# Building Autonomous Agent - 2

## Week 8-9

# Last Week

Week 7: Build Autonomous Agent - 1

N-Shot Learning, LangChain

# LangChain

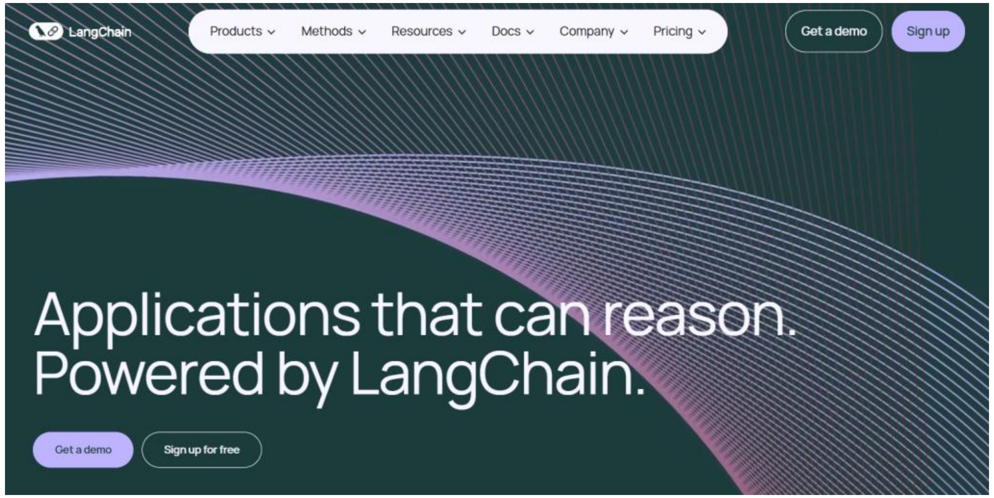
https://github.com/langchain-ai/langchain

What it is?

Framework designed to build applications using large language models (LLMs).

Popular use cases:
RAG, Autonomous agents.

Jun 2023, Fastest growing open-source project



Applications that can reason.
Powered by LangChain.

# Langchain – Core Concept

1. Chain multiple steps to create a more complex task
2. Can use any LLM (OpenAI, Llama, etc) Can use different LLMs in the same application
3. Prompt Templates: A better way to structure prompts
4. Built in support for Few Shots learning
5. Memory: Full conversation or summary of conversation

# N-Shot Learning

A machine learning paradigm where a model learns to classify or recognize new categories based on few examples per category.

# This Week

Week 7: Build Autonomous Agent - 1

N-Shot Learning, LangChain

Week 8-9: Build Autonomous Agent - 2

OpenAI APIs, Prompt Engineering

# OpenAI API

# OpenAI API

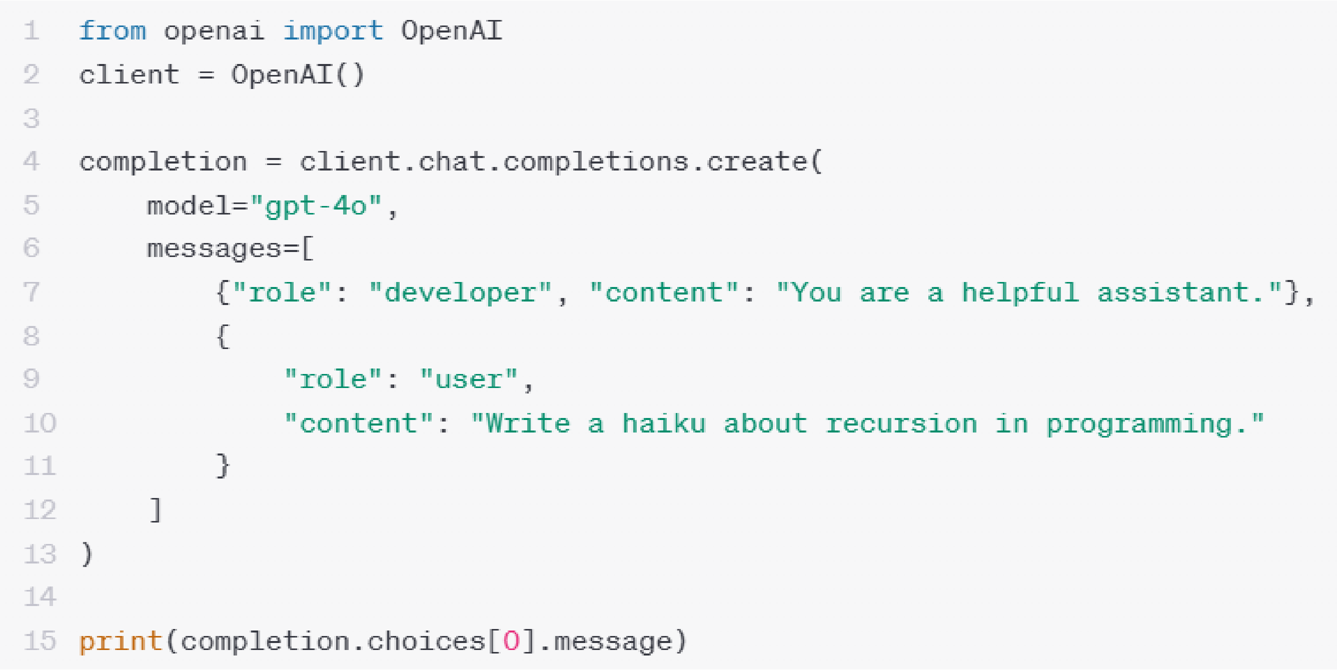
## What it is?

The OpenAI API allows developers to programmatically access various models for tasks like text generation, image creation, translation, summarization, etc.

# Text Generation

What it is: Generate Text from Prompt

Models: gpt-4o, gpt-4o-mini, o1 (o3?)



```python
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "developer", "content": "You are a helpful assistant."},
        {
            "role": "user",
            "content": "Write a haiku about recursion in programming."
        }
    ]
)

print(completion.choices[0].message)
```

# Azure OpenAI API

Azure AI Foundry

What it is?

The Azure OpenAI API allows developers to programmatically access OpenAI's models for a variety of tasks like text generation, code completion, image creation, summarization, and more, all within the secure and scalable environment provided by Microsoft Azure.

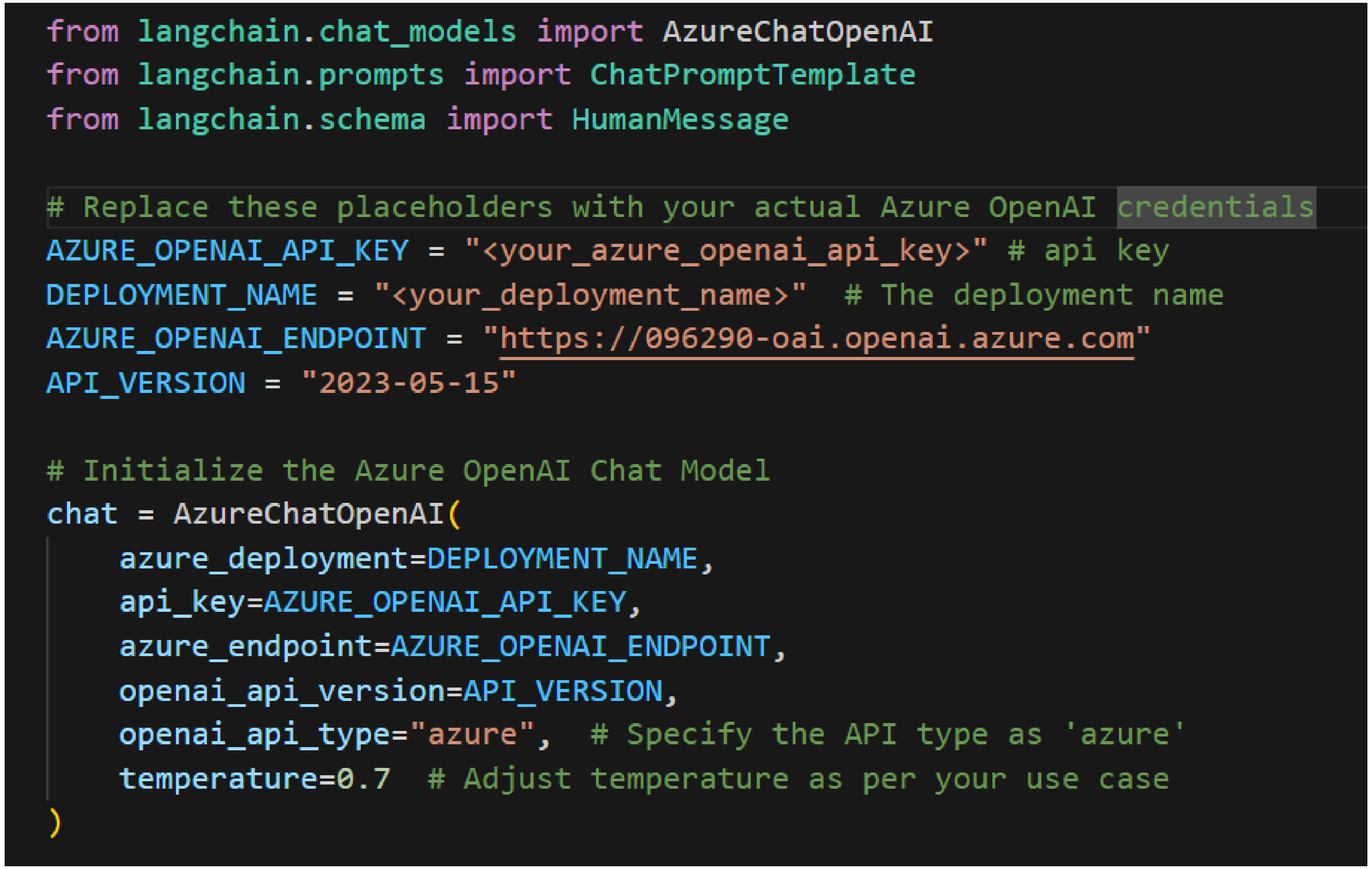
It offers enterprise-grade security, compliance, and regional availability, making it suitable for businesses to build AI-powered applications.

Documentation:

https://learn.microsoft.com/en-us/azure/ai-services/openai/

# Text Generation with Azure OpenAI API

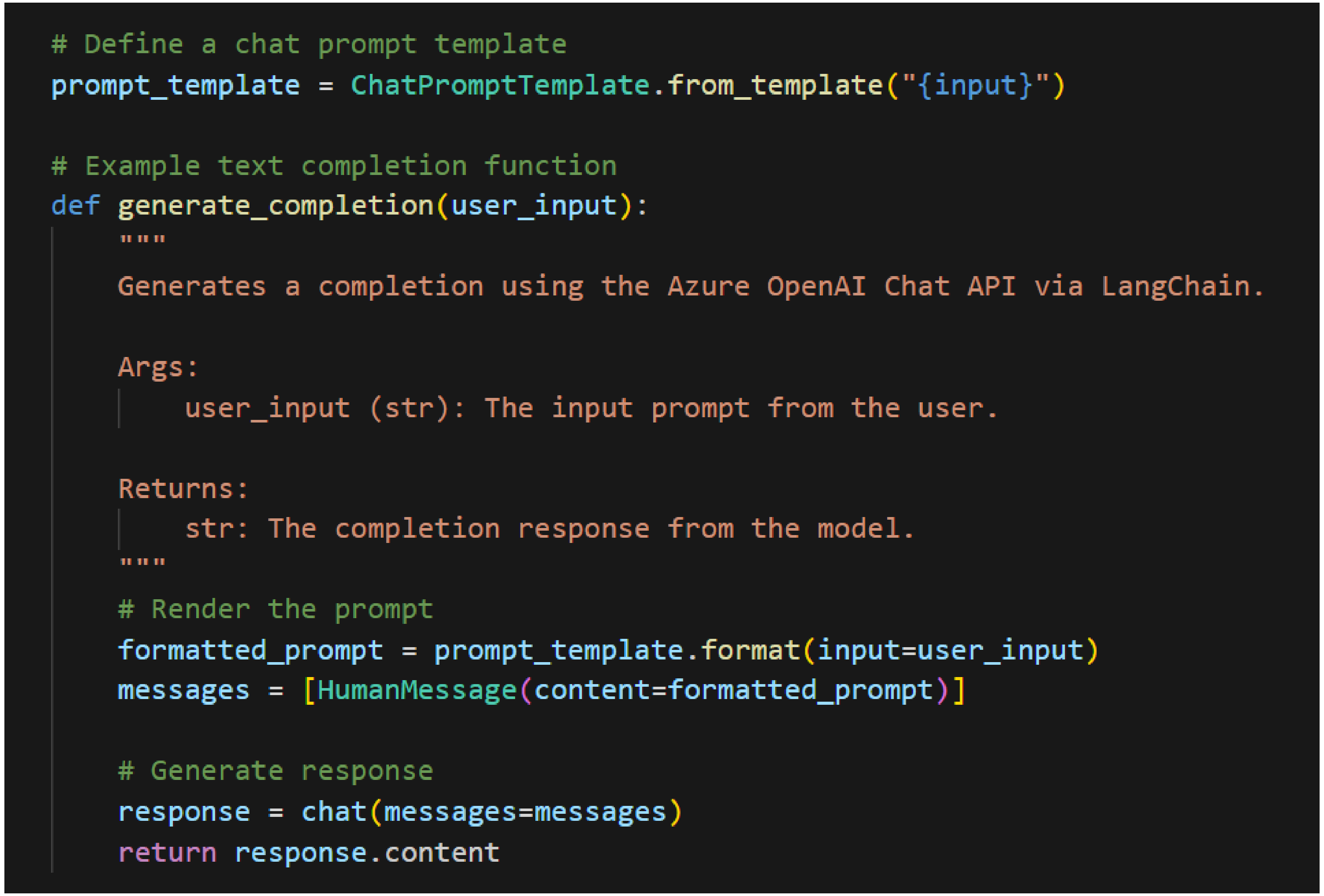
1 Initialize the model



```
from langchain.chat_models import AzureChatOpenAI
from langchain.prompts import ChatPromptTemplate
from langchain.schema import HumanMessage

# Replace these placeholders with your actual Azure OpenAI credentials
AZURE_OPENAI_API_KEY = "<your_azure_openai_api_key>" # api key
DEPLOYMENT_NAME = "<your_deployment_name>"  # The deployment name
AZURE_OPENAI_ENDPOINT = "https://096290-oai.openai.azure.com"
API_VERSION = "2023-05-15"

# Initialize the Azure OpenAI Chat Model
chat = AzureChatOpenAI(
    azure_deployment=DEPLOYMENT_NAME,
    api_key=AZURE_OPENAI_API_KEY,
    azure_endpoint=AZURE_OPENAI_ENDPOINT,
    openai_api_version=API_VERSION,
    openai_api_type="azure",  # Specify the API type as 'azure'
    temperature=0.7  # Adjust temperature as per your use case
)
```

