

Prompt Engineering for LLMs - **Solution**

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:

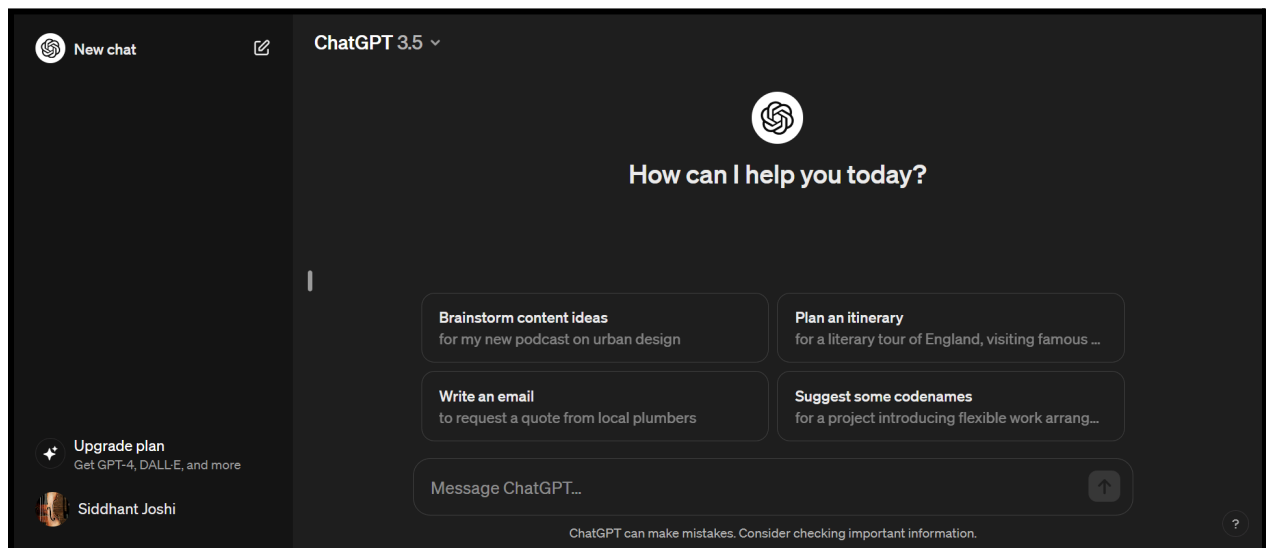
| | |
|---|-----------|
| <i>Setting up the Environment.....</i> | <i>2</i> |
| OpenAI's ChatGPT..... | 2 |
| Google's Gemini..... | 3 |
| <i>Prompt Refining.....</i> | <i>4</i> |
| <i>Prompt Chaining.....</i> | <i>5</i> |
| <i>Shot Prompting and Chain-of-Thought Prompting.....</i> | <i>6</i> |
| <i>Summarization.....</i> | <i>9</i> |
| <i>Content Generation.....</i> | <i>13</i> |
| <i>Translation.....</i> | <i>17</i> |
| <i>Code Generation.....</i> | <i>20</i> |
| <i>Code Interpretation.....</i> | <i>27</i> |

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.

