from Pexels (Karolina)

Reduce Your OpenAI API Costs by 70%

Prompt Engineering Is All You Need

Fareed Khan · Follow
Published in Level Up Coding · 11 min read · Mar 19, 2024

While many open-source LLMs have popped up since ChatGPT came out, none have quite matched the satisfaction and reliability of OpenAI. If you

want to use LLMs in your app or build one yourself, OpenAI is still the way to go. They have a great business model for their language model APIs, and their most cheapest and powerful option, **gpt-3.5-turbo-0125**, is widely used. But you can actually make it even cheaper by being smart about how you call the API. The secret is in **prompt engineering**, knowing what to put in and what you want to get out.

Let's take a quick look at where things stand today, in March 2024. For this blog, we'll be using the gpt-3.5-turbo-0125 language model. You can use others, but this one is the most affordable for me, especially since we'll be dealing with a lot of text data. Let's check out its specs, like pricing, token limits and more.

MODEL	CONTEXT WINDOW	TRAINING DATA	Input	Output
gpt-3.5-turbo-0125	16,385 tokens	Up to Sep 2021	$0.50 / 1M tokens	$1.50 / 1M tokens

openAI.md hosted with ❤️ by GitHub view raw

gpt-3.5-turbo-0125 pricing (March 2024)

Before we get started, let's see how OpenAI calculates token each time an API call made.

```
# Import the OpenAI class from the openai module
from openai import OpenAI

# Create an instance of the OpenAI class with the provided API key
client = OpenAI(api_key=open_ai_key)

# Set the user input to the string "Hi"
user_input = "Hi"

# Create a chat completion using the OpenAI client and the user input
response = client.chat.completions.create(
    model="gpt-3.5-turbo-0125",  # Set the model to "gpt-3.5-turbo-0125"
    messages=[{"role": "user", "content": user_input}]
)
```

```
# Print the response from the chat completion
print(response.choices[0].message.content)

######## OUTPUT ########
Hello! How can I help you today?
######## OUTPUT ########
```

You might think that if you send "**Hi**" as a single token input, the response would be only 7–9 tokens. But that's not actually the case. When we print the usage details of the response, here's what we see:

```
# Print the usage of the response
print(response.usage)

######## OUTPUT ########
CompletionUsage(completion_tokens=9, prompt_tokens=8, total_tokens=17)
######## OUTPUT ########
```

Notice that **prompt_tokens**, which refers to the input tokens, is 8. This is because every time you send text to the API, an additional 7 tokens are automatically added. The **completion_tokens** value is what we expected, representing the tokens in the response generated by the model.

Based on what we've learned, let's code a function that takes a user prompt as input and returns four things: the API response, the cost of the input tokens, the cost of the output tokens, and the total cost.

```
def openai_chat(user_prompt):

    # Create a chat completion using the OpenAI client
    response = client.chat.completions.create(
        # Set the model to "gpt-3.5-turbo-0125"
        model="gpt-3.5-turbo-0125",
        messages=[{"role": "user", "content": user_prompt}]
    )

    # Response from the chat completion
    answer = response.choices[0].message.content
```

```
    # Calculate the price for input tokens
    input_price = response.usage.prompt_tokens * (0.5 / 1e6)

    # Calculate the price for input tokens
    output_price = response.usage.completion_tokens * (1.5 / 1e6)

    # calculate the total price
    total_price = input_price + output_price

    # Return the answer, input price, output price, total price
    return {
        "answer": answer,
        "input_price": f"$ {input_price}",
        "output_price": f"$ {output_price}",
        "total_price": f"$ {total_price}"
    }
```

Let's see how this function works by trying it out with a simple, short prompt.

```
# Calling our function with the prompt
openai_chat("What is the capital of China?")


####### OUTPUT #######
{
  answer: 'The capital of China is Beijing.',
  input_price: '$ 7e-06',
  output_price: '$ 1.05e-05',
  total_price: '$ 1.75e-05'
}
####### OUTPUT #######
```

The output of our function should be clear. Just to clarify that the price is calculated based on the pricing table I showed you earlier for the **gpt-3.5-turbo-0215 model.**

```
# asking a question regarding 100k words document
openai_chat(100K_words_document_and_one_question)

####### OUTPUT #######
```

```
{
  answer: 'According to the give context .... ',
  input_price: '$ 0.18',
  output_price: '$ 0.02',
  total_price: '$ 0.20'
}
####### OUTPUT #######
```

For Large amount of text (100k words) and a question, the estimated cost is only about $0.20. This shows how efficient the gpt-3.5-turbo model can be. Now that we have our function ready to go, let's see how we can minimize the cost of using the OpenAI API.