2

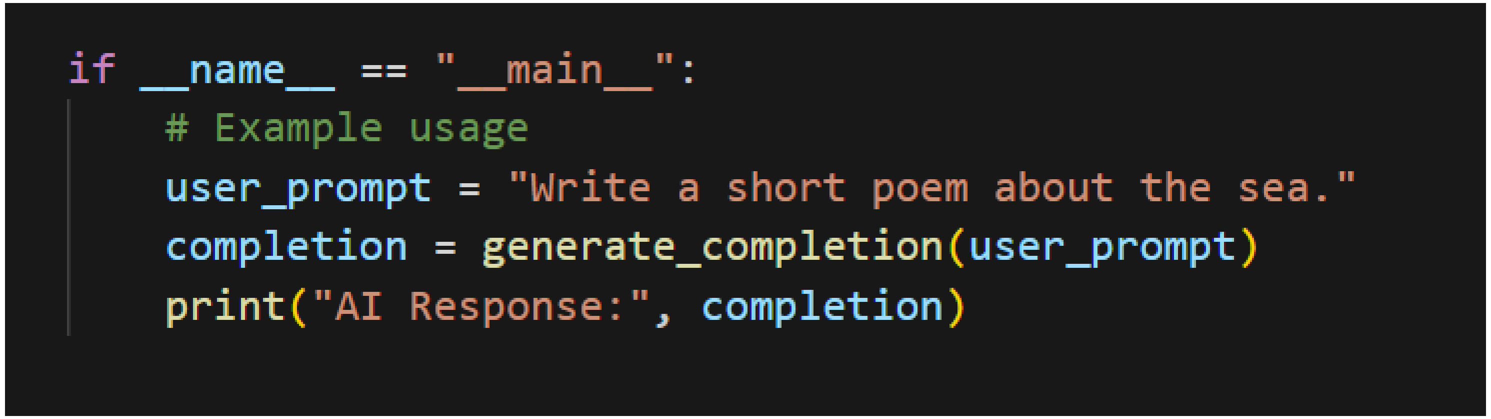
# Define prompt template & chat



```python
# Define a chat prompt template
prompt_template = ChatPromptTemplate.from_template("{input}")

# Example text completion function
def generate_completion(user_input):
    """
    Generates a completion using the Azure OpenAI Chat API via LangChain.

    Args:
        user_input (str): The input prompt from the user.

    Returns:
        str: The completion response from the model.
    """
    # Render the prompt
    formatted_prompt = prompt_template.format(input=user_input)
    messages = [HumanMessage(content=formatted_prompt)]

    # Generate response
    response = chat(messages=messages)
    return response.content
```

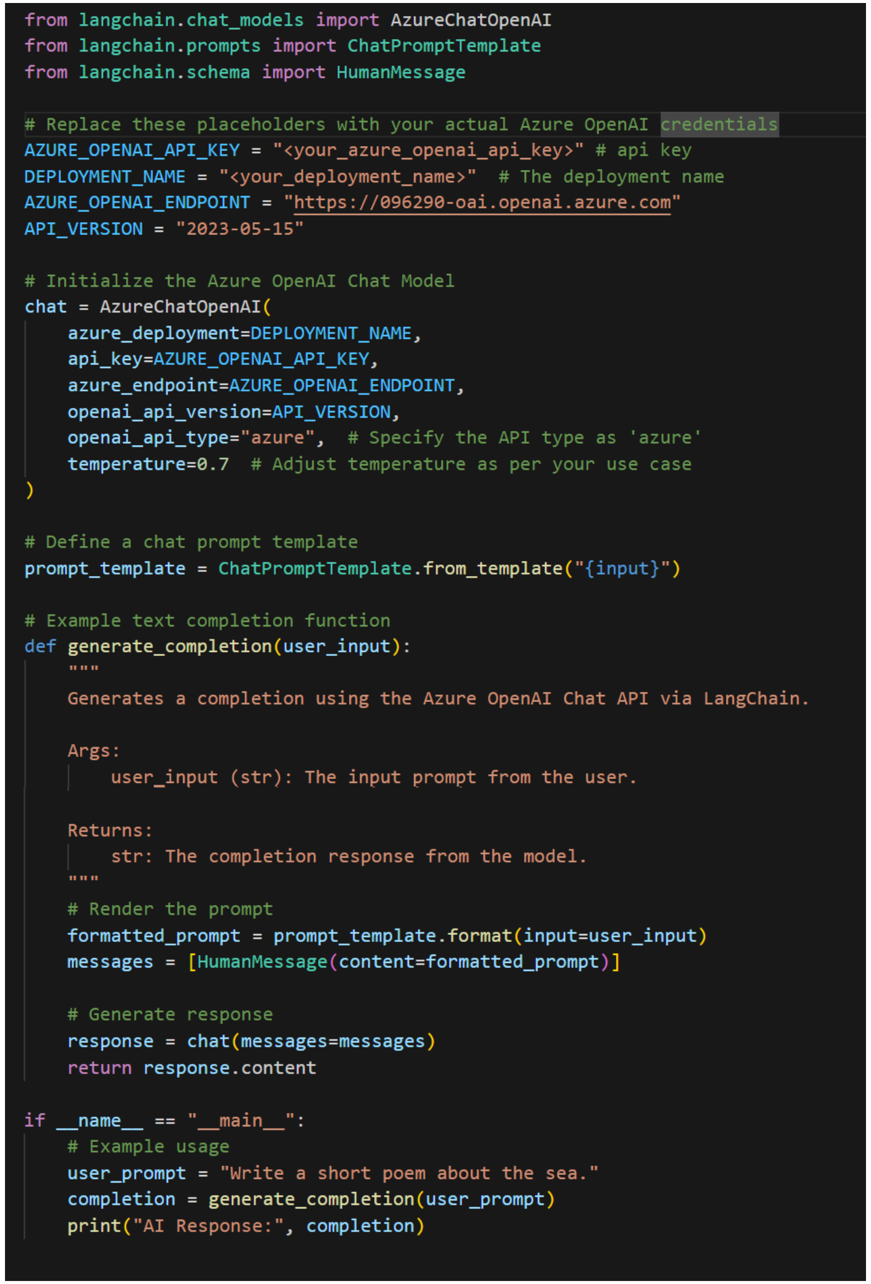
## 3

## Execute



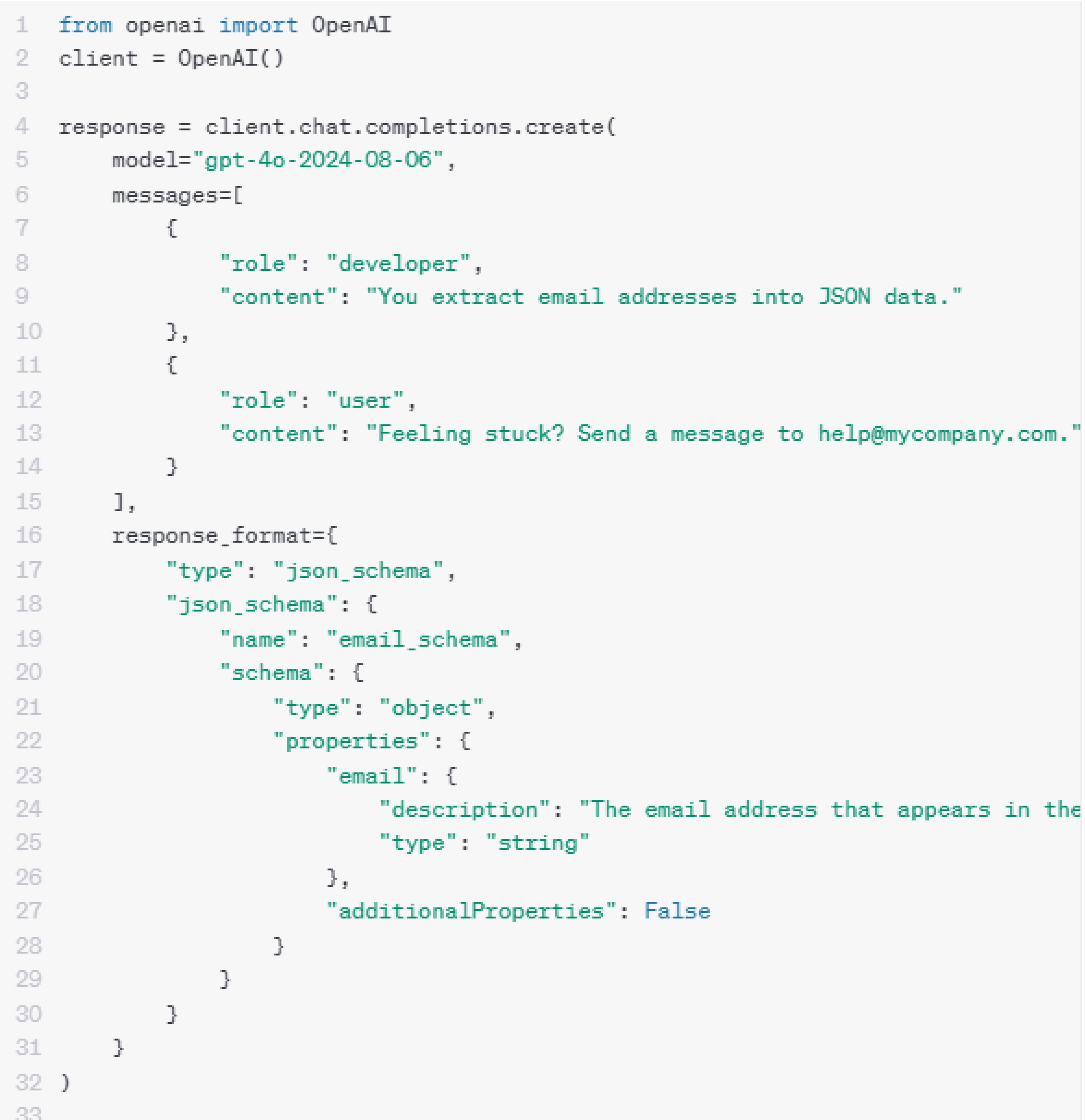
```
if __name__ == "__main__":
    # Example usage
    user_prompt = "Write a short poem about the sea."
    completion = generate_completion(user_prompt)
    print("AI Response:", completion)
```

# Full Code



```python
from langchain.chat_models import AzureChatOpenAI
from langchain.prompts import ChatPromptTemplate
from langchain.schema import HumanMessage

# Replace these placeholders with your actual Azure OpenAI credentials
AZURE_OPENAI_API_KEY = "<your_azure_openai_api_key>" # api key
DEPLOYMENT_NAME = "<your_deployment_name>"  # The deployment name
AZURE_OPENAI_ENDPOINT = "https://096290-oai.openai.azure.com"
API_VERSION = "2023-05-15"

# Initialize the Azure OpenAI Chat Model
chat = AzureChatOpenAI(
    azure_deployment=DEPLOYMENT_NAME,
    api_key=AZURE_OPENAI_API_KEY,
    azure_endpoint=AZURE_OPENAI_ENDPOINT,
    openai_api_version=API_VERSION,
    openai_api_type="azure",  # Specify the API type as 'azure'
    temperature=0.7  # Adjust temperature as per your use case
)

# Define a chat prompt template
prompt_template = ChatPromptTemplate.from_template("{input}")

# Example text completion function
def generate_completion(user_input):
    """
    Generates a completion using the Azure OpenAI Chat API via LangChain.

    Args:
        user_input (str): The input prompt from the user.

    Returns:
        str: The completion response from the model.
    """
    # Render the prompt
    formatted_prompt = prompt_template.format(input=user_input)
    messages = [HumanMessage(content=formatted_prompt)]

    # Generate response
    response = chat(messages=messages)
    return response.content

if __name__ == "__main__":
    # Example usage
    user_prompt = "Write a short poem about the sea."
    completion = generate_completion(user_prompt)
    print("AI Response:", completion)
```