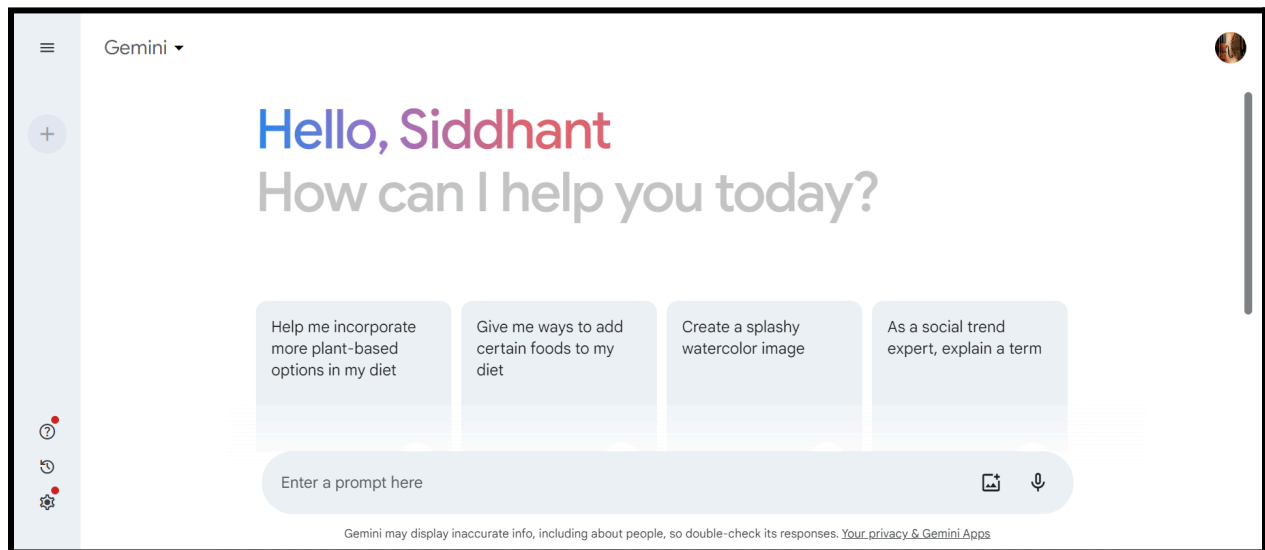
1. Navigate to <https://chat.openai.com/auth/login> and select *Sign Up* to create an account.
2. After creating an account, navigate back to the above link and login to your account.
3. The screen should look like the image below:



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](#) from OpenAI.

Exercise 1:

PROMPT:

Tell me about data visualization.

REFINED PROMPT:

Compare and contrast the use cases of line plots, bar charts, and histograms.

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:

Download the 'Iris' dataset. Conduct exploratory data analysis by calculating the mean and standard deviation for each species' petal length. Plot the results in a bar graph.

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.

Responses:

ChatGPT: <https://chatgpt.com/share/eb7fd9d4-5757-48bb-bea2-bbf956ec0f29>

Gemini: <https://g.co/gemini/share/a2ce187dcbd6>

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.

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:

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:

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:

2.33 hours.

GEMINI:

Answer: 4.5 hours

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:

1. Outbound Journey: Calculate the distance traveled by multiplying the speed (7 mph) by the time (4 hours). John traveled 28 miles.
2. Return Journey: Divide the outbound distance (28 miles) by the return speed (8 mph) to find the return time. John returned in 3.5 hours.

GEMINI:

Following the same logic as for Tom's trip, here's how long it takes John to get back:

1. **Outbound Journey:**

* Time spent sailing = 3 PM - 11 AM = 4 hours

* Distance traveled = Speed (7 mph) * Time (4 hours) = 28 miles

2. **Return Journey:**

* Time to return = Distance (28 miles) / Return Speed (8 mph) = 3.5 hours

John takes 3.5 hours to return to his starting point.

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:

"Attention is All You Need" is a seminal paper in the field of natural language processing, specifically in the area of neural machine translation. The paper, authored by Ashish Vaswani et al. and presented at the Neural Information Processing Systems (NeurIPS) conference in 2017, introduces the Transformer model, which relies solely on self-attention mechanisms and does not use recurrent or convolutional structures.

Here are the main findings and contributions of the paper:

1. Transformer Architecture:

The paper introduces the Transformer architecture, which employs self-attention mechanisms to capture contextual information from input sequences.

The model eliminates the need for recurrent layers, making it highly parallelizable and more efficient in terms of training compared to traditional sequence-to-sequence models.

2. Self-Attention Mechanism:

The self-attention mechanism allows the model to weigh different parts of the input sequence differently during the encoding process.

