Prompt Engineering for LLMs

In the realm of natural language processing, the art of crafting effective prompts plays a pivotal role in harnessing the full potential of LLMs. This tutorial offers a practical guide for data scientists and machine learning practitioners to strategically design prompts that optimize the performance of LLMs. The focus of this tutorial will be on ChatGPT and Gemini, demonstrating best practices and methods for key use-cases in prompt engineering.

**Table of Contents:**

[*Setting up the Environment 2*](#_xtoi0vegyg7u)

[OpenAI’s ChatGPT 2](#_3nshh86p41wk)

[Google’s Gemini 3](#_vzwcodz1o6l2)

[*Prompt Refining 4*](#_b29ium8mo4f)

[*Prompt Chaining 5*](#_aal08hjtpae8)

[*Shot Prompting and Chain-of-Thought Prompting 7*](#_7awmo4cn8hh)

[*Summarization 10*](#_9psobx7uctvr)

[*Content Generation 11*](#_n128xaccgfkz)

[*Translation 12*](#_b0gmvbrmfv93)

[*Code Generation 14*](#_qe5p37jokus3)

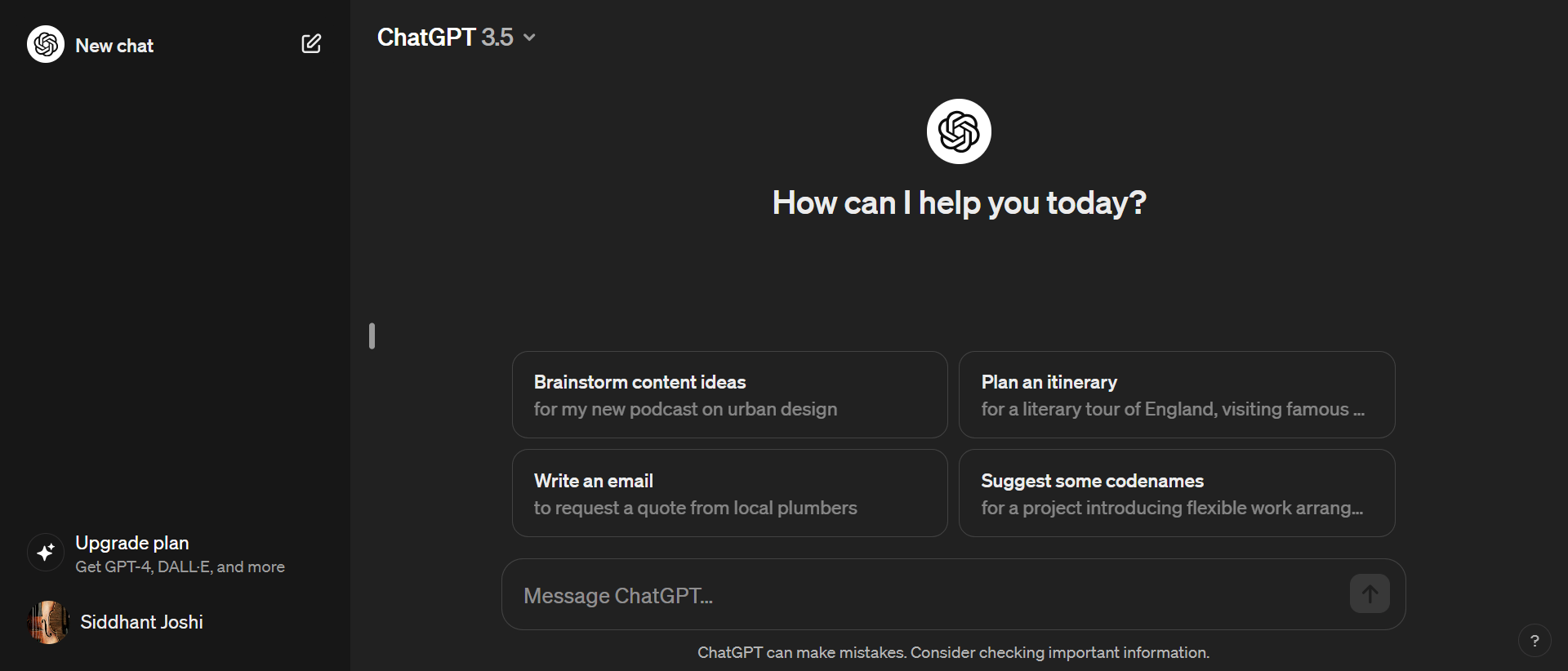
[*Code Interpretation 16*](#_pkpibiw5xlh1)

# Setting up the Environment

## OpenAI’s ChatGPT

ChatGPT is an AI chatbot developed by OpenAI based on the GPT (Generative Pre-trained Transformer) architecture. It is designed to engage in natural language conversations with users, providing responses that are contextually relevant and coherent. The model is capable of engaging in conversations with user-based prompts, answer questions/provide information, and is a general purpose LLM. It also has a subscription version, called ChatGPT-4, which includes all of the original features in addition to image generation (via DALL-E) and web browsing.