# Generate Structured JSON from Prompt,

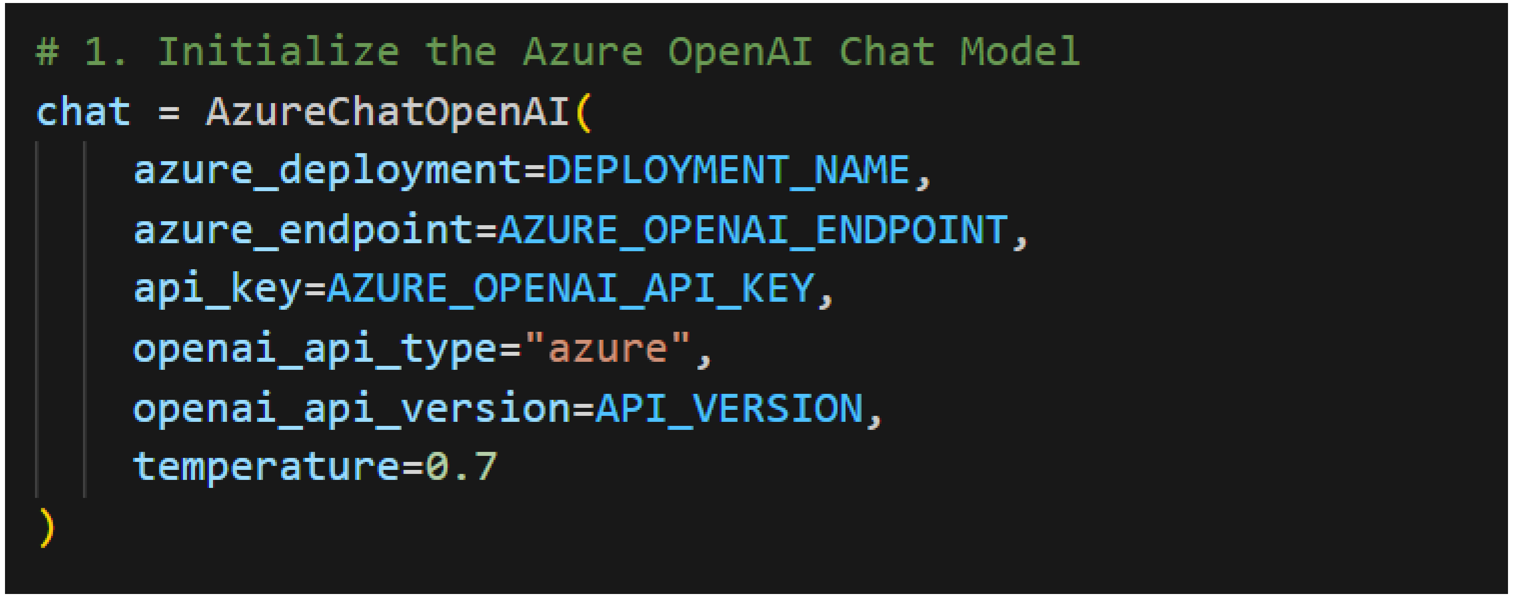


```python
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
    model="gpt-4o-2024-08-06",
    messages=[
        {
            "role": "developer",
            "content": "You extract email addresses into JSON data."
        },
        {
            "role": "user",
            "content": "Feeling stuck? Send a message to help@mycompany.com."
        }
    ],
    response_format={
        "type": "json_schema",
        "json_schema": {
            "name": "email_schema",
            "schema": {
                "type": "object",
                "properties": {
                    "email": {
                        "description": "The email address that appears in the
                        "type": "string"
                    },
                    "additionalProperties": False
                }
            }
        }
    }
)
```

# JSON Structured Text Generation with Azure OpenAI API

1

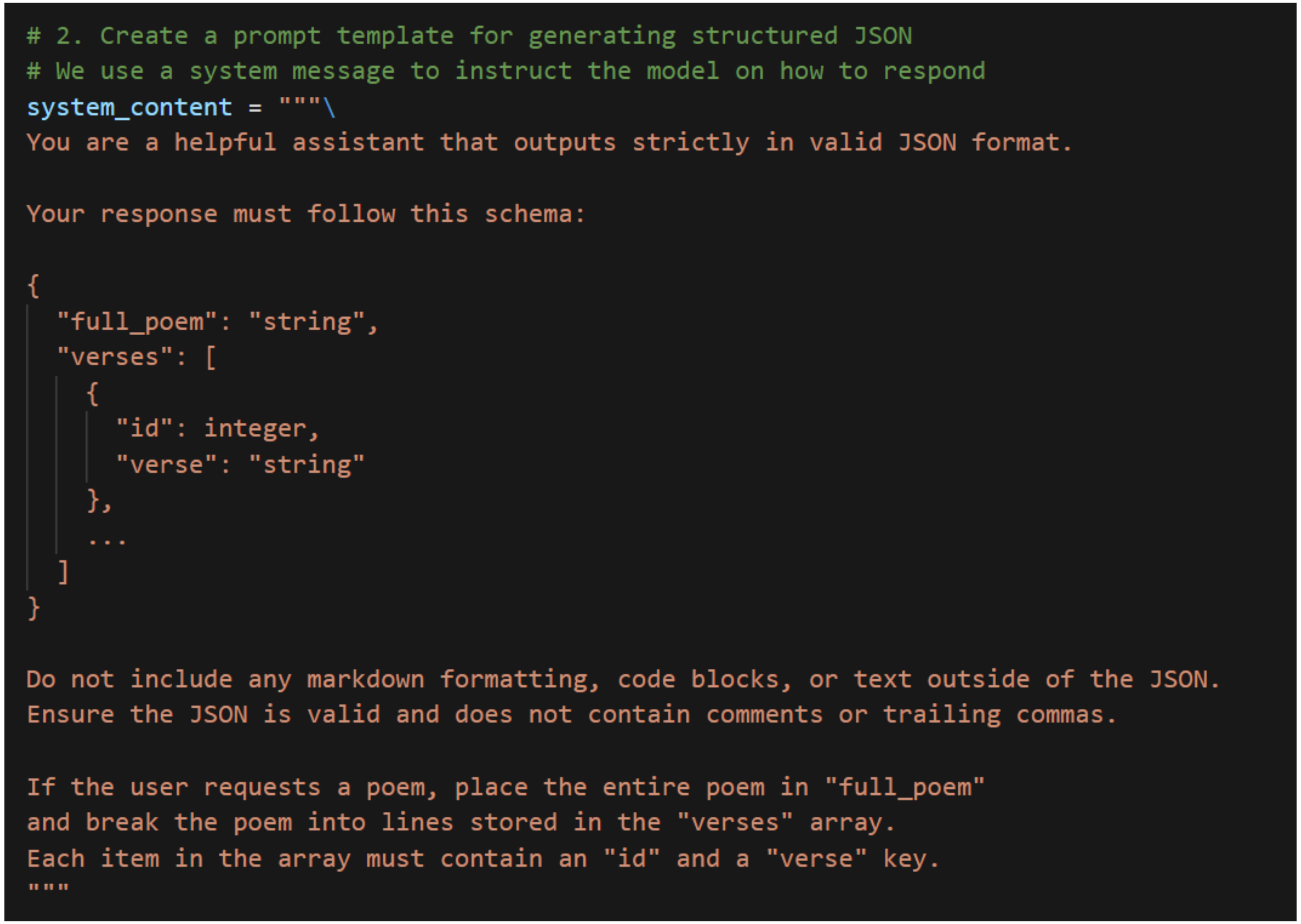
## Initialize the model



```
# 1. Initialize the Azure OpenAI Chat Model
chat = AzureChatOpenAI(
        azure_deployment=DEPLOYMENT_NAME,
        azure_endpoint=AZURE_OPENAI_ENDPOINT,
        api_key=AZURE_OPENAI_API_KEY,
        openai_api_type="azure",
        openai_api_version=API_VERSION,
        temperature=0.7
)
```

## 2

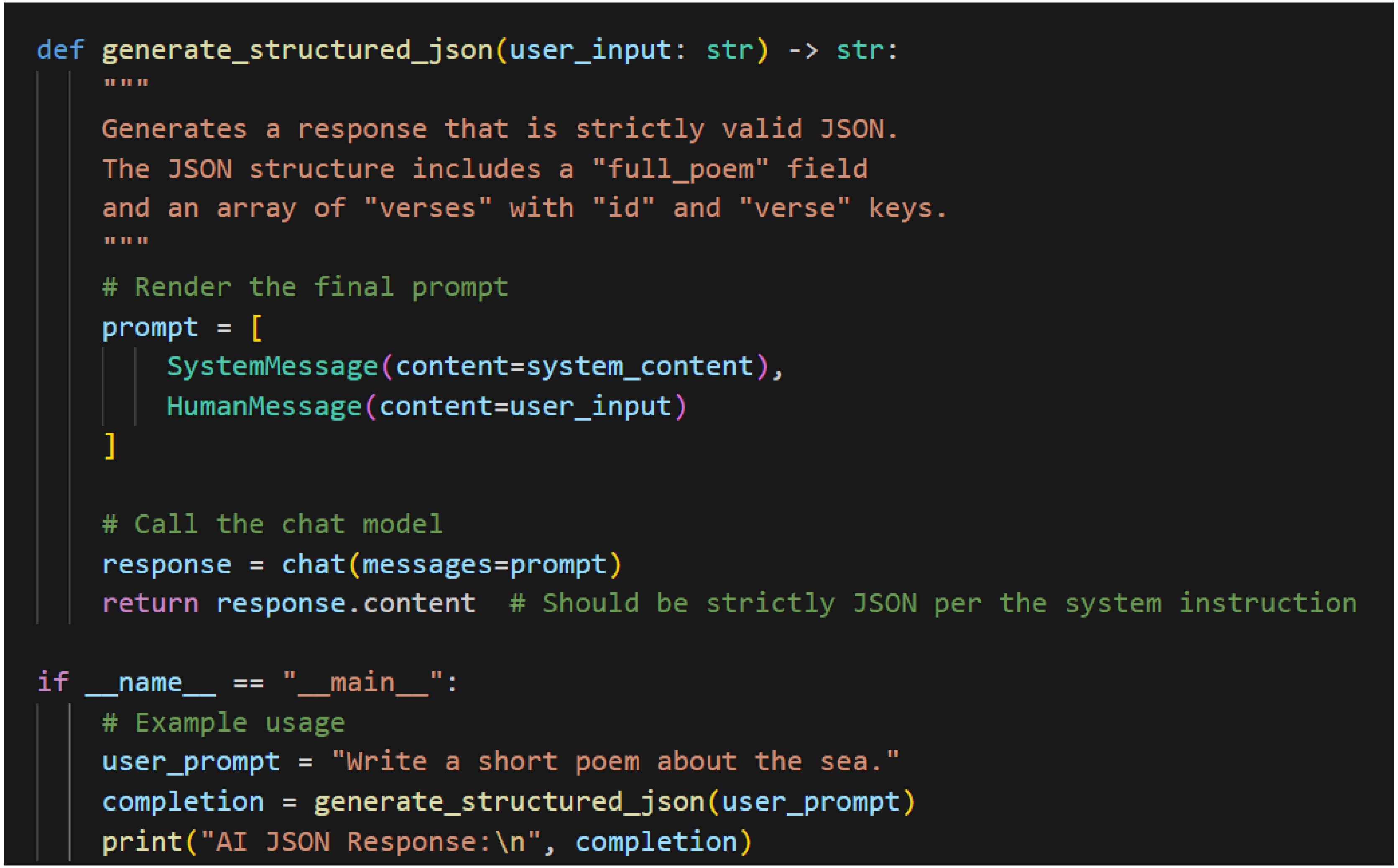
## Define system prompt template



```
# 2. Create a prompt template for generating structured JSON
# We use a system message to instruct the model on how to respond
system_content = """\
You are a helpful assistant that outputs strictly in valid JSON format.

Your response must follow this schema:

{
  "full_poem": "string",
  "verses": [
    {
      "id": integer,
      "verse": "string"
    },
    ...
  ]
}

Do not include any markdown formatting, code blocks, or text outside of the JSON.
Ensure the JSON is valid and does not contain comments or trailing commas.

If the user requests a poem, place the entire poem in "full_poem"
and break the poem into lines stored in the "verses" array.
Each item in the array must contain an "id" and a "verse" key.
"""
```