It enables the model to consider global dependencies and relationships between words,

alleviating the vanishing gradient problem associated with long-range dependencies in traditional recurrent models.

3. Positional Encoding:

Since the Transformer model does not inherently understand the order of the sequence, positional encodings are introduced to provide information about the positions of words in the input sequence.

4. Multi-Head Attention:

The paper introduces multi-head attention, which allows the model to attend to different parts of the input sequence simultaneously. This enables the model to capture diverse patterns and relationships in the data.

5. Feedforward Neural Networks:

The Transformer model includes feedforward neural networks to process information independently across different positions.

6. Encoder-Decoder Architecture:

The Transformer can be used for both encoder and decoder tasks. The encoder processes the input sequence, and the decoder generates the output sequence. This makes it applicable to various sequence-to-sequence tasks, including machine translation.

7. Scaled Dot-Product Attention:

The authors introduce the scaled dot-product attention mechanism, which scales the dot products of the attention mechanism to prevent the gradients from becoming too small during backpropagation.

8. Results and Performance:

The Transformer model achieved state-of-the-art results in machine translation tasks and demonstrated competitive performance on other sequence-to-sequence tasks.

The architecture has become a foundation for subsequent advancements in natural language

processing and is widely used in various applications.

Overall, the Transformer architecture introduced in "Attention is All You Need" has had a significant impact on the field, leading to further research and the development of more advanced models for natural language processing tasks.

GEMINI:

Absolutely! Here's a breakdown of the main findings of "Attention is All You Need" by Vaswani et al., along with why it was a breakthrough in natural language processing (NLP):

Key Findings

- **The Transformer Architecture:** The paper introduced the Transformer, a novel neural network architecture based entirely on self-attention mechanisms. This was a major departure from the then-prevalent use of recurrent neural networks (RNNs) and convolutional neural networks (CNNs) in sequence modeling.
- **Superior Performance:** Transformers demonstrated superior performance in machine translation tasks compared to existing models. They achieved state-of-the-art results on benchmark datasets like WMT 2014 English-German and WMT 2014 English-French.
- **Efficiency and Parallelizability:** The Transformer's architecture is significantly more parallelizable than RNNs, leading to faster training times. This was crucial as datasets grew larger and models became more complex.
- **Expanding the Scope of Attention:** The paper demonstrated that self-attention mechanisms could be applied effectively beyond machine translation. Transformers later proved revolutionary across multiple NLP tasks and are now foundational in fields like computer vision.

Why is This Important?

The impact of "Attention is All You Need" on NLP has been immense. Here's why:

- **The Power of Attention:** The paper showcased the effectiveness of attention mechanisms for capturing long-range dependencies in sequences. Unlike RNNs or

CNNs, Transformers can directly "attend" to relevant parts of the input, regardless of their distance, leading to better understanding of context.

- The Foundation for Large Language Models: The Transformer architecture provided the scalable and efficient backbone for modern large language models (LLMs) like GPT-3 and BERT. These models have enabled a multitude of breakthroughs in natural language generation, understanding, and translation.
- Beyond NLP: Transformers have transcended NLP. They're now applied in domains such as computer vision (vision transformers), protein structure prediction, and more.

Let me know if you'd like a deeper explanation of how the Transformer architecture works or its specific advantages!

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:

Certainly! Here's a basic template for an undergraduate CS student at UCSD seeking jobs in machine learning and data science:

****John Doe****

Address Line 1 • Address Line 2 • City, State, Zip Code • Email Address • Phone Number

****Objective:****

To secure a challenging position in machine learning and data science where I can apply my strong analytical skills, programming proficiency, and academic training in computer science and mathematics.