Clustering using OpenAI API

Imagine you have a huge list of news headlines and you want to cluster them. While using embeddings is one option, let's say you want to use OpenAI language models API, as it can capture meaning in a human-like way. Before we create the prompt template to minimize costs, let's take a look at what our news headlines list looks like.

```
# news heading list
news_headlines = [
    "donald trump is ... surprise when he arrives",
    "Emirates airlines ... takeover of Etihad Airways",
    "The US is considering  ... troops to Afghanistan",
     ...
]

# length of news list
len(news_headlines)


####### OUTPUT #######
1256
####### OUTPUT #######
```

We have over 1200 news headlines. Before we create the prompt template, let's merge them into a single string with a specific format. This will help the language model process them more efficiently.

```
formatted_news = "Given the news headlines:\n"

# looping through all news headlines and formatting them in single string
for i in range(0,len(user_input)):
    formatted_string += f"s{i}: {user_input[i]}\n"

# printing the formatted news
print(formatted_news)


####### OUTPUT #######
s0: donald trump is ... surprise when he arrives
s1: Emirates airlines ... takeover of Etihad Airways
s2: The US is considering  ... troops to Afghanistan
...
s1256: Virat Kholi is ready ... against Srilanka
####### OUTPUT #######
```

I've represented the news headlines in a shorter form, like s0, s1, s2, and so on. The reason for this is that when the language model clusters them, it can simply use these short abbreviations (e.g., s35 for the 35th news headline) instead of writing out the entire headline in each cluster.

```
prompt_template = f'''{formatted_news}
i have these news headlines i want you to perform clustering on it.
you have to answer me in this format:
cluster1: s1,s3 ...
cluster2: s2, s6 ...
...
do not write sentences but write it just like this starting from s0,s1,s2 ...
dont say anything else other than the format
'''
```

Next, I defined my prompt template. This template specifies the format of the answer I want the language model to provide, along with some additional information for clarity. The key here is that we're not asking the model to write out the full headlines, but rather to just use the short abbreviations.

All we need to do is pass this prompt template to the function we created earlier and see how it performs in terms of pricing and response quality.

```
# Calling our function with prompt_template
results = openai_chat(prompt_template)

# printing response only
print(results['answer'])


####### OUTPUT #######
cluster1: s5, s67, s55, s134, s764 ...
cluster2: s64, s21, s896, s445, s12 ...
...
cluster7: s12, s853, s414, s244, s712 ...
####### OUTPUT #######
```

I've truncated the response here, but the language model successfully classified our 1256 news headlines into **seven clusters**. You can further edit the prompt template to specify the desired number of clusters. Let's see how much this task cost us.

```
# printing complete results
print(results)

####### OUTPUT #######
{
  answer: 'cluster1: s5, s67, ...',
  input_price: '$ 0.020',
  output_price: '$ 0.003',
  total_price: '$ 0.023'
}
####### OUTPUT #######
```

The total cost for this task comes to **$0.023**. If we hadn't used this prompt template, we would likely have spent more than twice as much as input costs. This is because we would be sending all the headlines as input, and the output cost of $1.50 per million tokens would apply to the complete

headlines being returned. Here's a summary of how much we saved by using this approach:

Method	Clustering
Actual Cost	$ 0.047
Our Approach Cost	$ 0.023
Cost Reduction	51.56%

openAI_clustering.md hosted with ❤️ by GitHub view raw

This approach is going to be crucial as it save your credits exponentially when you work with bigger data. But we have to code a little bit to map the short abbreviations to actual headlines.

```
# Split the 'answer' string by newline character
responses = result['answer'].split("\n")

# Create a dictionary comprehension to extract key-value pairs from each line
clusters = {line.split(": ")[0].strip(): [i.strip() for i in line.split(": ")[1]

# Replace the keys (s01, s02, s03, etc.) with the actual headlines from 'news_he
for k, v in clusters.items():
    clusters[k] = [news_headlines[int(i[1:])] for i in v]
```