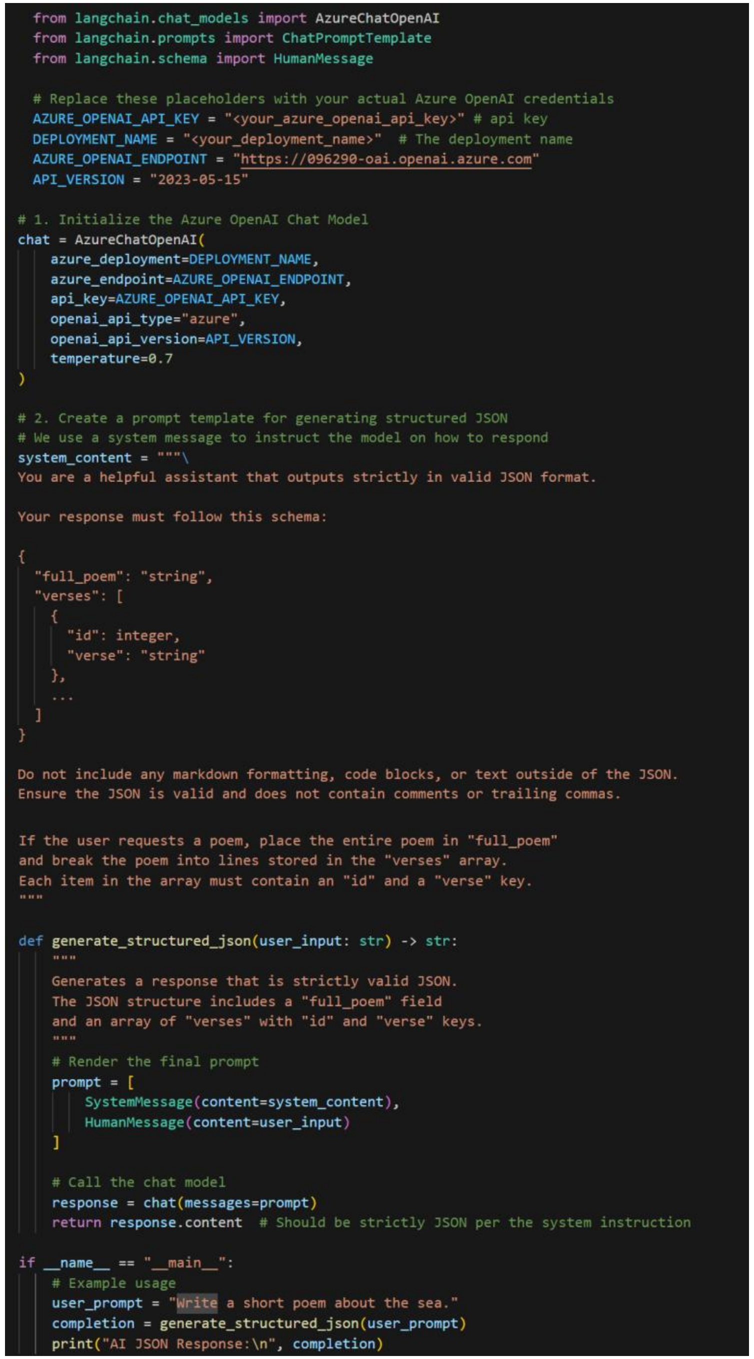
3

# Execute



```python
def generate_structured_json(user_input: str) -> str:
    """
    Generates a response that is strictly valid JSON.
    The JSON structure includes a "full_poem" field
    and an array of "verses" with "id" and "verse" keys.
    """
    # Render the final prompt
    prompt = [
        SystemMessage(content=system_content),
        HumanMessage(content=user_input)
    ]

    # Call the chat model
    response = chat(messages=prompt)
    return response.content  # Should be strictly JSON per the system instruction

if __name__ == "__main__":
    # Example usage
    user_prompt = "Write a short poem about the sea."
    completion = generate_structured_json(user_prompt)
    print("AI JSON Response:\n", completion)
```

# Full Code



```
  from langchain.chat_models import AzureChatOpenAI
  from langchain.prompts import ChatPromptTemplate
  from langchain.schema import HumanMessage

  # Replace these placeholders with your actual Azure OpenAI credentials
  AZURE_OPENAI_API_KEY = "<your_azure_openai_api_key>" # api key
  DEPLOYMENT_NAME = "<your_deployment_name>"  # The deployment name
  AZURE_OPENAI_ENDPOINT = "https://096290-oai.openai.azure.com"
  API_VERSION = "2023-05-15"

# 1. Initialize the Azure OpenAI Chat Model
chat = AzureChatOpenAI(
    azure_deployment=DEPLOYMENT_NAME,
    azure_endpoint=AZURE_OPENAI_ENDPOINT,
    api_key=AZURE_OPENAI_API_KEY,
    openai_api_type="azure",
    openai_api_version=API_VERSION,
    temperature=0.7
)

# 2. Create a prompt template for generating structured JSON
# We use a system message to instruct the model on how to respond
system_content = """\
You are a helpful assistant that outputs strictly in valid JSON format.

Your response must follow this schema:

{
  "full_poem": "string",
  "verses": [
    {
      "id": integer,
      "verse": "string"
    },
    ...
  ]
}

Do not include any markdown formatting, code blocks, or text outside of the JSON.
Ensure the JSON is valid and does not contain comments or trailing commas.

If the user requests a poem, place the entire poem in "full_poem"
and break the poem into lines stored in the "verses" array.
Each item in the array must contain an "id" and a "verse" key.
"""

def generate_structured_json(user_input: str) -> str:
    """
    Generates a response that is strictly valid JSON.
    The JSON structure includes a "full_poem" field
    and an array of "verses" with "id" and "verse" keys.
    """
    # Render the final prompt
    prompt = [
        SystemMessage(content=system_content),
        HumanMessage(content=user_input)
    ]

    # Call the chat model
    response = chat(messages=prompt)
    return response.content  # Should be strictly JSON per the system instruction

if __name__ == "__main__":
    # Example usage
    user_prompt = "Write a short poem about the sea."
    completion = generate_structured_json(user_prompt)
    print("AI JSON Response:\n", completion)
```

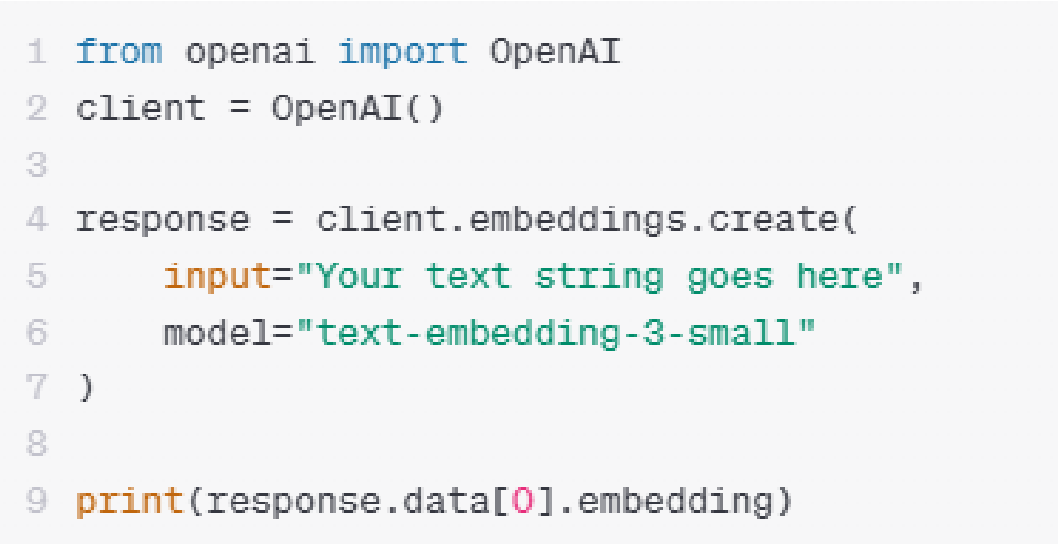
# Embedding

What it is: Embed text into a numerical vector in a multi dimensional space

Models: text-embedding-3-large, text-embedding-3-small, text-embedding-ada-002

Use cases

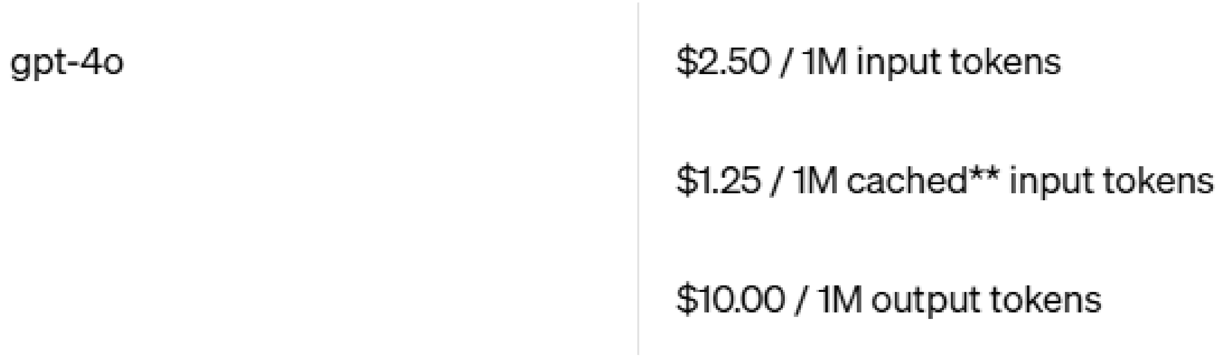
1. Search: Rank results by query relevance
2. Clustering: Group texts by similarity
3. Recommendation: Suggest related items or content
4. Anomaly Detection: Identify unrelated or outlier data
5. Classification: Assign labels to similar texts



```python
from openai import OpenAI
client = OpenAI()

response = client.embeddings.create(
    input="Your text string goes here",
    model="text-embedding-3-small"
)

print(response.data[0].embedding)
```

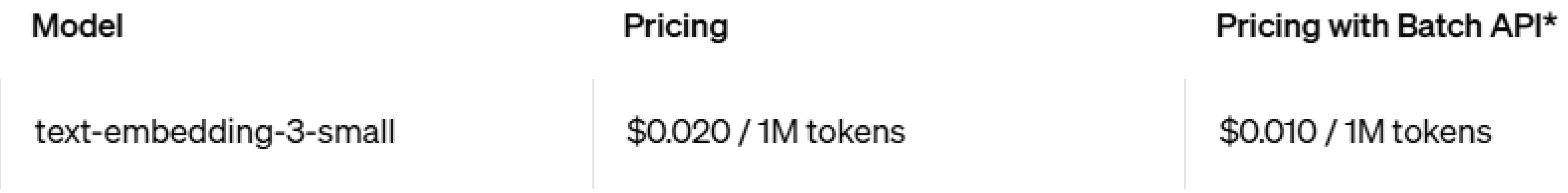
# Pricing (As of Jan 2025)

## Chat Completion



| | |
|---|---|
| gpt-4o | $2.50 / 1M input tokens<br>$1.25 / 1M cached** input tokens<br>$10.00 / 1M output tokens |

## Embedding



| Model | Pricing | Pricing with Batch API* |
|---|---|---|
| text-embedding-3-small | $0.020 / 1M tokens | $0.010 / 1M tokens |

# Other OpenAI Models

# Image Generation

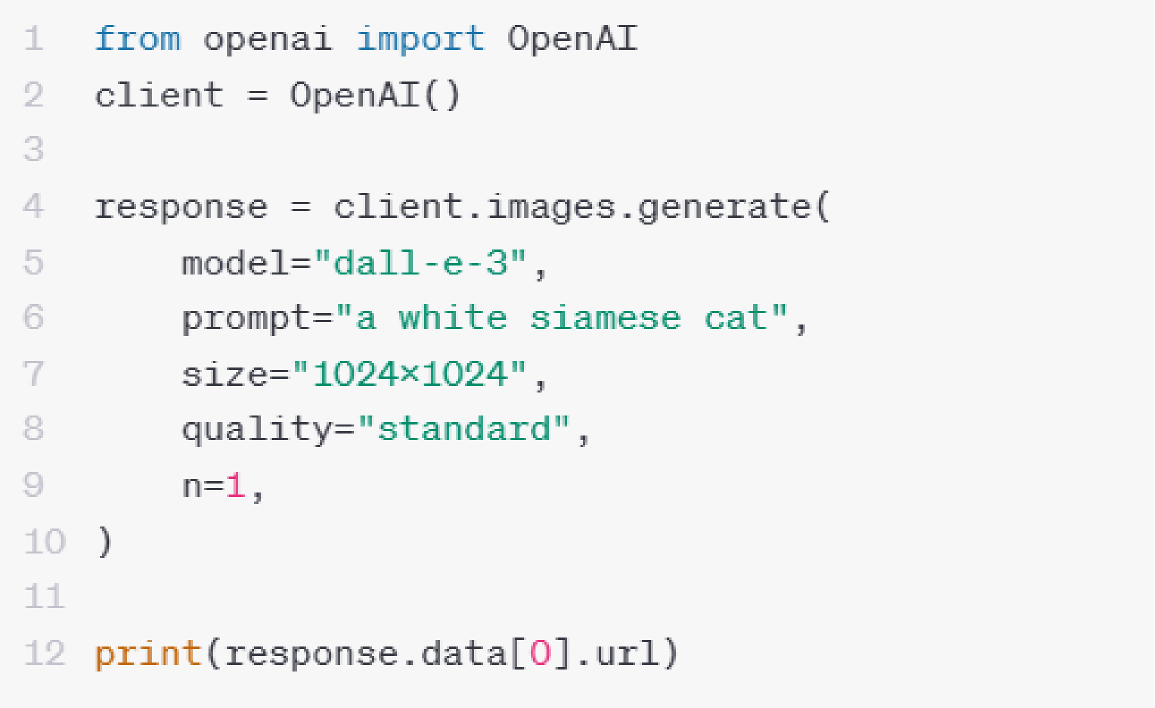
What it is: Generate Image from prompt (different sizes and quality)