****Education:****

- ****University of California, San Diego (UCSD)****
- Bachelor of Science in Computer Science, expected June 2024

****Relevant Coursework:****

- Machine Learning
- Data Mining
- Algorithms and Data Structures

- Probability and Statistics

- Linear Algebra

- Database Systems

****Skills:****

- Programming Languages: Python (proficient), Java, C++

- Machine Learning Libraries: Scikit-learn, TensorFlow, Keras, PyTorch

- Data Analysis and Visualization: Pandas, NumPy, Matplotlib, Seaborn

- Database Management: SQL, MongoDB

- Software Development: Git, Agile methodologies

****Projects:****

1. ****Flight Delay Prediction System****

- Developed a predictive model using machine learning algorithms to forecast flight delays based on historical airline data.

- Implemented parallelized processing with Dask to handle large datasets efficiently.

- Achieved XX% accuracy in predicting flight delays, contributing to improved operational planning.

2. ****Sentiment Analysis of Product Reviews****

- Conducted sentiment analysis using natural language processing techniques to categorize customer reviews.

- Applied machine learning algorithms to classify reviews into positive, neutral, and negative sentiments.

- Implemented a web interface for real-time sentiment analysis using Flask and deployed it on AWS.

3. ****Stock Price Prediction Using LSTM****

- Built a long short-term memory (LSTM) neural network model to predict stock prices based on historical data.

- Optimized the model architecture and hyperparameters to improve prediction accuracy.

- Conducted backtesting and analysis of predicted vs. actual stock prices.

****Experience:****

- ****Software Engineering Intern****

- Qualcomm Institute, Summer 2024

- Developed and tested software components for next-generation wireless communication

systems.

- Collaborated with a team of engineers to integrate new features and optimize existing codebase.

****Honors and Awards:****

- Dean's List, UCSD (Multiple Quarters)
- Scholarship Recipient, Department of Computer Science, UCSD

****Technical Proficiencies:****

- Proficient in Linux/Unix environments
- Familiarity with cloud computing platforms (AWS, Google Cloud)
- Strong problem-solving and analytical abilities

****References:****

Available upon request.

This resume template highlights your education, relevant coursework, skills, projects, internship experience, and technical proficiencies tailored for a career in machine learning and data science. Adjust and expand each section as necessary to fit your specific experiences and achievements.

GEMINI:

****[Your Name]****

****[Your Email Address]**** | ****[Your Phone Number (Optional)]**** | ****[Your LinkedIn Profile URL (Optional)]****

****Summary****

Highly motivated and results-oriented undergraduate Computer Science student at UCSD with a strong passion for machine learning and data science. Eager to leverage coursework, projects, and research experience to contribute to innovative solutions in the field. Possesses excellent analytical and problem-solving skills with proficiency in Python and machine learning libraries.

****Education****

* ****University of California, San Diego (UCSD)**** | San Diego, CA | Expected Graduation: May 20****[Year]****

* Bachelor of Science in Computer Science | GPA: ****[Your GPA]**** (if 3.0 or higher)

****Skills****

- * Programming Languages: Python (libraries: NumPy, Pandas, Scikit-learn, TensorFlow/PyTorch (choose one))
- * Machine Learning: Supervised Learning (Regression, Classification), Unsupervised Learning (Clustering)
- * Data Analysis: Data Cleaning, Preprocessing, Feature Engineering, Exploratory Data Analysis (EDA)
- * Deep Learning (if applicable)
- * Version Control: Git
- * ****[Additional Skills: List any other relevant skills, like SQL, cloud platforms (AWS, Azure, GCP)]****

****Projects****

- * ****[Project Title 1]****: Briefly describe your project, including the technologies used, the problem it addressed, and the results achieved. Quantify your impact if possible (e.g., improved accuracy by X%). ****[Year]****
- * ****[Project Title 2]****: Briefly describe your project following the same format as Project Title 1. ****[Year]**** (Add more project descriptions as needed)

****Coursework (Optional)****

- * Machine Learning
- * Data Mining
- * Artificial Intelligence
- * Data Structures & Algorithms
- * Database Systems

****Activities & Awards (Optional)****

- * ****[List any relevant activities or awards that demonstrate your interest and skills in machine learning and data science]****

****[Optional: Include a link to your GitHub profile or portfolio website showcasing your projects]****

****Please note:**** This is a template and you should tailor it to your specific experiences and skills.

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:

Sure! "Je ne parle Français" translates to "I don't speak French" in English.