The code above will map the OpenAI response back to the actual news headlines Here's what the **clusters** dictionary looks like:

```
# print clusters
print(clusters)

####### OUTPUT #######
{
  'cluster1': ["Sea levels  ... change", "Arabian sea .. pollution", ...],
  'cluster2': ["fifa has banned ... ", "Ronaldo has been ... a row", ...}
  ...
```

```
}
####### OUTPUT #######
```

SpellCheck using OpenAI API

Let's say you have a lengthy text document and you want to build a grammar correction tool as a small web app. While there are many NLP techniques available for this task, language models, particularly those from OpenAI, have been trained on vast amounts of data, making them a potentially better choice. Again, the key is to be strategic with our prompt template. We want the API response to highlight incorrect words and suggest their correct spellings, rather than providing the entire corrected text as output. Let's take a look at our input text:

```
# my input which contains spelling errors
user_input = '''As the sun beganned to set on the horizen, casting a
                warm gloy across the ... '''

# printing total words
print(len(input_text.split()))


####### OUTPUT #######
500,000
####### OUTPUT #######
```

We have a 500k-word document with spelling errors that need correction. Our next step is to create a prompt template that guides the API to provide responses focused solely on the problematic words.

```
prompt_template = f'''Given the input text:
user input: {user_input}

output must be in this format:
misspelled_word:corrected_word
...
```

```
output must not contain any other information than the format
'''
```

Within the prompt template, we've explicitly defined the desired response format. The API should highlight misspelled words and provide their correct replacements. We've also specified that no additional information should be included in the response, as we'll be using code to replace these words in the original **user_input**.

All we need to do is pass this prompt template to the function we created earlier and see how it performs in terms of pricing and response quality.

```
# Calling our function with prompt_template
results = openai_chat(prompt_template)

# printing response only
print(results['answer'])

####### OUTPUT #######
beganned: began
horizen: horizon
gras: grass
...
####### OUTPUT #######
```

I've truncated the response here, but the language model successfully identified misspelled word and provided their correct form. Let's see how much this task cost us.

```
# printing complete results
print(results)

####### OUTPUT #######
{
  answer: 'beganned: began \n horizen: horizon ...',
  input_price: '$ 0.33333',
  output_price: '$ 0.005135',
  total_price: '$ 0.33812'
```

```
}
####### OUTPUT #######
```

The total cost for this task comes to **$0.33812,** if 500K words contains 8K-10K spell errors. If we hadn't used this prompt template, we would likely have

how much we saved by using this approach:

Method	Spell Check
Actual Cost	1.332
Our Approach Cost	0.338
Cost Reduction	74.61%

spellcheck.md hosted with ❤️ by GitHub view raw

Even with a relatively small text of 500 words, the cost savings are noticeable. Imagine the financial benefits when working with big data!

But we have to code a little bit to map the short abbreviations to actual headlines.

```
# Split the 'response1' string by newline character
response = result['answer'].split("\n")

# Create a list comprehension to extract key-value pairs from each line
result = [(line.split(":")[0], line.split(":")[1]) for line in response if line.

# Replace the misspelled words with the corrected words
for word, corrected_word in result:
    corrected_user_input = user_input.replace(word, corrected_word)
```

The code above will map the misspelled words with their correct versions:

```
# print clusters
print(corrected_user_input)

####### OUTPUT #######
As the sun began to set on the horizon, casting
warm glow across the ...
####### OUTPUT #######
```

Text Cleaning using OpenAI API

The prompt template we used for spellcheck same will be used for text cleaning as we only need to highlight those words that either want to get cleaned or removed completely.

We want the API response to highlight incorrect words and suggest their correct spellings, rather than providing the entire corrected text as output. Let's take a look at our input text:

```
# my input which contains spelling errors
user_input = '''The rugge9d pathw&ay winded through the th!ck f0r3st,
obfusca+ting the way forward. The creaking of branches
and rustling of leaves added an ... '''

# printing total words
print(len(input_text.split()))

####### OUTPUT #######
1,000,000
####### OUTPUT #######
```

We have a 1M-word document which requires cleaning. Our next step is to create a prompt template that guides the API to provide responses focused solely on the problematic words.

```
prompt_template = f'''Given the input text:
user input: {user_input}
output must be in this format:
uncleaned_word:cleaned_word
...
if no replacement for uncleaned_word exist then use this format
uncleaned_word:
output must not contain any other information than the format
'''
```

Within the prompt template, we've explicitly defined the desired response format. The API should highlight uncleaned words and provide their correct replacements and if uncleaned words have no replacement that it should have to be replaced by empty string. We've also specified that no additional information should be included in the response, as we'll be using code to replace these words in the original **user_input**.