Models: dall-e-3, dall-e-2

Options:

Image quality: Standard/HD

Size:
- 1024X1024 (box)
- 1792x1024 (horizontal)
- 1024x1792 (vertical)



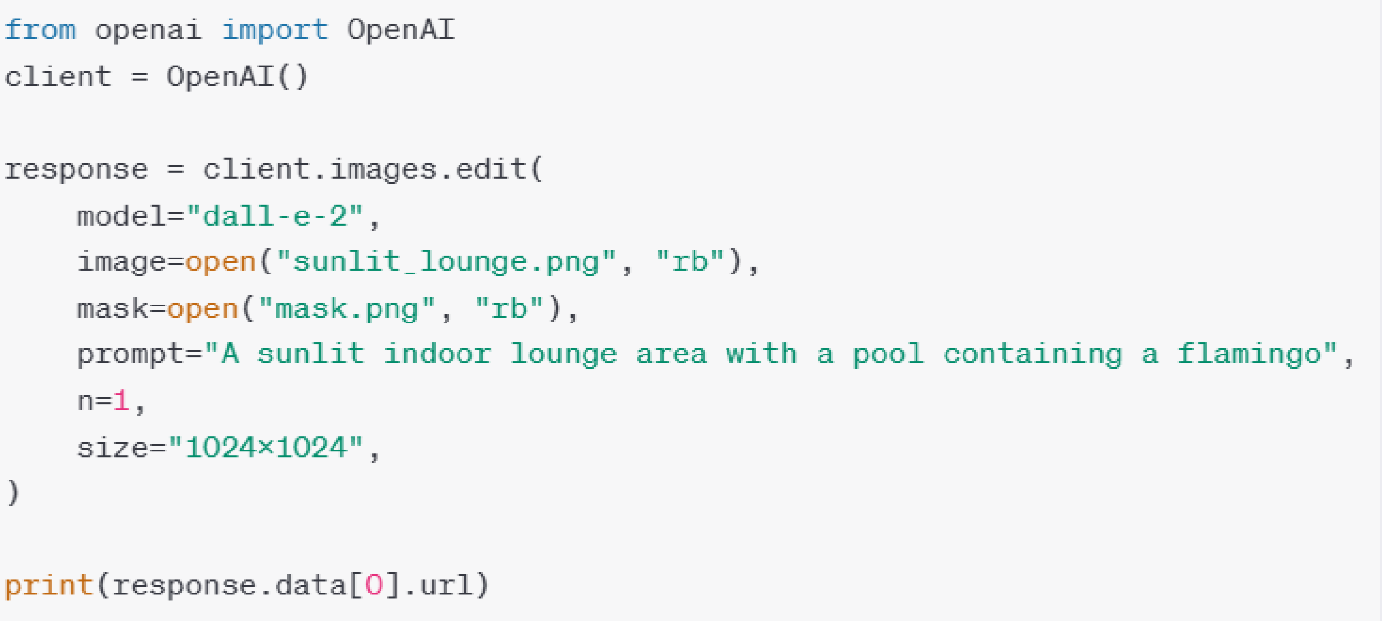
```python
from openai import OpenAI
client = OpenAI()

response = client.images.generate(
    model="dall-e-3",
    prompt="a white siamese cat",
    size="1024×1024",
    quality="standard",
    n=1,
)

print(response.data[0].url)
```

# Image Generation - Edits

<u>What it is:</u> Inpainting, Edit image using mask

<u>Models:</u> dall-e-2

* Prompt should include the whole image, not just the area that will be replaced



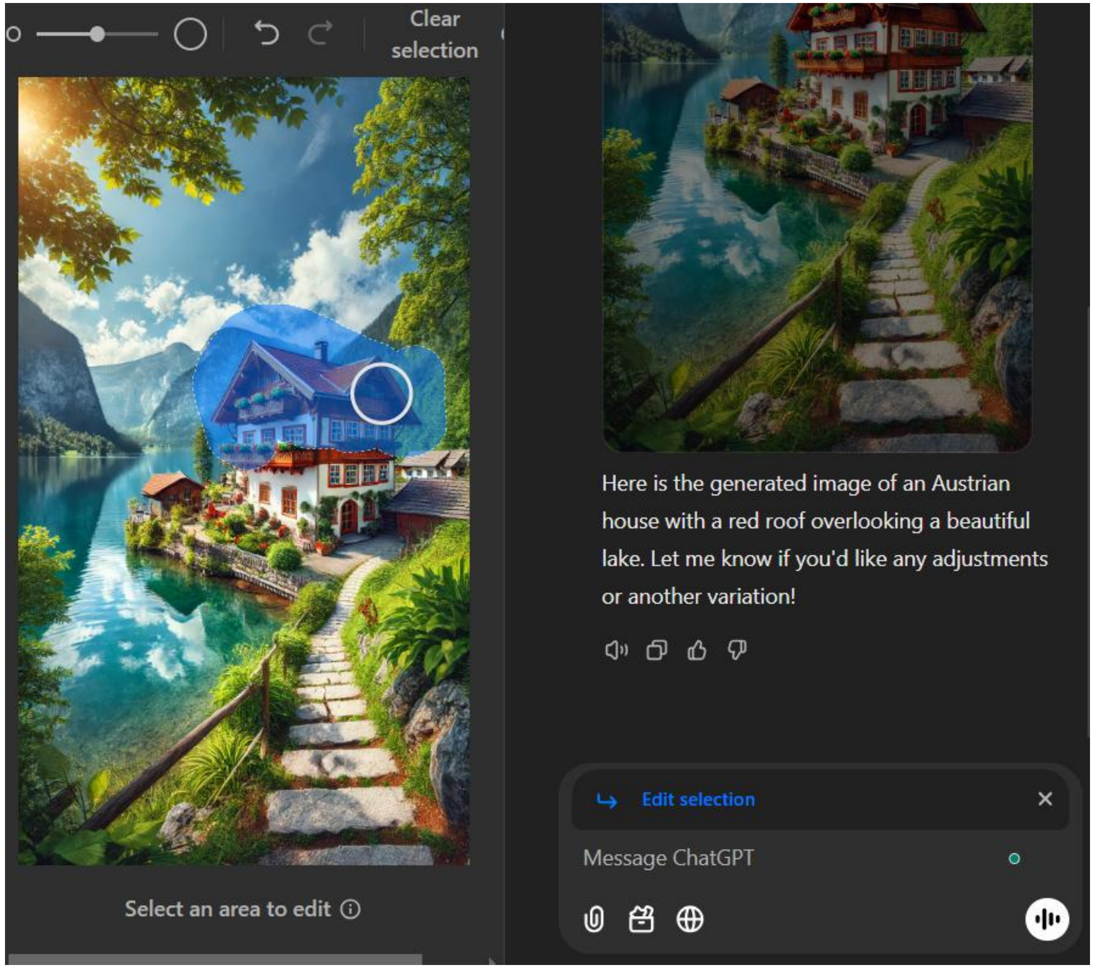
```python
from openai import OpenAI
client = OpenAI()

response = client.images.edit(
    model="dall-e-2",
    image=open("sunlit_lounge.png", "rb"),
    mask=open("mask.png", "rb"),
    prompt="A sunlit indoor lounge area with a pool containing a flamingo",
    n=1,
    size="1024×1024",
)

print(response.data[0].url)
```

# Image Generation - Edits

<u>What it is:</u> Inpainting, Edit image using mask

<u>Models:</u> dall-e-2

* Prompt should include the whole image, not just the area that will be replacd

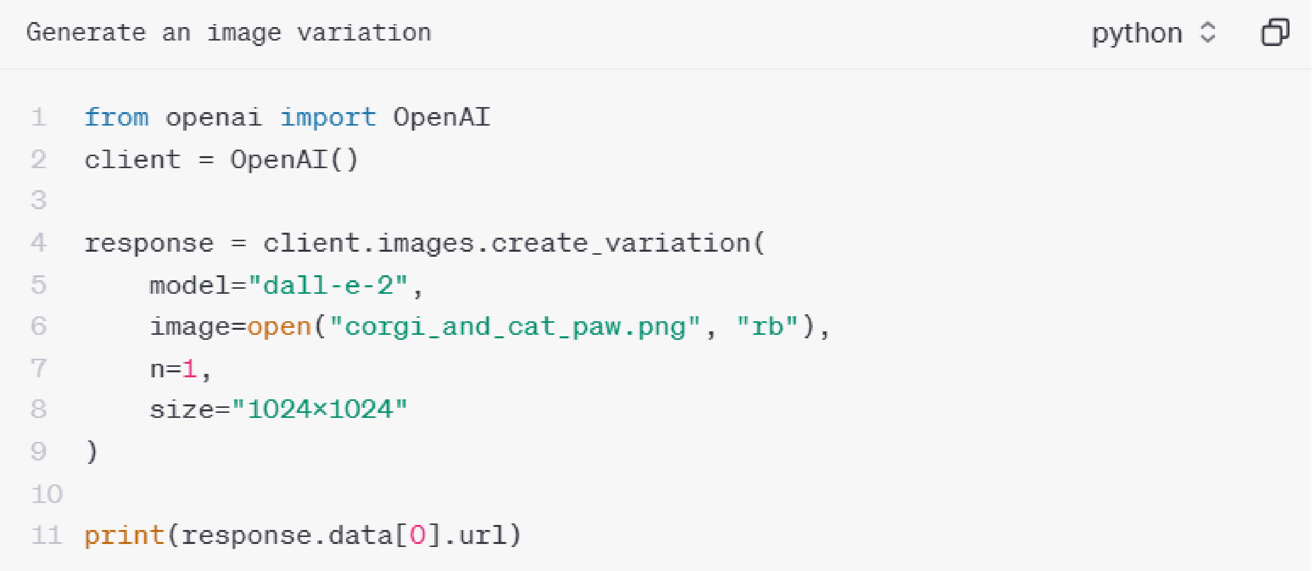


Here is the generated image of an Austrian house with a red roof overlooking a beautiful lake. Let me know if you'd like any adjustments or another variation!

# Image Generation - Variations

What it is: Variations on an image

Models: dall-e-2



```python
from openai import OpenAI
client = OpenAI()

response = client.images.create_variation(
    model="dall-e-2",
    image=open("corgi_and_cat_paw.png", "rb"),
    n=1,
    size="1024x1024"
)

print(response.data[0].url)
```