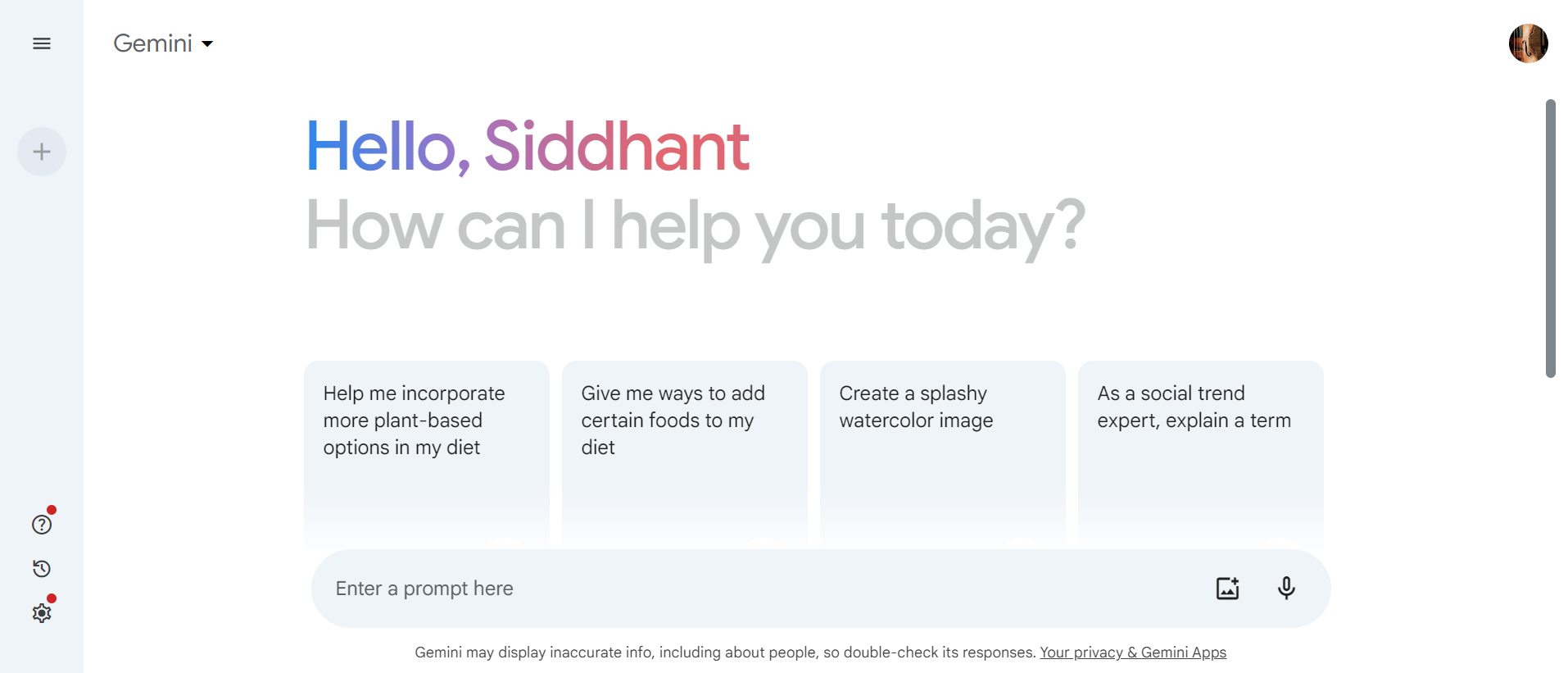
1. Navigate to <https://chat.openai.com/auth/login> and select *Sign Up* to create an account.
2. After creating an account, navigate back to the above link and login to your account.
3. The screen should look like the image below:



## Google’s Gemini

Formerly known as Bard, Gemini is an LLM developed and deployed by Google. It functions similarly to MS Copilot, however, it lacks the ability to tune conversation styles and domain-specific models. It runs on software managed by Google and is designed to engage in conversations with user-based prompts, answer questions, and put together responses based on information it gathers from the Internet.

1. Navigate to <https://gemini.google.com/app> and select or create a Google account (you may need to use a personal gmail account if your organization email does not work).
2. After creating an account, navigate back to the above link and you should see the following:



# Prompt Refining

Prompt refining in the context of prompt engineering refers to the process of reformulating and rephrasing prompts to achieve more accurate and relevant outputs from AI models. This involves adjusting the wording, structure, and specificity of the prompts to guide the AI effectively. By refining prompts, users can enhance the clarity and precision of their queries, leading to better performance and results from the AI.

