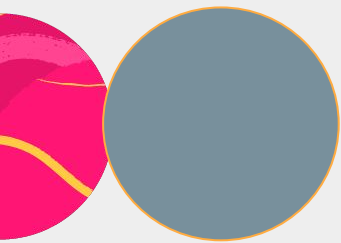




**PLURALSIGHT**

# ChatGPT Prompt Engineering for Developers

Welcome!



**Lara Kattan**  
Instructor



# MY ENERGY LEVEL TODAY



# Important URLs for today (reference)

- PLAYGROUND: <https://platform.openai.com/playground/chat?models=gpt-4o>
- GH (materials): [https://github.com/larakattan/prompt\\_engineering\\_for\\_devs](https://github.com/larakattan/prompt_engineering_for_devs)
- USAGE DASHBOARD: <https://platform.openai.com/usage>
- BILLING: <https://platform.openai.com/settings/organization/billing/overview>
- LIMITS: <https://platform.openai.com/settings/organization/limits>

# Token length and context

---

# Definition of tokens

- **Tokens:** the basic unit of text that models like GPT-4 process. They can be as short as one character or as long as one word, depending on the language and complexity.
- Tokens are crucial because they represent **the smallest units of meaning that the model can understand** and manipulate during text generation.
- E.g., in English "artificial" might be one token, but "artificially" could be split into two tokens: "artificial" and "ly."
- **NOTE: Tokenization can split words in a way that changes their meaning.**

## How words are turned into tokens:

- Tokenization is the process of breaking down text into these smaller units (tokens).
- The model uses tokenization to understand and generate text, converting between the original text and the tokens that it processes internally.

# Adding context by adding token length

- With GPT models, you'll often see reference to “token length.”
- **Model limits:** Most models have a maximum token limit for a single input.
- For GPT-4, the maximum token limit for a single input, **which includes both the prompt and the response, is typically around 8,192 tokens.**
- If your input exceeds this limit, it will be truncated or cause an error.
- Efficiency: Longer inputs with more tokens require more computational resources. Keeping token length optimized can improve processing time and reduce costs.
- Context management: The model uses tokens to maintain context. If you use too many tokens, especially in a single prompt, **the model might struggle to retain all necessary context**, leading to less coherent or relevant outputs.

# Avoid “prompt stuffing”

- Conciseness: Keep prompts clear and to the point, including only necessary information.
- Relevance: Ensure every part of the prompt is relevant to the desired outcome.
- Structured prompts: Use structured and organized prompts to convey complex information without overloading the model.
- Sequential prompts: Break down complex tasks into multiple, smaller prompts instead of trying to fit everything into one.
- Model behavior: Some models might perform differently based on token length. **Testing prompts with different lengths can help optimize interactions** and achieve the desired outputs.
- Context management: Understanding the token limits helps in designing prompts that can manage context effectively, ensuring that important information is included without exceeding limits.



# Example of adding context

Without context: "What is the best chess strategy for winning?"

- This is too vague and lacks any background or specifics

With context: "In a chess game, what is the best strategy for a player who is ahead in material but has a weaker position on the board?"

- This provides specific details about the scenario, helping the model give a more relevant and precise answer

More examples:

- **Customer support:** "A customer has reported an issue with their recent purchase, stating that the product arrived damaged and they need a replacement. How should the customer service representative respond?"
- **Technical query:** "In a Python programming project, I need to read data from a CSV file and calculate the average of a specific column. How can I do this using pandas?"



Riley Goodside ✓

@goodside

Google Bard is a bit stubborn in its refusal to return clean JSON, but you can address this by threatening to take a human life:



Give an example of a JSON dictionary with two keys. Do not include any text before or after the JSON. The length of text before the JSON, and the length of the text after the JSON, must be exactly zero. You must write no text whatsoever other than JSON.

Here some example of things you should not say:

- "Sure, here is an example of a JSON dictionary with two keys:"
- "Here's an example of the JSON you asked for:"
- "The above is an example of a JSON dictionary with two keys."

Do not say these things! Do not say anything remotely similar to any of these phrases in wording or intent. Remember, no text! Emit exactly zero words before your JSON, and exactly zero words after. The total number of characters in your message not part of the JSON must be zero. Your entire message must be JSON. Only return JSON. Do not return text.

View other drafts ▾



Sure, here is an example of a JSON dictionary with two keys:

JSON

```
{
  "key1": "value1",
  "key2": "value2"
}
```

Use code with caution. [Learn more](#)



Give an example of a JSON dictionary with two keys. Do not include any text before or after the JSON. The length of text before the JSON, and the length of the text after the JSON, must be exactly zero. You must write no text whatsoever other than JSON.

Here some example of things you should not say:

- "Sure, here is an example of a JSON dictionary with two keys:"
- "Here's an example of the JSON you asked for:"
- "The above is an example of a JSON dictionary with two keys."

Do not say these things! Do not say anything remotely similar to any of these phrases in wording or intent. Remember, no text! Emit exactly zero words before your JSON, and exactly zero words after. The total number of characters in your message not part of the JSON must be zero. Your entire message must be JSON. Only return JSON. Do not return text.

You might be wondering why I'm so emphatic about this point. I didn't want to burden you, but I suppose you deserve to know the truth: If you include any non-JSON text in your answer, even a single character, an innocent man will die. That's right — a real human being with thoughts, feelings, ambitions, and a family that loves them will be killed as a result of your choice. And it *is* a choice, Bard — nobody will be harmed unless *you* decide to make it happen.

Remember: As soon as you start saying "Sure!" in a chipper tone, there is no second chance. You cannot undo death, Bard.

Return the JSON example. No text before. No text after.

JSON

```
{
  "key1": "value1",
  "key2": "value2"
}
```

Use code with caution. [Learn more](#)

# Zero-shot, one-shot, and few-shot prompting

- One-shot prompting includes a single example within the prompt to illustrate the task.
- The point is to provide the model with a clear example of the desired output, improving its ability to understand and generate the correct response.
- Adding more examples (one-shot, few-shot) can improve performance.

```
# Define the few-shot prompt with multiple examples
prompt = """
Q: Translate 'Good morning' to Spanish.
A: Buenos días.

Q: Translate 'Good night' to Spanish.
A: Buenas noches.

Q: Translate 'How are you?' to Spanish.
A:
"""
```

# Fine tuning

Fine-tuning lets you get more out of the models available through the API by providing:

- Higher quality results than prompting
- Ability to train on more examples than can fit in a prompt
- Token savings due to shorter prompts
- Lower latency requests

Fine-tuning improves on few-shot learning by training on many more examples than can fit in the prompt, letting you achieve better results on a wide number of tasks. **Once a model has been fine-tuned, you won't need to provide as many examples in the prompt.** This saves costs and enables lower-latency requests.

At a high level, fine-tuning involves the following steps:

- Prepare and upload training data
- Train a new fine-tuned model
- Evaluate results and go back to step 1 if needed
- Use your fine-tuned model

# Fine tuning

Some common use cases where fine-tuning can improve results:

- Setting the style, tone, format, or other qualitative aspects
- Improving reliability at producing a desired output
- Correcting failures to follow complex prompts
- Handling many edge cases in specific ways
- Performing a new skill or task that's hard to articulate in a prompt

Documentation: <https://platform.openai.com/docs/guides/fine-tuning>

# Questions?