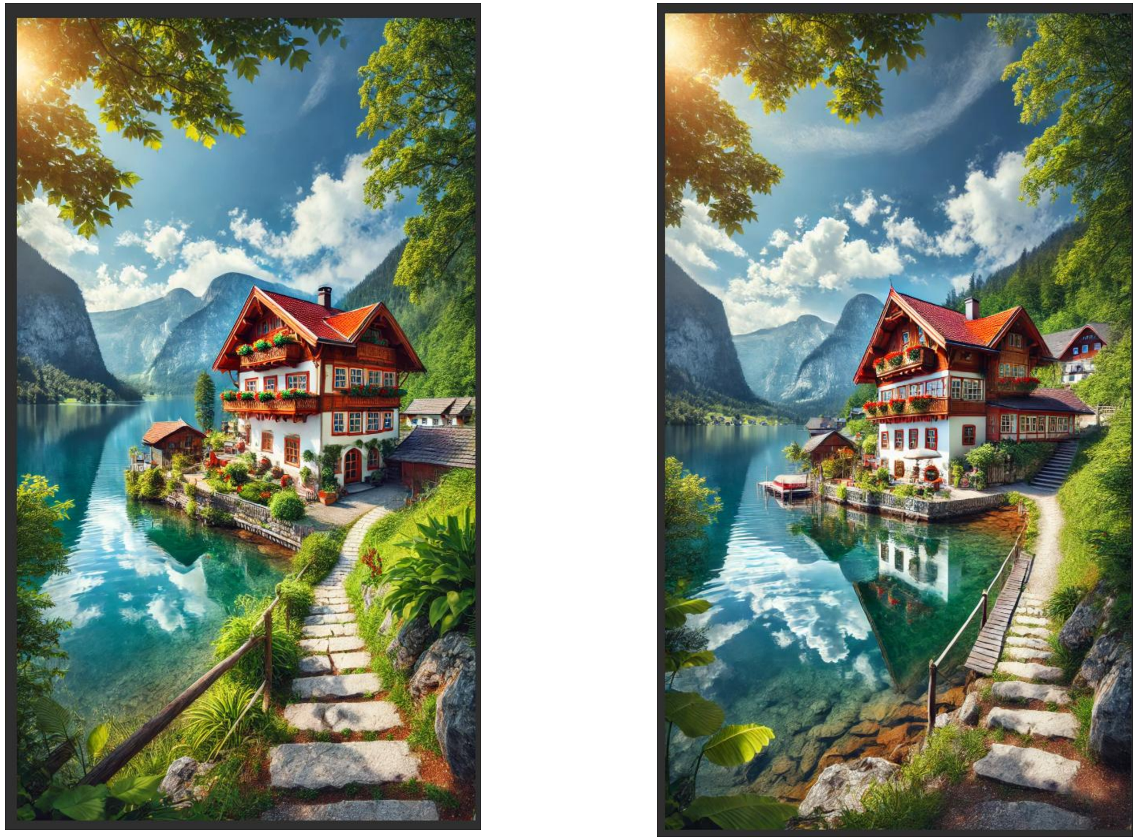
# Image Generation - Variations

What it is: Variations on an image

Models: dall-e-2

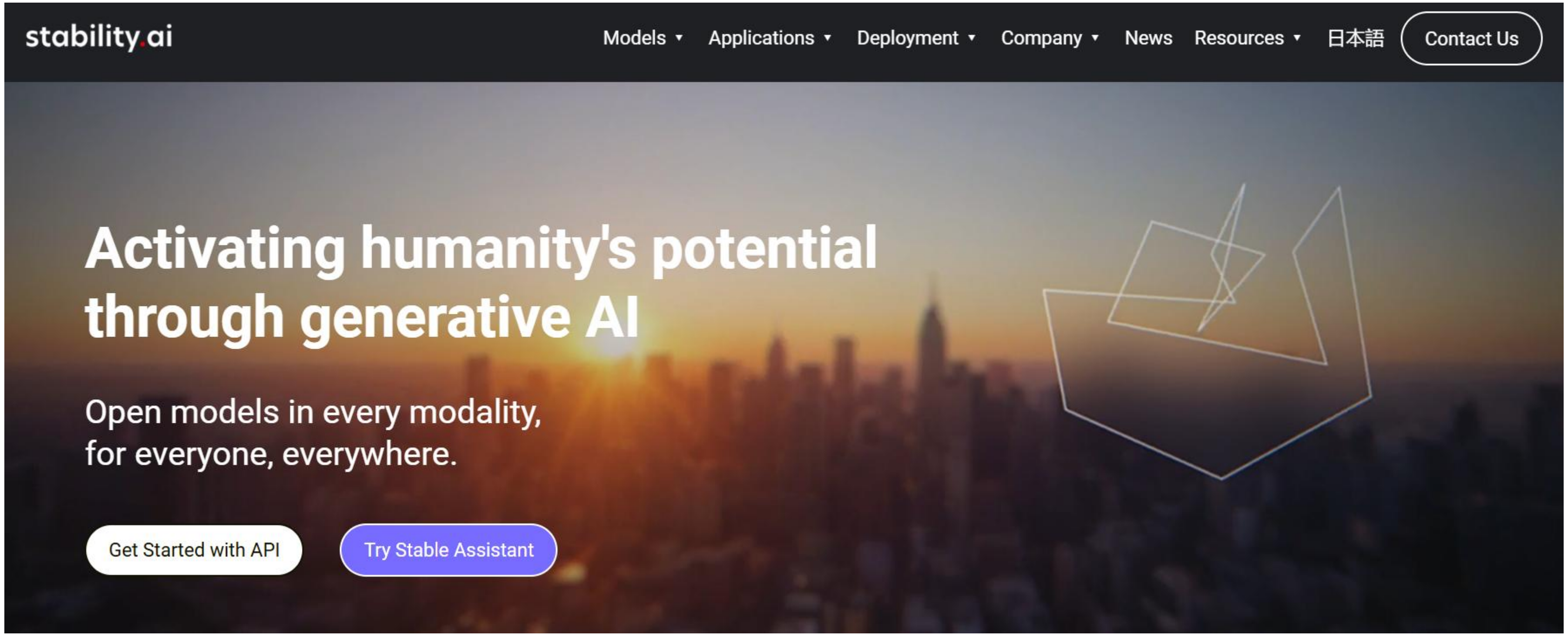
How it’s done?

Stable diffusion



Open AI Doesn’t offer very strong stable diffusions models

<u>One Option:</u> Stability.ai (Open Source)



Activating humanity's potential through generative AI

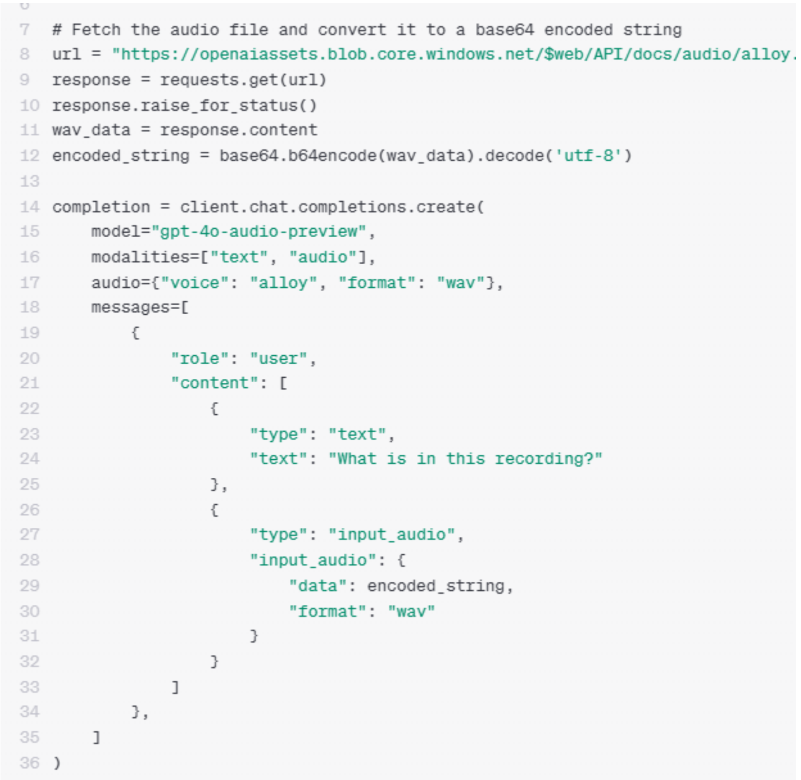
Open models in every modality, for everyone, everywhere.

# Audio Generation

What it is: Generate and understand audio

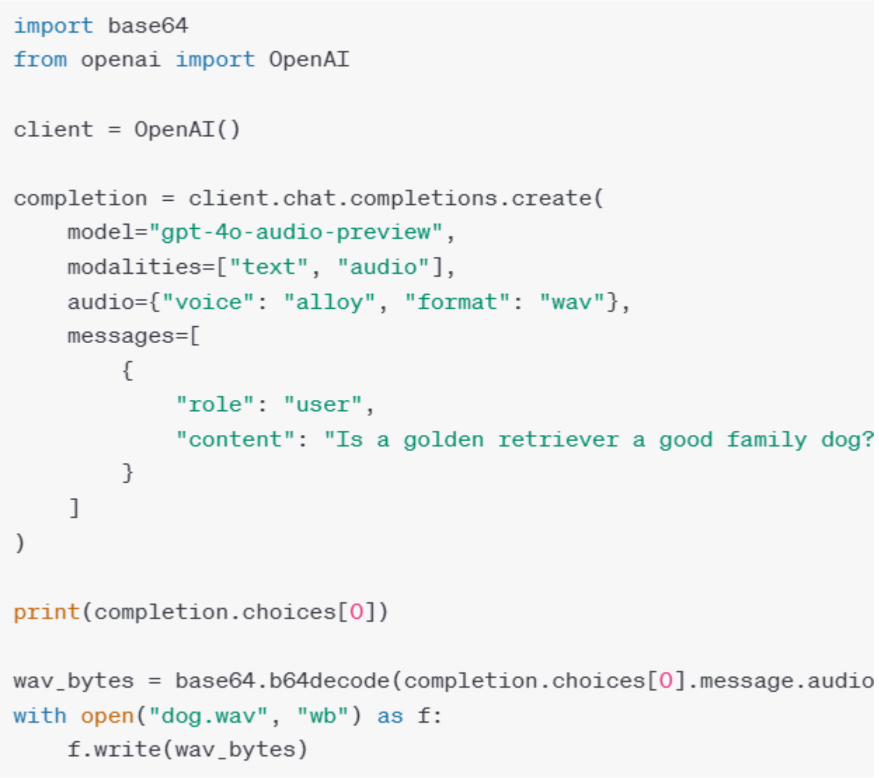
Models: gpt-4o-audio-preview

Input Audio and Ask questions



```
# Fetch the audio file and convert it to a base64 encoded string
url = "https://openaiassets.blob.core.windows.net/$web/API/docs/audio/alloy.
response = requests.get(url)
response.raise_for_status()
wav_data = response.content
encoded_string = base64.b64encode(wav_data).decode('utf-8')

completion = client.chat.completions.create(
    model="gpt-4o-audio-preview",
    modalities=["text", "audio"],
    audio={"voice": "alloy", "format": "wav"},
    messages=[
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": "What is in this recording?"
                },
                {
                    "type": "input_audio",
                    "input_audio": {
                        "data": encoded_string,
                        "format": "wav"
                    }
                }
            ]
        },
    ]
)
```

Prompt Model and request the output in audio format



```
import base64
from openai import OpenAI

client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-audio-preview",
    modalities=["text", "audio"],
    audio={"voice": "alloy", "format": "wav"},
    messages=[
        {
            "role": "user",
            "content": "Is a golden retriever a good family dog?
        }
    ]
)

print(completion.choices[0])

wav_bytes = base64.b64decode(completion.choices[0].message.audio
with open("dog.wav", "wb") as f:
    f.write(wav_bytes)
```

# Audio Generation

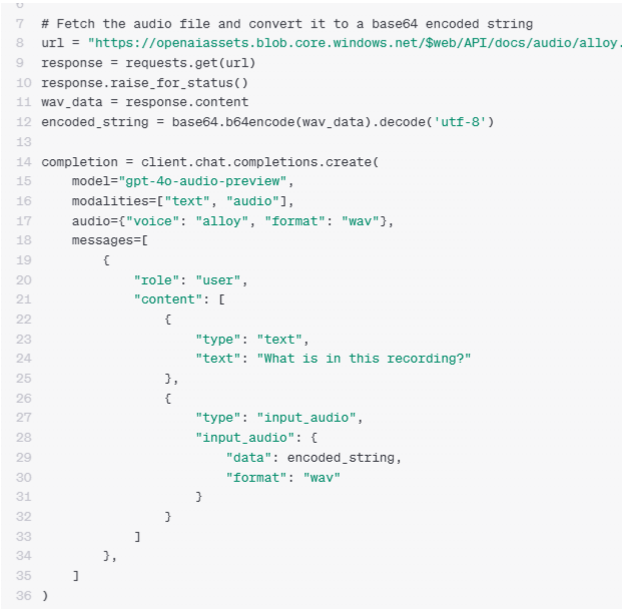
What it is: Generate and understand audio

Models: gpt-4o-audio-preview

## Use cases:

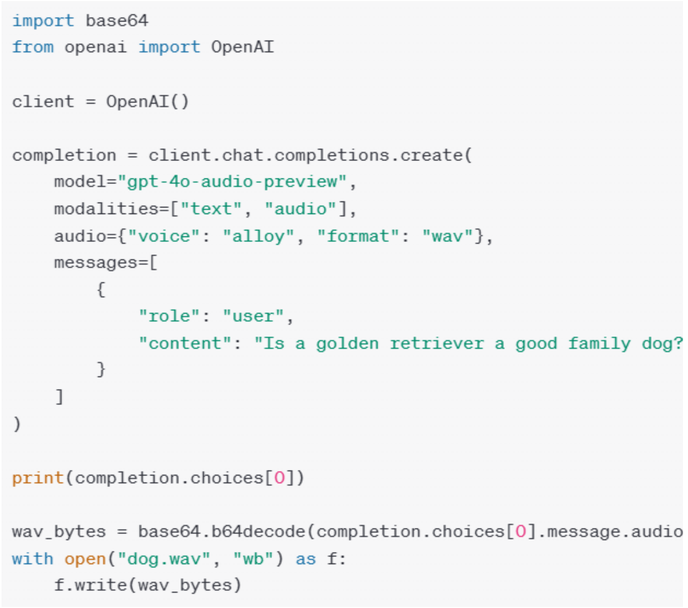
1. Generate Spoken Audio
2. Sentiment Analysis on recording
3. Converse with model

### Audio Input



```
# Fetch the audio file and convert it to a base64 encoded string
url = "https://openaiassets.blob.core.windows.net/$web/API/docs/audio/alloy.
response = requests.get(url)
response.raise_for_status()
wav_data = response.content
encoded_string = base64.b64encode(wav_data).decode('utf-8')

completion = client.chat.completions.create(
    model="gpt-4o-audio-preview",
    modalities=["text", "audio"],
    audio={"voice": "alloy", "format": "wav"},
    messages=[
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": "What is in this recording?"
                },
                {
                    "type": "input_audio",
                    "input_audio": {
                        "data": encoded_string,
                        "format": "wav"
                    }
                }
            ]
        },
    ]
)
```