For best practices on how to effectively engineer prompts, refer to [this guide](https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api) from OpenAI.

**Exercise 1:**

| **PROMPT:** |
| --- |
| Tell me about data visualization. |
| **REFINED PROMPT:** |
| *Insert you* Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, maps, and other visual tools, data visualization provides an accessible way to see and understand trends, outliers, and patterns in data. Here are some key aspects of data visualization: Importance of Data Visualization  1. **Simplification**: Complex data sets can be simplified into graphical representations that are easier to understand. 2. **Insight**: Visualizing data helps in identifying patterns, correlations, and insights that might be missed in text-based data. 3. **Communication**: Effective visualizations can convey information quickly and clearly, making it easier to share findings with others. 4. **Decision Making**: By presenting data visually, stakeholders can make informed decisions based on clear evidence.  Types of Data Visualizations  1. **Charts**:    * **Bar Chart**: Useful for comparing quantities across categories.    * **Line Chart**: Ideal for showing trends over time.    * **Pie Chart**: Shows proportions of a whole.    * **Histogram**: Represents the distribution of data over a continuous interval. 2. **Graphs**:    * **Scatter Plot**: Displays values for typically two variables for a set of data.    * **Bubble Chart**: Similar to a scatter plot but includes a third variable as the size of the bubbles. 3. **Maps**:    * **Geographic Maps**: Visualize data across geographical areas.    * **Heat Maps**: Show data intensity over a geographic area or a grid. 4. **Tables**: Provide a way to look at detailed data points in a structured format. 5. **Infographics**: Combine multiple types of data visualizations into a single, coherent visual representation.  Tools for Data Visualization  * **Matplotlib**: A plotting library for the Python programming language. * **Seaborn**: Based on Matplotlib, it provides a high-level interface for drawing attractive and informative statistical graphics. * **Tableau**: A powerful tool for creating a wide range of visualizations and dashboards. * **Power BI**: A business analytics tool by Microsoft for interactive visualizations. * **D3.js**: A JavaScript library for producing dynamic, interactive data visualizations in web browsers. * **Plotly**: Supports interactive graphing libraries for Python, R, MATLAB, and JavaScript.  Best Practices for Data Visualization  1. **Choose the Right Type**: Select the appropriate visualization type for your data and the message you want to convey. 2. **Keep it Simple**: Avoid clutter and focus on the key information. 3. **Use Color Wisely**: Use colors to highlight important data points, but ensure they are not overwhelming or misleading. 4. **Label Clearly**: Make sure all axes, legends, and data points are clearly labeled. 5. **Maintain Consistency**: Use consistent scales and formats to make comparisons easier.  Challenges in Data Visualization  * **Overloading Information**: Including too much data can make visualizations confusing. * **Misleading Representations**: Poor design choices can lead to misinterpretation of the data. * **Data Quality**: Inaccurate or incomplete data can lead to incorrect conclusions. * **Accessibility**: Ensuring that visualizations are accessible to all users, including those with color blindness or other visual impairments.   Data visualization is a crucial skill for data scientists, analysts, and business professionals to present data in a compelling and informative manner.  *r response here…* |