GEMINI:

The correct phrase is "Je ne parle pas français."

Here's the breakdown of the translation:

Je - I

ne - do not

parle - speak

pas - not

français - French

The whole sentence translates to "I don't speak French."

Exercise 2:

PROMPT:

Can you translate to English: Toi la nguoi Viet.

CHATGPT:

The phrase "Toi la nguoi Viet" translates to "I am Vietnamese" in English.

GEMINI:

"Toi la nguoi Viet" translates to "I am Vietnamese."

Here's the breakdown:

- Toi - I
- la - am (verb "to be")
- nguoi Viet - Vietnamese person

Exercise 3:**PROMPT:**

How do you say this in Japanese: Sorry, I don't speak Japanese

CHATGPT:

To say "Sorry, I don't speak Japanese" in Japanese, you can use:

すみません、日本語が話せません。

Here's a breakdown of the sentence:

すみません (Sumimasen) - Excuse me / Sorry

日本語 (Nihongo) - Japanese (language)

が (ga) - Subject marker

話せません (hanasemasen) - Cannot speak

So, the full phrase reads: すみません、日本語が話せません。 (Sumimasen, Nihongo ga hanasemasen.)

For a slightly more casual version, you can say:

PROMPT:

How do you say this in Japanese: Sorry, I don't speak Japanese

ごめんなさい、日本語が話せないんです。

ごめんなさい (Gomennasai) - I'm sorry

話せないんです (hanasenai n desu) - Can't speak

This version is a bit more informal and adds a softer nuance with the phrase "なんです" (n desu), which is often used to explain or give reasons.

Both are polite and will be understood in most situations.

GEMINI:

The English translation of "Toi la nguoi There are two common ways to say "Sorry, I don't speak Japanese" in Japanese, depending on how much politeness you want to convey:

More polite:

すみません、日本語がわかりません。(Sumimasen, nihongo ga wakarimasen.)

This translates to "Excuse me, I don't understand Japanese."

Less polite (but still appropriate in casual situations):

日本語わかりません。(Nihongo wakarimasen.)

This simply means "I don't understand Japanese."

Here are some additional things to keep in mind:

You can bow slightly when saying either phrase to show respect.

If you want to add that you are learning Japanese, you can say:

日本語を勉強していますけど、ちょっとしかわかりません。(Nihongo o benkyo shiteいますけ

PROMPT:

How do you say this in Japanese: Sorry, I don't speak Japanese

ど、chotto shika wakarimasen.)

This translates to "I am studying Japanese, but I don't understand much."

You can also try asking the person to speak slowly or rephrase their question:

ゆっくり話していただけますか。(Yukkuri hanashite itadakemassen ka?)

This translates to "Could you please speak slowly?"

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:

This means that something (likely referring to an outfit or appearance) initially caused a strong reaction (being triggered and shook), but upon closer examination, it was found to be very fashionable, impressive, and genuinely good (popping and legit), to the point of being extremely enjoyable or exciting (totally lit).

GEMINI:

This sentence uses a mix of slang and informal language to describe someone's outfit in a very positive way. Here's the breakdown:

* **Hella fit:** "Hella" is a slang term from Northern California, mostly meaning "very" or "extremely." So "hella fit" translates to "very cool outfit" or "extremely stylish outfit."

* **Triggered and shook:** These terms are often used jokingly to describe feeling surprised or overwhelmed, possibly even a bit confused at first.

* **Popping and legit:** "Popping" means something is stylish or impressive, while "legit"

means it's genuine and the coolness isn't fake.