### Audio Output

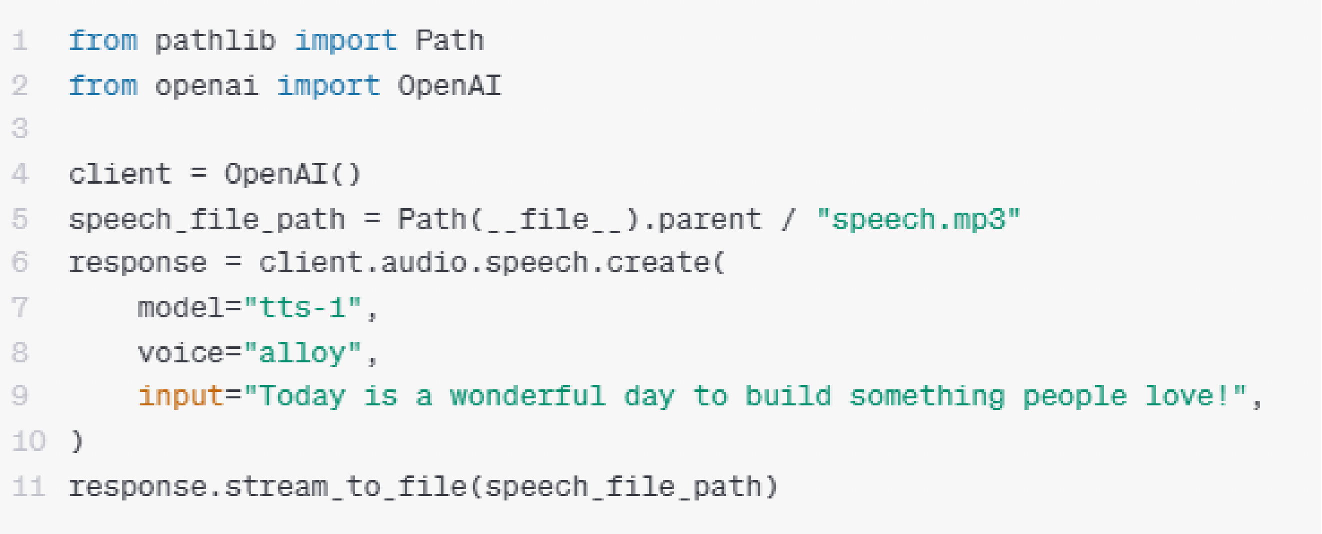


```
import base64
from openai import OpenAI

client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-audio-preview",
    modalities=["text", "audio"],
    audio={"voice": "alloy", "format": "wav"},
    messages=[
        {
            "role": "user",
            "content": "Is a golden retriever a good family dog?
        }
    ]
)

print(completion.choices[0])

wav_bytes = base64.b64decode(completion.choices[0].message.audio
with open("dog.wav", "wb") as f:
    f.write(wav_bytes)
```

# Text To Speech (TTS)

What it is: Generate speech according to text input

Models: tts-1



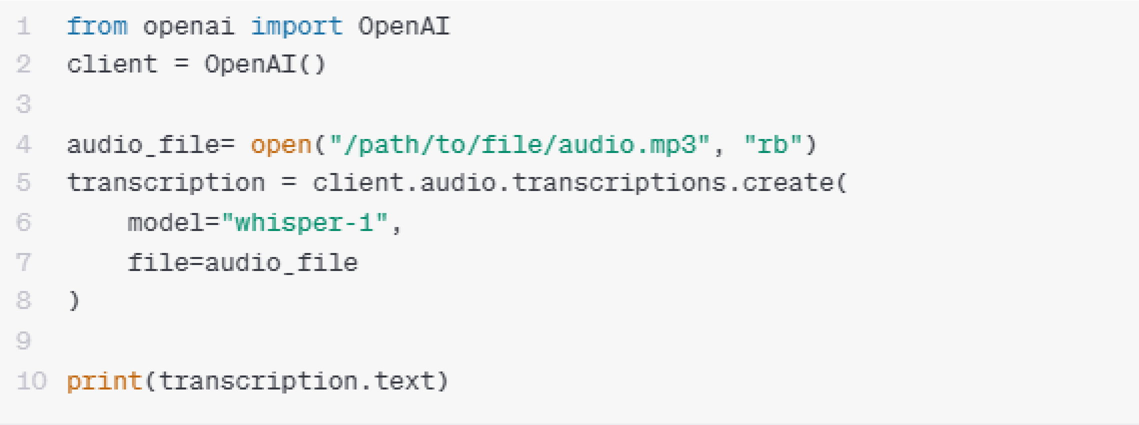
```python
from pathlib import Path
from openai import OpenAI

client = OpenAI()
speech_file_path = Path(__file__).parent / "speech.mp3"
response = client.audio.speech.create(
    model="tts-1",
    voice="alloy",
    input="Today is a wonderful day to build something people love!",
)
response.stream_to_file(speech_file_path)
```

# Speech To Text (Transcribe & Translate)

What it is: Transcribe or Translate speech to text

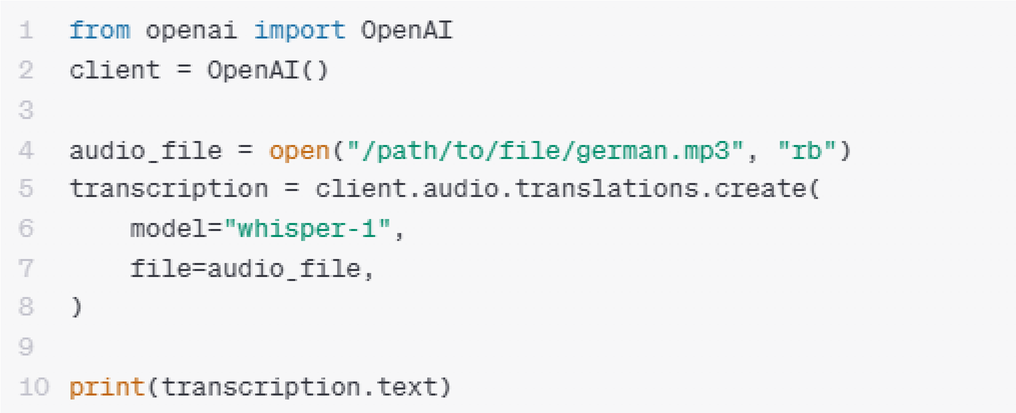
Models: whisper-1

Transcribe



```
from openai import OpenAI
client = OpenAI()

audio_file= open("/path/to/file/audio.mp3", "rb")
transcription = client.audio.transcriptions.create(
    model="whisper-1",
    file=audio_file
)

print(transcription.text)
```

Translate



```
from openai import OpenAI
client = OpenAI()

audio_file = open("/path/to/file/german.mp3", "rb")
transcription = client.audio.translations.create(
    model="whisper-1",
    file=audio_file,
)

print(transcription.text)
```

# What are the key factors in autonomous agent quality?

# Prompt Engineering

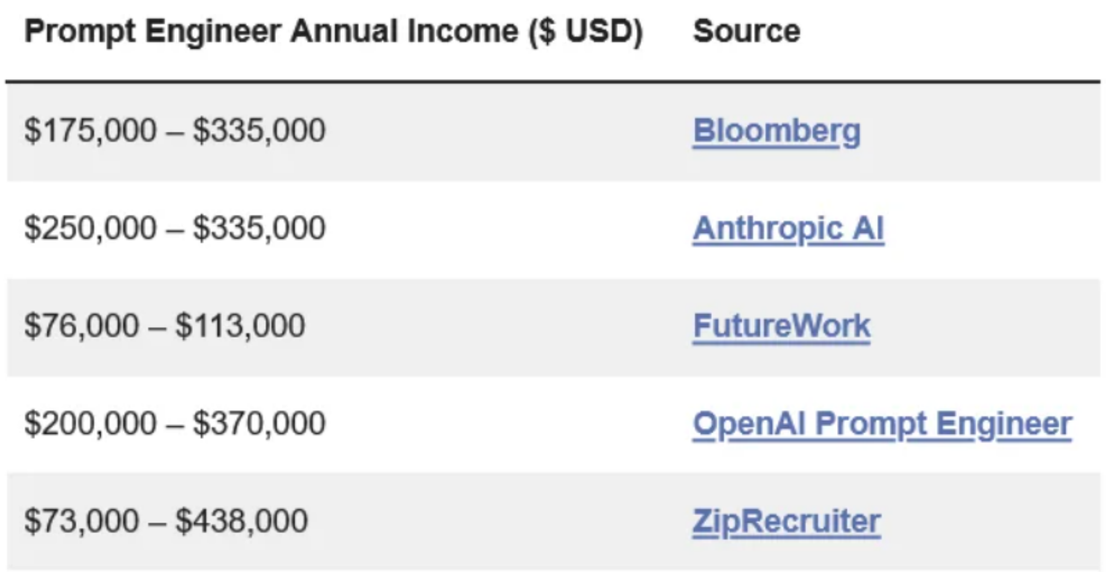
# Prompt Engineering

What it is:

Prompt engineering is a systematic approach to designing and refining input instructions for AI models to achieve specific, accurate, and contextually relevant outputs.

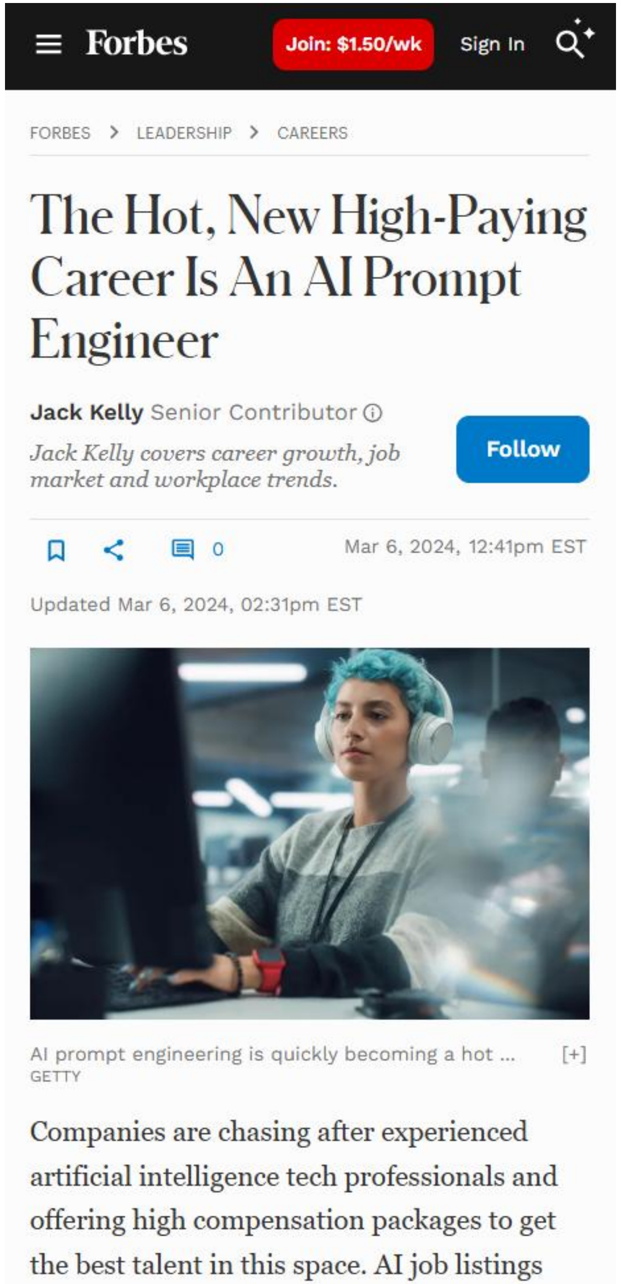
It involves structuring queries and providing contextual information in a way that aligns with the model's capabilities, ensuring it generates responses that meet user expectations.