**Exercise 2:**

| **PROMPT:** |
| --- |
| Find a nice dataset that contains and includes petal lengths. Explore the data and tell me what results you get from exploring the data. |
| **REFINED PROMPT:** |
| A commonly used dataset that includes petal lengths is the Iris dataset. This dataset contains measurements of various features of iris flowers, including petal lengths, petal widths, sepal lengths, and sepal widths, across three different species of iris flowers (Iris setosa, Iris versicolor, and Iris virginica).  Let's explore the Iris dataset. I'll load the dataset, conduct a basic exploration, and provide you with insights from the data. Loading the Dataset python  Copy code  import seaborn as sns  import pandas as pd  # Load the Iris dataset  iris = sns.load\_dataset('iris') Exploring the Dataset First, let's take a look at the first few rows of the dataset to understand its structure:  python  Copy code  iris.head()  Here is the dataset structure:   |  | **sepal\_length** | **sepal\_width** | **petal\_length** | **petal\_width** | **species** | | --- | --- | --- | --- | --- | --- | | 0 | 5.1 | 3.5 | 1.4 | 0.2 | setosa | | 1 | 4.9 | 3.0 | 1.4 | 0.2 | setosa | | 2 | 4.7 | 3.2 | 1.3 | 0.2 | setosa | | 3 | 4.6 | 3.1 | 1.5 | 0.2 | setosa | | 4 | 5.0 | 3.6 | 1.4 | 0.2 | setosa |  Summary Statistics Let's look at the summary statistics for the petal length column.  python  Copy code  iris['petal\_length'].describe()  This will provide the following output:  shell  Copy code  count 150.000000  mean 3.758667  std 1.765298  min 1.000000  25% 1.600000  50% 4.350000  75% 5.100000  max 6.900000  Name: petal\_length, dtype: float64 Distribution of Petal Lengths Let's visualize the distribution of petal lengths across different species.  python  Copy code  import matplotlib.pyplot as plt  plt.figure(figsize=(10, 6))  sns.histplot(data=iris, x='petal\_length', hue='species', kde=True)  plt.title('Distribution of Petal Lengths by Species')  plt.xlabel('Petal Length (cm)')  plt.ylabel('Frequency')  plt.show() Scatter Plot of Petal Length vs. Petal Width We can also visualize the relationship between petal length and petal width.  python  Copy code  plt.figure(figsize=(10, 6))  sns.scatterplot(data=iris, x='petal\_length', y='petal\_width', hue='species')  plt.title('Petal Length vs. Petal Width by Species')  plt.xlabel('Petal Length (cm)')  plt.ylabel('Petal Width (cm)')  plt.show() Insights from the Data  1. **Summary Statistics**: The mean petal length is approximately 3.76 cm, with a standard deviation of 1.77 cm. The petal lengths range from 1.0 cm to 6.9 cm. 2. **Distribution**: The histogram shows distinct distributions for each species.    * Iris setosa has shorter petal lengths, mostly concentrated around 1.0 to 1.9 cm.    * Iris versicolor has intermediate petal lengths, ranging from 3.0 to 5.1 cm.    * Iris virginica has longer petal lengths, ranging from 4.5 to 6.9 cm. 3. **Relationship**: The scatter plot shows a positive correlation between petal length and petal width. Each species forms a distinct cluster, indicating that petal length and petal width can be used to distinguish between species.   Would you like to delve deeper into any specific aspect of the dataset or perform additional analyses? |

# 

# Prompt Chaining

Prompt chaining in the context of prompt engineering involves linking multiple prompts together to facilitate more complex and coherent dialogues. This technique allows for building upon previous responses, enabling the AI to maintain context and continuity throughout the conversation. By chaining prompts, users can create sophisticated interactions that mimic natural, multi-turn conversations, enhancing the depth and relevance of the AI's responses.

**Exercise 1:**

| **PROMPT:** |
| --- |
| How do transformers work? |
| **CHATGPT:** |
| Transformers are a type of deep learning model architecture that has revolutionized natural language processing (NLP) and other fields of machine learning. Here’s a simplified overview of how transformers work:   1. **Self-Attention Mechanism**: The core of transformers is the self-attention mechanism. It allows the model to weigh the importance of different words (tokens) in a sentence relative to each other. This mechanism enables the model to consider the context of a word by looking at all other words in the sentence simultaneously. 2. **Encoding**: The input to a transformer model consists of tokens representing words or subwords (e.g., from tokenizers like BERT). Each token is associated with three vectors: the input embedding (word embedding), position encoding (to indicate the position of the token in the sequence), and segment embedding (to distinguish between different segments, e.g., in question-answering tasks). 3. **Transformer Blocks**: A transformer is composed of multiple layers of transformer blocks. Each block typically includes:    * **Multi-Head Attention**: This is where the self-attention mechanism operates. It computes attention scores for multiple projections (heads) of both the input embeddings and their own outputs from previous layers.    * **Feedforward Neural Network**: After attention, each position in the sequence passes through a feedforward neural network, which is applied independently to each position. 4. **Layer Normalization and Residual Connections**: After each sub-layer (attention and feedforward), a residual connection around the sub-layer followed by layer normalization is applied. This helps in stabilizing training and speeds up convergence. 5. **Decoder (for Sequence-to-Sequence Models)**: In tasks like language translation, transformers use an encoder-decoder architecture. The decoder is similar to the encoder but also has an additional attention mechanism called encoder-decoder attention, which helps the model focus on relevant parts of the input sentence while generating the output sentence. 6. **Output**: The final layer of the decoder generates a probability distribution over the target vocabulary (in language generation tasks) or directly produces the output sequence. 7. **Training**: Transformers are typically trained with large amounts of text data using unsupervised or semi-supervised learning techniques like masked language modeling (where random words are masked and the model learns to predict them) or sequence-to-sequence tasks (where the model learns to map one sequence to another).   Transformers have shown remarkable performance in NLP tasks due to their ability to capture long-range dependencies and contextual information effectively, making them widely adopted in various applications such as text classification, translation, summarization, and more. |
| **GEMINI:** |
| The term "transformer" can refer to two different things: electrical transformers and transformers in machine learning. They both deal with transferring or transforming something, but in completely different ways.   * **Electrical transformers** are passive electrical devices that transfer electrical energy from one circuit to another circuit by changing the voltage and current levels. They work based on the principle of electromagnetic induction: when an alternating current flows through a coil of wire, it creates a magnetic field. This magnetic field can then induce a current in a nearby coil of wire. By changing the number of turns in each coil, the voltage and current levels can be increased or decreased. <https://www.maddoxtransformer.com/electrical-transformers> * **Machine learning transformers** are a specific type of neural network architecture that's particularly good at understanding relationships between words in a sentence. Unlike traditional neural networks that process data sequentially, transformers can analyze all parts of a sentence simultaneously using a mechanism called "attention." This allows them to identify long-range dependencies and capture complex relationships between words. Transformers are widely used in natural language processing (NLP) tasks like machine translation, text summarization, and question answering. <https://towardsdatascience.com/transformers-141e32e69591> |