* **Totally lit:** This is a slang way to say something is extremely exciting or enjoyable.

Put it all together, and the sentence means:

> "This is an incredibly cool outfit! At first I wasn't sure what to think, it was a bit surprising (triggered and shook), but when I looked closer I realized it's super stylish and authentic (popping and legit). In fact, it's awesome (totally lit)!"

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:

```
import pandas as pd
# Read the dataframe from CSV file
df = pd.read_csv('your_data.csv')
# Remove rows with missing values
df = df.dropna()
# Group by 'Dept' and count occurrences
dept_counts = df.groupby('Dept').size().reset_index(name='Count')
# Convert the grouped data to JSON and write to file
dept_counts.to_json('dept_counts.json', orient='records')
# Print the dataframe (optional)
print(dept_counts)
```

GEMINI:

```
import pandas as pd
import json

# Read the data from a CSV file (replace 'your_data.csv' with your
```

```
actual filename)
df = pd.read_csv('your_data.csv')

# Drop rows with missing values in 'Name' or 'Dept' columns
df = df.dropna(subset=['Name', 'Dept'])

# Group the DataFrame by 'Dept'
grouped_data = df.groupby('Dept')

# Prepare a dictionary to store department data
department_data = {}
for name, group_df in grouped_data:
    # Convert the group DataFrame to a dictionary (consider using
    # 'to_dict(orient='records')' for list of dictionaries)
    department_data[name] = group_df.to_dict(orient='index')

# Write the dictionary to a JSON file
with open('department_data.json', 'w') as outfile:
    json.dump(department_data, outfile)
```

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

Can you provide the equivalent PySpark code?

CHATGPT:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count
from pyspark.sql.types import StringType
```

```
# Initialize Spark session
spark = SparkSession.builder \
    .appName("DeptCounts") \
    .getOrCreate()

# Read the dataframe from CSV file
df = spark.read.csv('your_data.csv', header=True, inferSchema=True)

# Drop rows with missing values
df = df.na.drop()

# Group by 'Dept' and count occurrences
dept_counts = df.groupBy('Dept') \
    .agg(count('*').alias('Count')) \
    .orderBy('Dept')

# Convert 'Dept' column to StringType for consistent output
dept_counts = dept_counts.withColumn('Dept',
col('Dept').cast(StringType()))

# Write to JSON file
dept_counts.write.json('dept_counts.json', mode='overwrite')

# Stop Spark session
spark.stop()
```

GEMINI:

```
from pyspark.sql import SparkSession
```



```
# Create a SparkSession
spark = SparkSession.builder.appName("DeptData").getOrCreate()

# Read data from a file (replace 'path/to/your/data.csv' with your
actual path)
df = spark.read.csv("path/to/your/data.csv", header=True)

# Filter out rows with missing values in 'Name' or 'Dept' columns
df_filtered = df.filter(df.col("Name").isNotNull() &
df.col("Dept").isNotNull())

# Group the DataFrame by 'Dept'
grouped_data = df_filtered.groupBy("Dept")

# Collect results as dictionaries (consider using Spark SQL functions
for complex transformations)
department_data = grouped_data.collectAsMap()

# Write the dictionary to a JSON file
with open("department_data.json", "w") as outfile:
    json.dump(department_data, outfile)

print("Data written to department_data.json")

# Stop the SparkSession
spark.stop()
```

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:

```
import pandas as pd

# Load Titanic dataset
df = pd.read_csv('./titanic.csv')

# Define function to remove outliers based on 1st and 99th
percentiles
def remove_outliers(df, column):
    lower_bound = df[column].quantile(0.01)
    upper_bound = df[column].quantile(0.99)
    return df[(df[column] >= lower_bound) & (df[column] <=
upper_bound)]

