbations to the test data. Method exists for quantifying explainability in the manner and for providing visual explanations of it.
* Exploratory testing can also be used to better understand the relationship between the inputs and output of a model.

**Adversarial Testing**

Adversarial testing is a method for systematically evaluating ML model with the intent of learning how it behaves when provided with malicious or inadvertently harmful input.