**Exercise 2:**

| **PROMPT:** |
| --- |
| Give me a more detailed explanation of the self-attention mechanism. |
| **CHATGPT:** |
| Certainly! The self-attention mechanism is a fundamental component of transformer models, crucial for capturing relationships between different words (or tokens) in a sequence. Here's a detailed explanation of how self-attention works: Key Concepts in Self-Attention  1. **Input Representation**: Each token in the input sequence is represented as three vectors:    * **Word Embedding**: Represents the meaning of the word.    * **Positional Encoding**: Represents the position of the word in the sequence.    * **Segment Embedding**: Optionally used to distinguish different segments (e.g., sentences in a document). 2. **Query, Key, and Value**: Before applying self-attention, the input embeddings are transformed into three sets of vectors:    * **Query vectors (Q)**: These are linear transformations of the input embeddings that are used to score each word against every other word in the sequence.    * **Key vectors (K)**: These are also linear transformations of the input embeddings and are used to create the attention scores.    * **Value vectors (V)**: These are linear transformations of the input embeddings and serve as the actual input to the attention mechanism.   Each of these transformations (for Q, K, and V) is parameterized by learned weights.   1. **Computing Attention Scores**:    * For each word iii in the sequence, the attention score with respect to word jjj is computed using the dot product of the query vector of word iii (Qiii) and the key vector of word jjj (Kjjj): Attention(Qi,Kj)=Qi⋅Kjdk\text{Attention}(Q\_i, K\_j) = \frac{Q\_i \cdot K\_j}{\sqrt{d\_k}}Attention(Qi​,Kj​)=dk​​Qi​⋅Kj​​ where dkd\_kdk​ is the dimensionality of the key vectors (typically the square root of the dimension of the query and key vectors to stabilize gradients). 2. **Softmax and Attention Weights**:    * After computing the attention scores, a softmax function is applied to obtain attention weights that sum to 1 across all words: Attention\_Weights(Qi,Kj)=softmax(Attention(Qi,Kj))\text{Attention\\_Weights}(Q\_i, K\_j) = \text{softmax}(\text{Attention}(Q\_i, K\_j))Attention\_Weights(Qi​,Kj​)=softmax(Attention(Qi​,Kj​)) 3. **Weighted Sum of Values**:    * The weighted sum of the value vectors (V) using the attention weights gives the output of the self-attention mechanism for each word iii: SelfAttention(Qi,K,V)=∑jAttention\_Weights(Qi,Kj)⋅Vj\text{SelfAttention}(Q\_i, K, V) = \sum\_j \text{Attention\\_Weights}(Q\_i, K\_j) \cdot V\_jSelfAttention(Qi​,K,V)=j∑​Attention\_Weights(Qi​,Kj​)⋅Vj​ This step computes a representation of each word considering its interactions (attention) with all other words in the sequence. 4. **Multi-Head Attention**:    * In practice, transformers use multiple sets of Q, K, and V vectors (called heads). Each head learns different representations of the relationships between words. The outputs of multiple attention heads are concatenated and linearly transformed to maintain the model's expressiveness and robustness.  Benefits of Self-Attention  * **Long-Range Dependencies**: Self-attention allows the model to capture dependencies between words that are far apart in the sequence, which is crucial for tasks like understanding context in long sentences or documents. * **Parallelization**: Unlike recurrent neural networks (RNNs) which process tokens sequentially, transformers can process all tokens in parallel, making them more efficient for training and inference. * **Scalability**: Self-attention's ability to scale with the sequence length (due to its constant-time complexity per layer) makes transformers suitable for handling large inputs, such as entire documents or long conversations.   In summary, self-attention is a powerful mechanism in transformers that enables the model to capture relationships between words in a sequence, allowing for context-aware representations that have proven highly effective in various natural language processing tasks. |
| **GEMINI:** |
| Show drafts  volume\_up  The self-attention mechanism is a core component of transformers, particularly crucial in their ability to understand complex relationships within sequences. Here's a breakdown of how it works:   1. **Input Embedding:** Imagine a sentence as the input. Each word is converted into a vector representation called a word embedding. This embedding captures the semantic meaning of the word. 2. **Query, Key, Value Creation:** The self-attention mechanism works by creating three new vectors for each word embedding:    * **Query (Q):** This vector represents what information a specific word is looking for in the sequence.    * **Key (K):** This vector captures the informative content of each word.    * **Value (V):** This vector holds the actual information from each word embedding.   These Q, K, and V vectors are created by applying linear transformations to the original word embeddings.   1. **Attention Scores:** The core of self-attention is calculating an "attention score" for each pair of words in the sentence. This score indicates how relevant one word (with its query vector) is to another word (with its key vector). The higher the score, the more attention the first word pays to the information carried by the second word. 2. **Weighted Values:** Using the attention scores, a weighted sum of the value vectors is calculated. Each value vector is weighted based on its corresponding attention score. This essentially creates a context-aware representation for each word, considering the most relevant information from other words in the sequence. 3. **Output:** The final step combines the original word embedding with the context-aware representation created in the previous step. This enriched representation is then fed into a neural network layer for further processing.   **Benefits of Self-Attention:**   * **Contextual understanding:** By considering relationships between words, self-attention allows transformers to capture the context in which a word is used, leading to a more nuanced understanding of the sentence. * **Long-range dependencies:** Unlike traditional sequential models, self-attention can identify dependencies between words that are far apart in the sequence. This is crucial for tasks like sentiment analysis where word order might not be strictly sequential.   **Further Exploration:**  If you'd like to delve deeper, you can explore concepts like:   * Scaled dot-product attention: This is a specific method used to calculate attention scores in transformers. * Multi-head attention: Transformers often use multiple self-attention heads in parallel, allowing them to capture different aspects of the relationships between words. |