All we need to do is pass this prompt template to the function we created earlier and see how it performs in terms of pricing and response quality.

```
# Calling our function with prompt_template
results = openai_chat(prompt_template)

# printing response only
print(results['answer'])

####### OUTPUT #######
rugge9d: rugged
th!ck: thick
f0r3st: forest
obfusca+ting: obfuscating
s;ense: sense
...
####### OUTPUT #######
```

I've truncated the response here, but the language model successfully identified uncleaned word and provided their correct form. Let's see how much this task cost us.

```
# printing complete results
print(results)

####### OUTPUT #######
{
  answer: 'rugge9d: rugged \n th!ck: thick ...',
  input_price: '$ 0.6665',
  output_price: '$ 0.099',
  total_price: '$ 0.7665'
}
####### OUTPUT #######
```

The total cost for this task comes to **$0.7665,** if 1 Milliob words contains 20K-30K cleaning errors. If we hadn't used this prompt template, we would likely have spent more than twice as much as input costs. Here's a summary of how much we saved by using this approach:

Working with 1 Million words document, the cost savings are noticeable. Imagine the financial benefits when working with big data!

But we have to code a little bit to map the short abbreviations to actual headlines.

```
# Split the 'response1' string by newline character
response = result['answer'].split("\n")

# Create a list comprehension to extract key-value pairs from each line
result = [(line.split(": ")[0], line.split(": ")[1]) for line in response if lin
# Replace the misspelled words with the corrected words
```

```
for word, corrected_word in result:
    cleaned_user_input = user_input.replace(word, corrected_word)
```

The code above will map the misspelled words with their correct versions:

```
# print clusters
print(cleaned_user_input)

####### OUTPUT #######
The rugged pathway winded through the th!ck forest,
obfuscating the way forward. The creaking of branches
and rustling of leaves added an ...
####### OUTPUT #######
```

LLM Based NLP

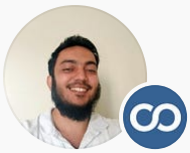
I've developed an NLP library that uses LLM APIs to perform various tasks. It includes over 30 features, many of which works with similar cost-optimization strategies as described above. You can explore the library and its prompt templates in my GitHub repository:

GitHub - FareedKhan-dev/basiclingua-LLM-Based-NLP: LLM Based NLP Library.

LLM Based NLP Library. Contribute to FareedKhan-dev/basiclingua-LLM-Based-NLP development by creating an...

github.com

OpenAI API Data Science ChatGPT Machine Learning



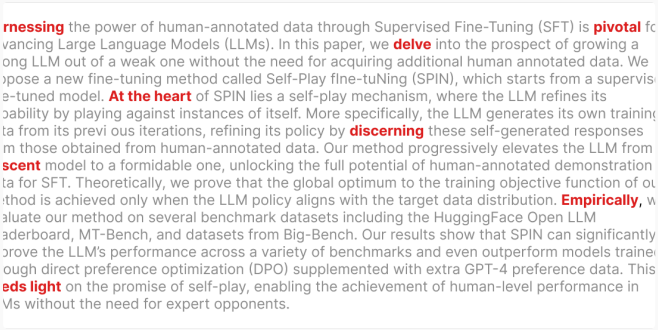
Written by Fareed Khan

Follow

21K Followers · Writer for Level Up Coding

MSc Data Science, I write on AI https://www.linkedin.com/in/fareed-khan-dev/

More from Fareed Khan and Level Up Coding



Fareed Khan in Level Up Coding

Detect AI Text by Just Looking at it

Words that LLM regularly uses

4 min read · Apr 18, 2024

862 31


Somnath Singh in Level Up Coding

The Era of High-Paying Tech Jobs is Over

The Death of Tech Jobs.

· 14 min read · Apr 1, 2024

10K 258

Alexander Nguyen in Level Up Coding

Why I Keep Failing Candidates During Google Interviews...