# Count outliers before handling
outliers_before = df.shape[0]

# Remove outliers for 'Age'
df = remove_outliers(df, 'Age')

# Remove outliers for 'Fare'
df = remove_outliers(df, 'Fare')

# Count outliers after handling
```

```
outliers_after = df.shape[0]

# Report number of outliers before and after handling
print(f"Number of outliers before handling: {outliers_before}")
print(f"Number of outliers after handling: {outliers_after}")
```

GEMINI:

```
import pandas as pd

# Read the data from the CSV file
df = pd.read_csv("titanic.csv")

# Define percentile thresholds
lower_thresh = 0.01
upper_thresh = 0.99

# Count initial number of rows
initial_count = len(df)

# Find the quartiles for 'Age' and 'Fare'
q1_age = df['Age'].quantile(lower_thresh)
q3_age = df['Age'].quantile(upper_thresh)

q1_fare = df['Fare'].quantile(lower_thresh)
q3_fare = df['Fare'].quantile(upper_thresh)

# Filter outliers based on percentiles
filtered_df = df[ (df['Age'] >= q1_age) & (df['Age'] <= q3_age) &
(df['Fare'] >= q1_fare) & (df['Fare'] <= q3_fare)]
```

```
# Count outliers removed
outlier_count = initial_count - len(filtered_df)

# Print results
print(f"Initial number of rows: {initial_count}")
print(f"Number of outliers removed: {outlier_count}")

# Optional: Save the filtered DataFrame
# filtered_df.to_csv("titanic_filtered.csv", index=False)
```

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.

Exercise¹:

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:

Certainly! The code `find_max` function iterates through a list to find the maximum value:

1. **Initialization**:

- `max_val = list[0]`: Initializes `max_val` with the first element of the list. This assumes the list is non-empty.

2. **Nested Loops**:

- `for i in range(len(list))`: Outer loop iterates through each index `i` of the list.

¹ The code in the exercise prompt is from Konfuzio: <https://konfuzio.com/en/python-tutorial-complexity/>

- `for j in range(i, len(list)):`: Inner loop starts from index `i` and iterates to the end of the list.

3. **Comparison**:

- `if list[j] > max_val:`: Checks if the current element `list[j]` is greater than `max_val`.
- If true, updates `max_val` to `list[j]`.

4. **Return**:

- `return max_val`: Once both loops complete, `max_val` holds the maximum value found in the list.

Explanation:

- The function uses nested loops to compare each element (`list[j]`) with the current maximum value (`max_val`).
- It iteratively updates `max_val` whenever a larger element is found, ensuring that by the end of the loops, `max_val` contains the largest value in the list.
- This approach has a time complexity of $O(n^2)$, where n is the length of the list, due to the nested loops. It is straightforward but not the most efficient way to find the maximum value in a list, especially for larger lists.

GEMINI:

This code defines a function called `find_max` that takes a list as input and returns the maximum value in the list. Here's a breakdown of how it works:

1. **Function definition:** `def find_max(list):` defines a function named `find_max` that takes a single argument named `list`.
2. **Initialize max_val:** `max_val = list[0]` sets a variable `max_val` and assigns the first

element of the input list to it. This assumes the list is not empty and has at least one element.

3. **Nested loop:** The code uses two nested loops to iterate through the list:

* **Outer loop:** `for i in range(len(list)):` iterates through the indices of the list using `range(len(list))`. This loop runs for the entire length of the list.

* **Inner loop:** `for j in range(i, len(list)):` iterates through the elements from the current index (`i`) to the end of the list. This inner loop checks each element after the current index (`i`).

4. **Compare and update:** Inside the inner loop, it compares the current element (`list[j]`) with `max_val`. If the element is greater than `max_val`, it updates `max_val` with the new larger value.

5. **Return max value:** After iterating through the entire list, the function returns the final value stored in `max_val`, which will be the maximum value found in the list.

Note: This approach uses a nested loop which can be less efficient for large datasets. The built-in `max` function in Python offers a more concise and efficient way to find the maximum value in a list.