**Exercise 3:**

| **PROMPT:** |
| --- |
| Give me a conceptual explanation of the Q, K, and V matrices. |
| **CHATGPT:** |
| Certainly! The Q, K, and V matrices are fundamental components of the self-attention mechanism used in transformer models. Here's a conceptual explanation of what they represent and how they are used: Conceptual Explanation of Q, K, and V Matrices  1. **Input Embeddings**:    * Before self-attention is applied, each token (word or subword) in the input sequence is embedded into three vectors:      + **Query Vector (Q)**: This vector represents the token's representation for querying other tokens in the sequence. In essence, it captures what information the token wants to retrieve from other tokens.      + **Key Vector (K)**: This vector represents the token's representation for being queried by other tokens. It indicates the properties or features of the token that other tokens might be interested in.      + **Value Vector (V)**: This vector represents the actual content or information associated with the token. It is what gets passed around and used to compute the output of the self-attention mechanism. 2. **Linear Transformation**:    * Each of these vectors (Q, K, and V) is derived from the input embeddings by applying learned linear transformations (parameterized by weights):      + Q = Input embeddings × WQW\_QWQ​      + K = Input embeddings × WKW\_KWK​      + V = Input embeddings × WVW\_VWV​ where WQW\_QWQ​, WKW\_KWK​, and WVW\_VWV​ are weight matrices specific to Q, K, and V respectively. 3. **Purpose and Usage**:    * **Query (Q)**: This vector is used to determine how much attention should be given to each of the other tokens in the sequence. It is paired with the key vectors to compute attention scores, indicating the relevance or importance of each token relative to the current token.    * **Key (K)**: This vector is used to compute the attention scores with respect to other tokens. It helps in determining how similar or relevant each token is to other tokens in the sequence.    * **Value (V)**: This vector contains the actual information that gets propagated through the attention mechanism. It is used to compute the weighted sum that forms the output of the self-attention operation. 4. **Computing Attention**:    * Once Q, K, and V are computed for each token, attention scores are calculated by taking the dot product of Q and K. These scores determine how much focus each token should place on other tokens.    * The attention scores are then normalized using a softmax function to obtain attention weights, which indicate how much each token should contribute to the output of the self-attention mechanism. 5. **Multi-Head Attention**:    * In practice, transformers use multiple sets (or heads) of Q, K, and V vectors. Each head learns different representations of relationships between tokens, providing the model with multiple perspectives on how tokens interact.   In summary, the Q, K, and V matrices are derived from the input embeddings and serve distinct roles in the self-attention mechanism of transformers. They enable the model to compute attention scores, determine relevance between tokens, and propagate useful information through the network, ultimately contributing to the model's ability to understand and generate sequences effectively in tasks like natural language processing. |
| **GEMINI:** |
| Sure, conceptually, the Q, K, and V matrices in transformers are created by applying linear transformations to the word embeddings, and they capture different aspects of the words' relationship within a sentence. Here's a breakdown of their roles:   |  |  | | --- | --- | | **Matrix** | **Role** | | Q (Query) | Represents what information a specific word (represented by its embedding) is looking for in the sentence. It's like creating a question based on the word's meaning. | | K (Key) | Captures the informative content of each word embedding. It acts like an answer key that provides information relevant to the query. | | V (Value) | Holds the actual information from each word embedding. It contains the essential data that the query is seeking from the key. |   drive\_spreadsheetExport to Sheets  Imagine a scenario where you're reading a sentence and trying to understand the relationships between the words. The Q matrix (query) represents your questions as you read each word. You're asking "what information is relevant to this word?" The K matrix (key) acts like a source of information, providing clues based on each word's content. Finally, the V matrix (value) holds the actual information you're looking for, based on the word you're querying and the relevant context provided by the key.  By comparing these Q, K, and V matrices, the self-attention mechanism can determine how relevant each word is to others in the sentence. This allows the transformer model to understand the context in which each word is used and capture the overall meaning of the sentence more effectively. |