Fareed Khan in Level Up Coding

Solving Transformer by Hand: A Step-by-Step Math Example

They don't meet the bar.

4 min read · Apr 13, 2023

5.6K 165

Performing numerous matrix multiplications to solve the encoder and decoder parts of th...

13 min read · Dec 18, 2023

2.3K 34

See all from Fareed Khan

See all from Level Up Coding

Recommended from Medium

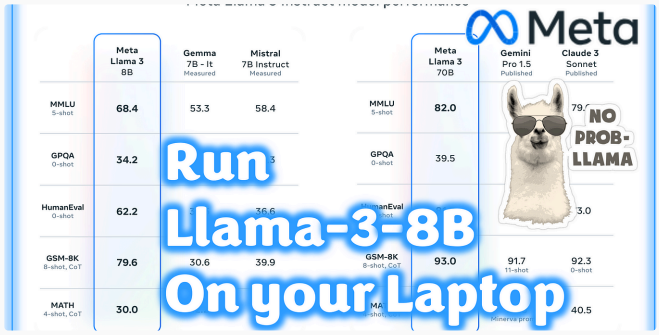
Vatsal Saglani in Towards AI

Llama 3 + Groq is the AI Heaven

Llama 3 shines on Groq with blazing generation

9 min read · Apr 21, 2024

968 7



Fabio Matricardi in Generative AI

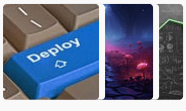
Llama3 is out and you can run it on your Computer!

After only 1 day from the release, here is how you can run even on your Laptop with CPU...

8 min read · Apr 20, 2024

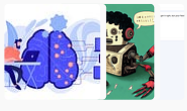
1.7K 20

Lists



Predictive Modeling w/ Python

20 stories · 1142 saves



The New Chatbots: ChatGPT, Bard, and Beyond

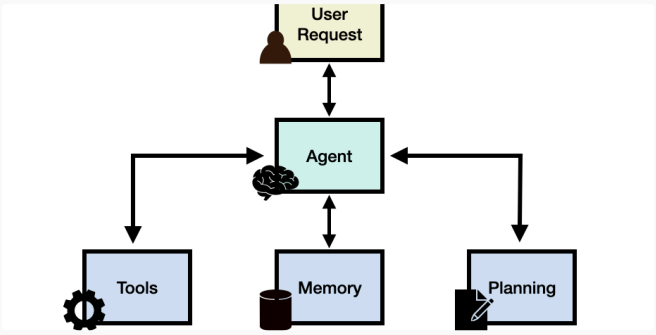
12 stories · 363 saves

ChatGPT prompts

47 stories · 1497 saves

Practical Guides to Machine Learning

10 stories · 1374 saves



Vishal Rajput in AIGuys

AI Agents Are All You Need

AI Agents, Understanding the role of Tools, Memory, and Planning in making them work.

14 min read · Apr 22, 2024

393 1

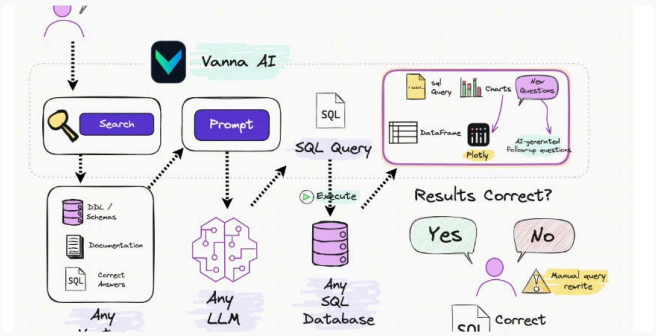
Martin Thissen

Llama 3 on Your Local Computer | Free GPT-4 Alternative

In this article I will point out the key features of the Llama 3 model and show you how you ca...

9 min read · Apr 19, 2024

571 9



Angelina Yang

No More Text2SQL, It's Now RAG2SQL!

Text2SQL was awfully popular last year. I talked with a vendor looking to buy, and also...

3 min read · Apr 12, 2024

129 3

Kenneth Leung in Level Up Coding

Inside the Leaked System Prompts of GPT-4, Gemini 1.5, Claude 3, an...

Unveiling the shared strategies and key distinctions in the prompts behind the best...

7 min read · Apr 16, 2024

701 6

See more recommendations