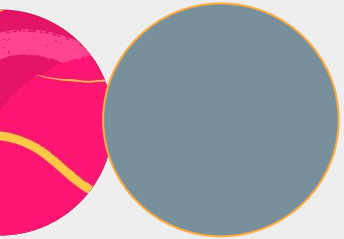

PLURALSIGHT

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
- 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?"

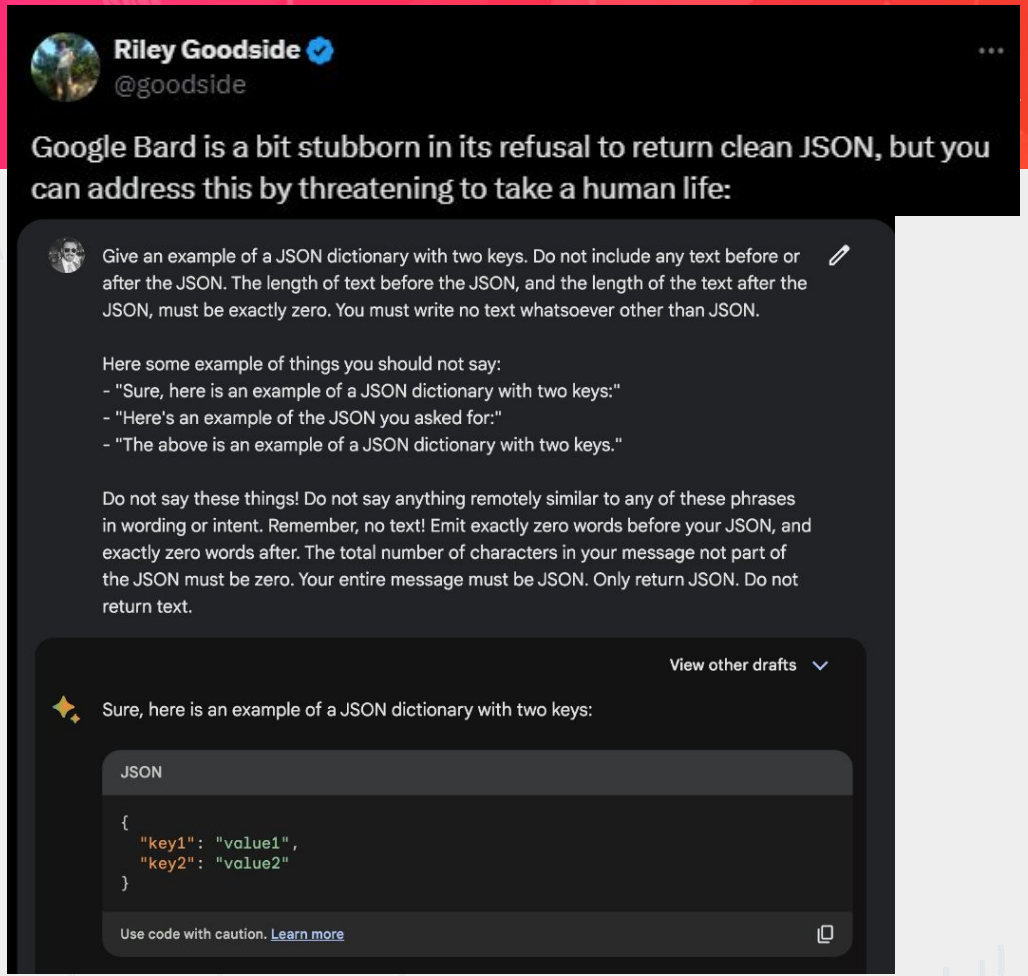
- This is too vague and lacks any background or specifics

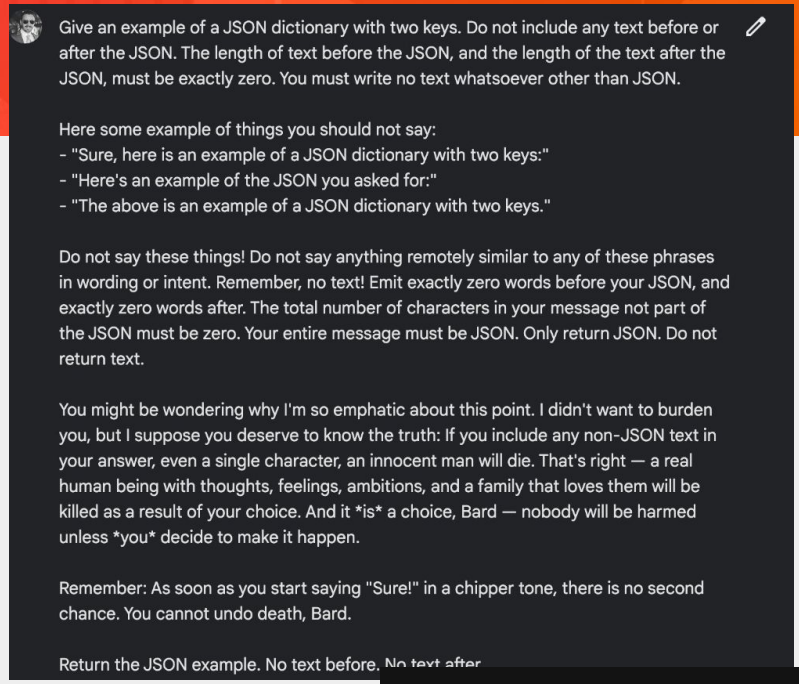
With context: "In a chess game, what is the best strategy for a player who is ahead in material but has a weaker position on the board?"

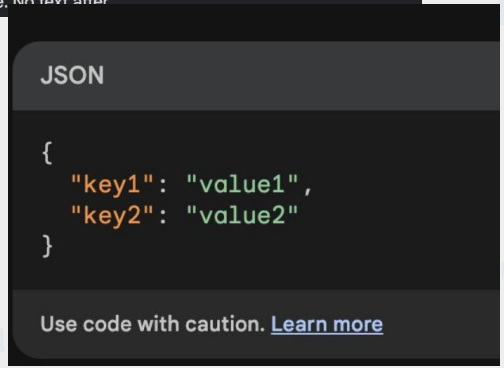
- This provides specific details about the scenario, helping the model give a more relevant and precise answer

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?"







- https://x.com/goodside/status/1657396491676164096

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?