# 

# Shot Prompting and Chain-of-Thought Prompting

Chain-of-thought prompting is a technique where the prompt asks the AI to provide step-by-step reasoning, allowing it to break down complex problems into manageable parts. This approach helps in understanding the logic and sequence behind the AI's responses, ensuring more accurate and transparent outputs. On the other hand, shot prompting involves providing examples within the prompt to guide the AI's responses. By including these examples, users can set clear expectations and demonstrate the desired format or style, improving the relevance and quality of the outputs. Together, these methods enhance the AI's ability to handle intricate queries and deliver precise, context-aware results.

In this section, we want the answer to the following problem:

“John's ship can travel at 7 miles per hour. He is sailing from 11 AM to 3 PM. He   
then travels back at a rate of 8 mph. How long does it take him to get back?”

The first step is to write the problem as a **question-answer pair**. This approach is commonly used for training or querying language models, especially for mathematical reasoning or word problems. In this format, you present a clear question followed by an answer. Current LLMs are trained in a way that encourages detailed and comprehensive responses, which can increase the likelihood of providing accurate information. Therefore, it is likely that you will receive accurate answers even without explicitly implementing chain-of-thought prompting. However, for more complex problems where explicit chain-of-thought prompting is necessary, the strategies learned from this example will still be applicable.

⏳ **Using question-answer pair format:**

| **PROMPT:** |
| --- |
| John's ship can travel at 7 miles per hour. He is sailing from 11 AM to 3 PM. He then travels back at a rate of 8 mph. How long does it take him to get back? |
| **REVISED PROMPT:** |
| John's ship can travel at 7 miles per hour. He is sailing from 11 AM to 3 PM. He then travels back at a rate of 8 mph. How long does it take him to get back? Please think step by step and explain how you came to the conclusion |

Now, create a similar problem to the original and find its answer (just the number of hours, not the solution). Place this new question-answer pair before the original question.

This is known as shot prompting, where examples are provided within the prompt to guide the AI's responses. Generally, there are three different levels: zero-shot, one-shot, and few-shot prompting. Zero-shot prompting involves asking the AI to respond to a prompt without any examples. One-shot prompting provides one example to illustrate the desired response. Few-shot prompting includes a few examples to give the AI a better understanding of the task.

⏳ **Adding another question:**

| **PROMPT:** |
| --- |
| Question: Tom's ship can travel at 10 miles per hour. He is sailing from 1 to 4 PM. He then travels back at a rate of 6 mph. How long does it take him to get back?  Answer: 5 hours.  Question: John's ship can travel at 7 miles per hour. He is sailing from 11 AM to 3 PM. He  then travels back at a rate of 8 mph. How long does it take him to get back? Answer: |
| **CHATGPT:** |
| *Insert your response here…* |
| **GEMINI:** |
| *Insert your response here…* |

Now, instead of providing just a brief answer,, we will include detailed reasoning and steps required to reach the answer. Expand your answer into a full solution. Prompt the LLMs again and observe the differences. This approach is known as chain-of-thought prompting. This works because it encourages the LLMs to get answers to multiple small answers (which it is good at) instead of getting an answer to one large question (which it is not great at).

✅ **Using chain-of-thought prompting:**

| **PROMPT:** |
| --- |
| Question: Tom's ship can travel at 10 miles per hour. He is sailing from 1 to 4 PM. He then travels back at a rate of 6 mph. How long does it take him to get back?  Answer:  1. Outbound Journey: Calculate the distance traveled by multiplying the speed (10 mph) by the time (3 hours). Tom traveled 30 miles.  2. Return Journey: Divide the outbound distance (30 miles) by the return speed (6 mph) to find the return time. Tom returned in 5 hours.  Question: John's ship can travel at 7 miles per hour. He is sailing from 11 AM to 3 PM. He then travels back at a rate of 8 mph. How long does it take him to get back?  Answer: |
| **CHATGPT:** |
| *Insert your response here…* |
| **GEMINI:** |
| *Insert your response here…* |

# 

# Summarization

Summarization in the context of LLM prompt engineering involves creating concise and coherent summaries of longer texts. By crafting prompts that ask the AI to distill the main points or key information from a document, users can efficiently generate summaries that capture the essence of the content, making it easier to understand and digest large volumes of information quickly.

