> You don’t need to be a data scientist or a machine learning engineer – everyone can write a prompt.

Introduction

When thinking about a large language model input and output, a text prompt (sometimes accompanied by other modalities such as image prompts) is the input the model uses to predict a specific output. You don’t need to be a data scientist or a machine learning engineer – everyone can write a prompt. However, crafting the most effective prompt can be complicated. Many aspects of your prompt affect its efficacy: the model you use, the model’s training data, the model configurations, your word-choice, style and tone, structure, and context all matter. Therefore, prompt engineering is an iterative process. Inadequate prompts can lead to ambiguous, inaccurate responses, and can hinder the model’s ability to provide meaningful output.

When you chat with the Gemini chatbot,[1] you basically write prompts, however this whitepaper focuses on writing prompts for the Gemini model within Vertex AI or by using the API, because by prompting the model directly you will have access to the configuration such as temperature etc.

This whitepaper discusses prompt engineering in detail. We will look into the various prompting techniques to help you getting started and share tips and best practices to become a prompting expert. We will also discuss some of the challenges you can face while crafting prompts.

Prompt engineering

Remember how an LLM works; it's a prediction engine. The model takes sequential text as an input and then predicts what the following token should be, based on the data it was trained on. The LLM is operationalized to do this over and over again, adding the previously predicted token to the end of the sequential text for predicting the following token. The next token prediction is based on the relationship between what's in the previous tokens and what the LLM has seen during its training.

When you write a prompt, you are attempting to set up the LLM to predict the right sequence of tokens. Prompt engineering is the process of designing high-quality prompts that guide LLMs to produce accurate outputs. This process involves tinkering to find the best prompt, optimizing prompt length, and evaluating a prompt's writing style and structure in relation to the task. In the context of natural language processing and LLMs, a prompt is an input provided to the model to generate a response or prediction.

These prompts can be used to achieve various kinds of understanding and generation tasks such as text summarization, information extraction, question and answering, text classification, language or code translation, code generation, and code documentation or reasoning.

Please feel free to refer to Google's prompting guides[2,3] with simple and effective prompting examples.

When prompt engineering, you will start by choosing a model. Prompts might need to be optimized for your specific model, regardless of whether you use Gemini language models in Vertex AI, GPT, Claude, or an open source model like Gemma or LLaMA.

Besides the prompt, you will also need to tinker with the various configurations of a LLM.

LLM output configuration

Once you choose your model you will need to figure out the model configuration. Most LLMs come with various configuration options that control the LLM's output. Effective prompt engineering requires setting these configurations optimally for your task.

Output length

An important configuration setting is the number of tokens to generate in a response. Generating more tokens requires more computation from the LLM, leading to higher energy consumption, potentially slower response times, and higher costs.

Reducing the output length of the LLM doesn't cause the LLM to become more stylistically or textually succinct in the output it creates, it just causes the LLM to stop predicting more tokens once the limit is reached. If your needs require a short output length, you'll also possibly need to engineer your prompt to accommodate.

Output length restriction is especially important for some LLM prompting techniques, like ReAct, where the LLM will keep emitting useless tokens after the response you want.

Sampling controls

LLMs do not formally predict a single token. Rather, LLMs predict probabilities for what the next token could be, with each token in the LLM's vocabulary getting a probability. Those token probabilities are then sampled to determine what the next produced token will be. Temperature, top-K, and top-P are the most common configuration settings that determine how predicted token probabilities are processed to choose a single output token.

Temperature

Temperature controls the degree of randomness in token selection. Lower temperatures are good for prompts that expect a more deterministic response, while higher temperatures can lead to more diverse or unexpected results. A temperature of 0 (greedy decoding) is deterministic: the highest probability token is always selected (though note that if two tokens have the same highest predicted probability, depending on how tiebreaking is implemented you may not always get the same output with temperature 0).

Temperatures close to the max tend to create more random output. And as temperature gets higher and higher, all tokens become equally likely to be the next predicted token.

The Gemini temperature control can be understood in a similar way to the softmax function used in machine learning. A low temperature setting mirrors a low softmax temperature (T), emphasizing a single, preferred temperature with high certainty. A higher Gemini temperature setting is like a high softmax temperature, making a wider range of temperatures around the selected setting more acceptable. This increased uncertainty accommodates scenarios where a rigid, precise temperature may not be essential like for example when experimenting with creative outputs.

Top-K and top-P

Top-K and top-P (also known as nucleus sampling)[4] are two sampling settings used in LLMs to restrict the predicted next token to come from tokens with the top predicted probabilities. Like temperature, these sampling settings control the randomness and diversity of generated text.

- **Top-K** sampling selects the top K most likely tokens from the model's predicted distribution. The higher top-K, the more creative and varied the model's output; the lower top-K, the more restive and factual the model's output. A top-K of 1 is equivalent to greedy decoding.
- **Top-P** sampling selects the top tokens whose cumulative probability does not exceed a certain value (P). Values for P range from 0 (greedy decoding) to 1 (all tokens in the LLM's vocabulary).

The best way to choose between top-K and top-P is to experiment with both methods (or both together) and see which one produces the results you are looking for.