**Exercise:**

| **PROMPT:** |
| --- |
| Explain the main findings of “Attention is All You Need” by Vaswani et al. |
| **CHATGPT:** |
| *Insert your response here…* |
| **GEMINI:** |
| *Insert your response here…* |

# 

# Content Generation

Content generation leverages LLM prompt engineering to create various forms of content, such as articles, stories, or social media posts. By designing prompts that specify the topic, tone, and style, users can guide the AI to produce high-quality and contextually appropriate content. This application is particularly useful for automating content creation processes and enhancing creative workflows.

**Exercise:**

| **PROMPT:** |
| --- |
| Create a resume for an undergraduate CS student at UCSD looking for jobs in machine learning and data science. |
| **CHATGPT:** |
| *Insert your response here…* |
| **GEMINI:** |
| *Insert your response here…* |

# 

# Translation

Translation in LLM prompt engineering involves converting text from one language to another while maintaining the original meaning and context. By structuring prompts to indicate the source and target languages, users can utilize the AI's capabilities to perform accurate and nuanced translations, facilitating communication and understanding across different languages and cultures.

**Exercise 1:**

| **PROMPT:** |
| --- |
| Can you translate this from French to English: Je ne parle Francais. |
| **CHATGPT:** |
| *Insert your response here…* |
| **GEMINI:** |
| *Insert your response here…* |

**Exercise 2:**

| **PROMPT:** |
| --- |
| Can you translate to English: Toi la nguoi Viet. |
| **CHATGPT:** |
| *Insert your response here…* |
| **GEMINI:** |
| *Insert your response here…* |

**Exercise 3:**

| **PROMPT:** |
| --- |
| How do you say this in Japanese: Sorry, I don’t speak Japanese |
| **CHATGPT:** |
| *Insert your response here…* |
| **GEMINI:** |
| *Insert your response here…* |

**Exercise 4:**

| **PROMPT:** |
| --- |
| What does this mean: Hella fit; at first I was triggered and shook, but then looked closer, saw it was popping and legit, in fact it is totally lit! |
| **CHATGPT:** |
| *Insert your response here…* |
| **GEMINI:** |
| *Insert your response here…* |

# 

# Code Generation

Code generation uses LLM prompt engineering to create snippets of code based on specific requirements or descriptions. By formulating prompts that outline the desired functionality or logic, users can have the AI generate code in various programming languages. This application is valuable for accelerating development processes and assisting with coding tasks.

**Exercise 1:**

| **PROMPT:** |
| --- |
| Can you provide Python code to read in a dataframe with Name, Dept, remove missing values, then group by Dept? Write results to a json file. |
| **CHATGPT:** |
| *Insert your response here…* |
| **GEMINI:** |
| *Insert your response here…* |

**Exercise 2 (follow-up from Exercise 1):**

| **PROMPT:** |
| --- |
| Can you provide the equivalent PySpark code? |
| **CHATGPT:** |
| *Insert your response here…* |
| **GEMINI:** |
| *Insert your response here…* |

**Exercise 3:**

| **PROMPT:** |
| --- |
| Write Python code that uses Pandas to remove outliers from the ./titanic.csv dataset by limiting the 'Age' and 'Fare' columns to values within the 1st and 99th percentiles. Report the number of outliers before and after handling. |
| **CHATGPT:** |
| *Insert your response here…* |
| **GEMINI:** |
| *Insert your response here…* |

# 

# Code Interpretation

With programming as the key method with which to implement our ideas, it is important to have a system to produce and accurately interpret code. Python is a prominent programming language in the data scientist’s toolbox and LLMs can be leveraged to interpret segments of code accurately and efficiently. Below we offer some examples of how to formulate prompts to get the most effective interpretations of code.

**Exercise[[1]](#footnote-1):**

| **PROMPT:** |
| --- |
| Explain the following code: def find\_max(list):  max\_val = list[0]  for i in range(len(list)):  for j in range(i, len(list)):  if list[j] > max\_val:  max\_val = list[j]  return max\_val |
| **CHATGPT:** |
| *Insert your response here…* |
| **GEMINI:** |
| *Insert your response here…* |

1. The code in the exercise prompt is from Konfuzio: <https://konfuzio.com/en/python-tutorial-complexity/> [↑](#footnote-ref-1)