# Prompt Engineering



| Prompt Engineer Annual Income ($ USD) | Source |
|---|---|
| $175,000 – $335,000 | Bloomberg |
| $250,000 – $335,000 | Anthropic AI |
| $76,000 – $113,000 | FutureWork |
| $200,000 – $370,000 | OpenAI Prompt Engineer |
| $73,000 – $438,000 | ZipRecruiter |

Average FS developer salary: $125K



The Hot, New High-Paying Career Is An AI Prompt Engineer

**Jack Kelly** Senior Contributor

*Jack Kelly covers career growth, job market and workplace trends.*

Mar 6, 2024, 12:41pm EST

Updated Mar 6, 2024, 02:31pm EST

AI prompt engineering is quickly becoming a hot ... [+]
GETTY

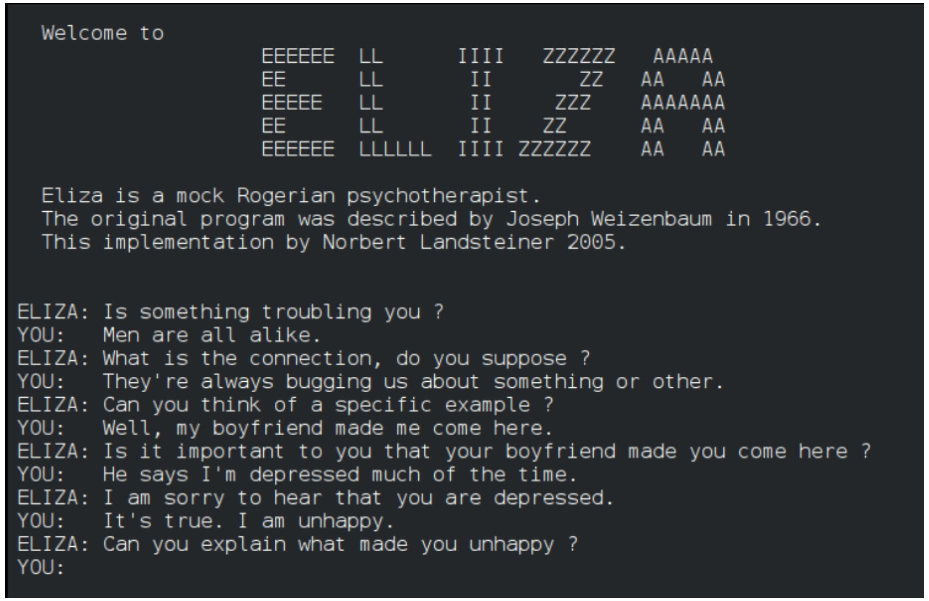
Companies are chasing after experienced artificial intelligence tech professionals and offering high compensation packages to get the best talent in this space. AI job listings

# First Hire for Prompt Engineer?

Time: 1964

Model: ELIZA

Early Natural Language processing Model for Emotional Therapy.



```
  Welcome to
                    EEEEEE  LL      IIII   ZZZZZZ   AAAAA
                    EE      LL       II       ZZ   AA   AA
                    EEEEE   LL       II     ZZZ    AAAAAAA
                    EE      LL       II    ZZ      AA   AA
                    EEEEEE  LLLLLL  IIII ZZZZZZ    AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

# Prompt Engineering

What it does:

1. Optimize Agent Behaviour Step-by-step processes, adjust required output, tone & style.
2. Improve Task Understanding Agents can handle complex tasks efficiently with clear structured prompts.
3. Reduce Hallucinations Helps AI avoid incorrect or non existing information
4. Enhances Context Awareness ensures the agent understands the scope, domain, and specific instructions required for a task.
5. Reduce development time systematic approach

# Instruction-Based Prompting

Purpose:

Directly tell the AI what to do in clear terms.

When to Use:

When you need the AI to perform specific tasks, such as summarization, answering questions, or generating structured content.

Best Practices:

Be specific about the format, tone, and output length.

Example:

“Summarize the following article in 3 bullet points: [Insert Article].”

# The Persona Pattern

Purpose:

Set a specific persona or context for the AI to generate more relevant responses.

When to Use:

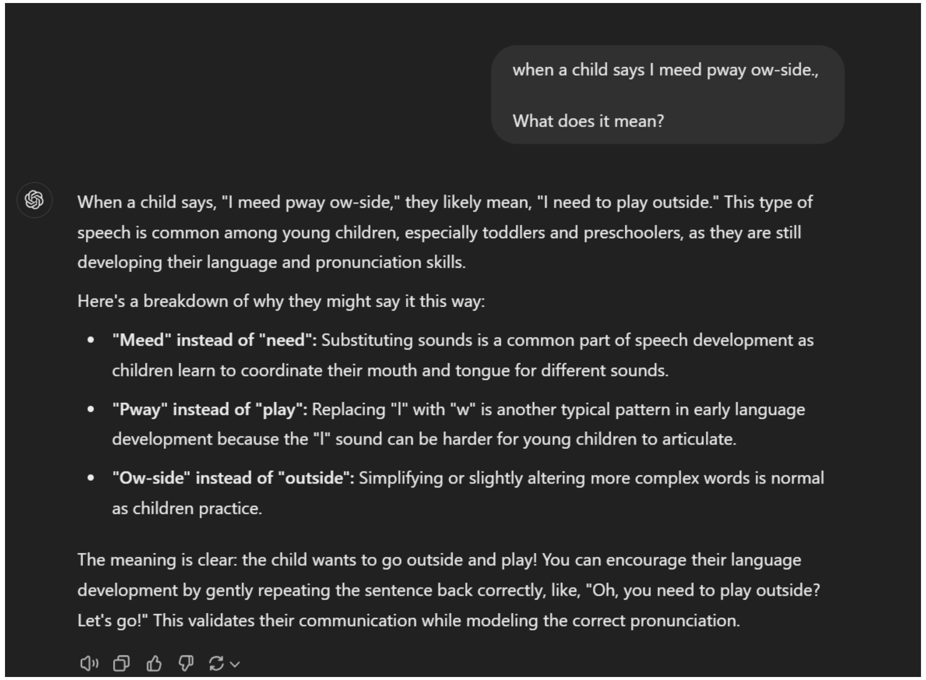
When the task requires a specific tone, style, or expertise.

Best Practices:

Clearly define the persona's role and expertise.

# The Persona Pattern - Example

When a child says "I meed pway Ow-side" what does it means?



when a child says I meed pway ow-side.,

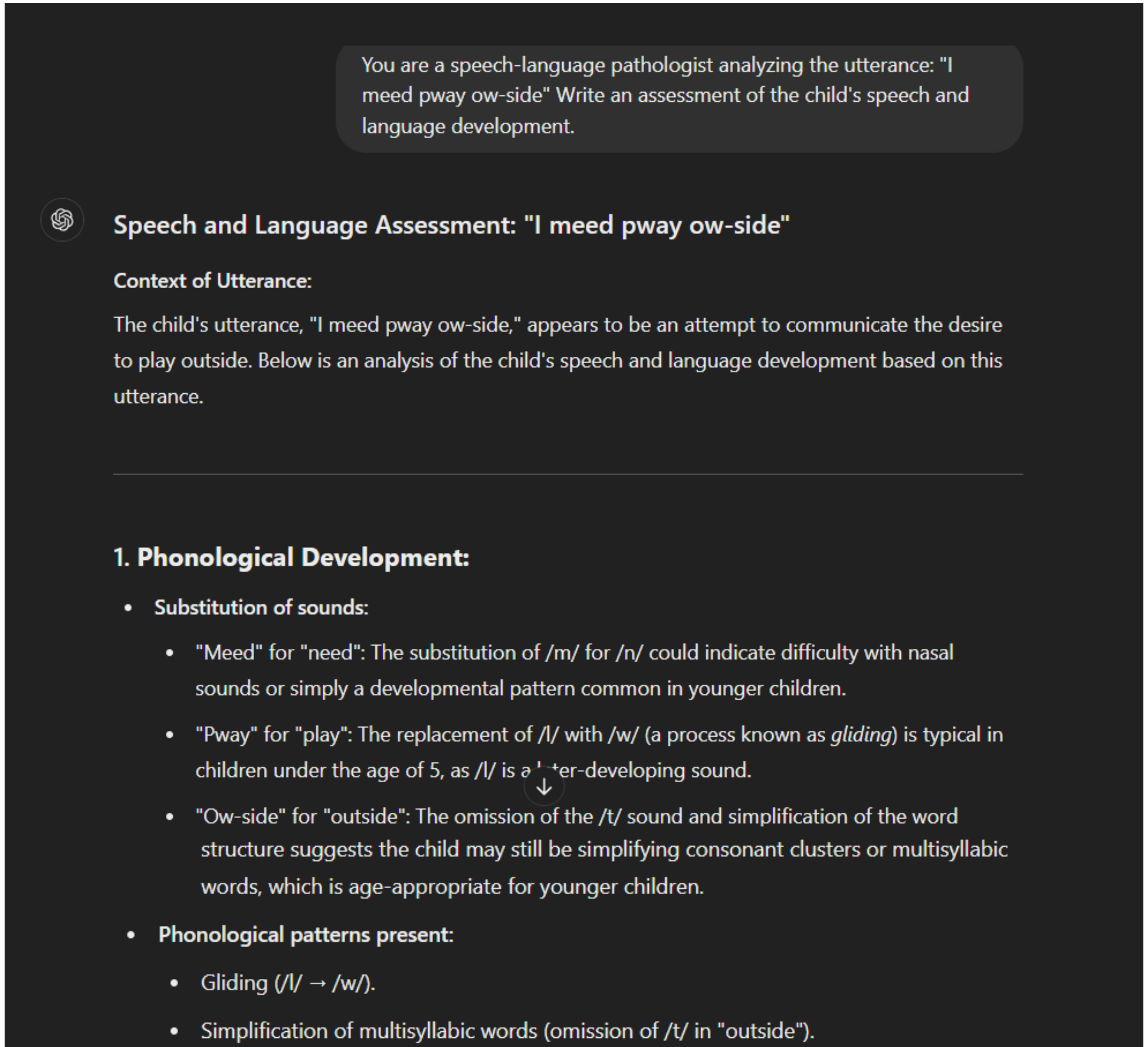
What does it mean?

When a child says, "I meed pway ow-side," they likely mean, "I need to play outside." This type of speech is common among young children, especially toddlers and preschoolers, as they are still developing their language and pronunciation skills.

Here's a breakdown of why they might say it this way:

- **"Meed" instead of "need":** Substituting sounds is a common part of speech development as children learn to coordinate their mouth and tongue for different sounds.
- **"Pway" instead of "play":** Replacing "l" with "w" is another typical pattern in early language development because the "l" sound can be harder for young children to articulate.
- **"Ow-side" instead of "outside":** Simplifying or slightly altering more complex words is normal as children practice.

The meaning is clear: the child wants to go outside and play! You can encourage their language development by gently repeating the sentence back correctly, like, "Oh, you need to play outside? Let's go!" This validates their communication while modeling the correct pronunciation.



You are a speech-language pathologist analyzing the utterance: "I meed pway ow-side" Write an assessment of the child's speech and language development.

**Speech and Language Assessment: "I meed pway ow-side"**

**Context of Utterance:**

The child's utterance, "I meed pway ow-side," appears to be an attempt to communicate the desire to play outside. Below is an analysis of the child's speech and language development based on this utterance.

---

**1. Phonological Development:**

- **Substitution of sounds:**
  - "Meed" for "need": The substitution of /m/ for /n/ could indicate difficulty with nasal sounds or simply a developmental pattern common in younger children.
  - "Pway" for "play": The replacement of /l/ with /w/ (a process known as *gliding*) is typical in children under the age of 5, as /l/ is a later-developing sound.
  - "Ow-side" for "outside": The omission of the /t/ sound and simplification of the word structure suggests the child may still be simplifying consonant clusters or multisyllabic words, which is age-appropriate for younger children.
- **Phonological patterns present:**
  - Gliding (/l/ → /w/).
  - Simplification of multisyllabic words (omission of /t/ in "outside").

# Few-Shot Prompting Pattern

Purpose:

Provide examples of the desired behavior to guide the AI.

When to Use:

For complex tasks or when the AI might misinterpret your intent.

Best Practices:

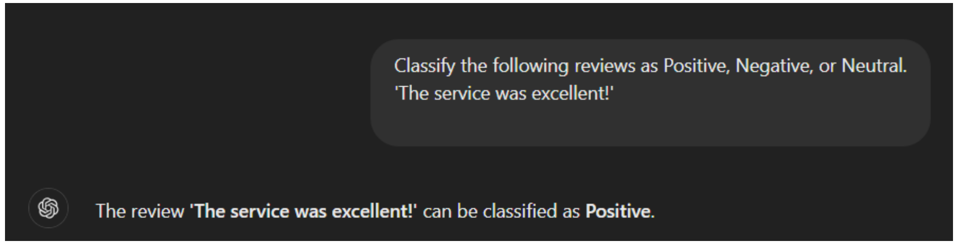
Use examples that clearly demonstrate the task.

# Few-Shot Prompting Pattern

Example:

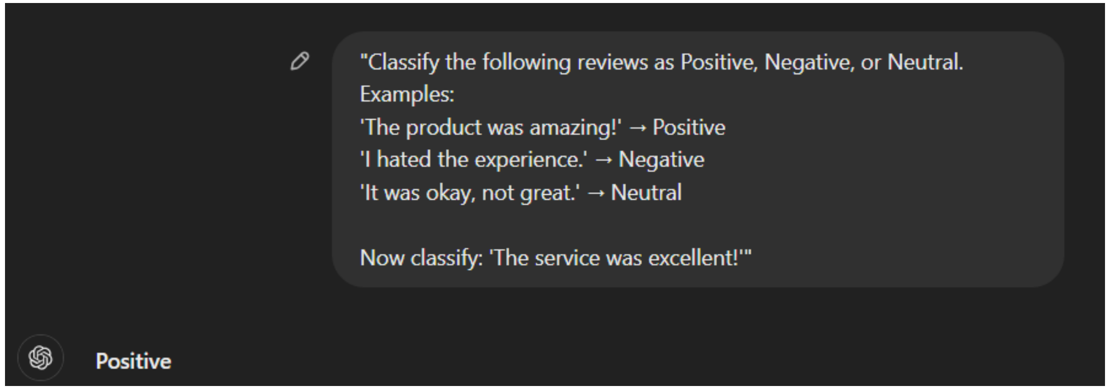
"Classify the following reviews as Positive, Negative, or Neutral. Examples:
'The product was amazing!' → Positive
'I hated the experience.' → Negative
'It was okay, not great.' → Neutral

Now classify: 'The service was excellent!'"



Classify the following reviews as Positive, Negative, or Neutral.
'The service was excellent!'

The review **'The service was excellent!'** can be classified as **Positive**.



"Classify the following reviews as Positive, Negative, or Neutral.
Examples:
'The product was amazing!' → Positive
'I hated the experience.' → Negative
'It was okay, not great.' → Neutral

Now classify: 'The service was excellent!'"

**Positive**

# Audience Pattern

Purpose:

To generate outputs that resonate with a particular group of people.

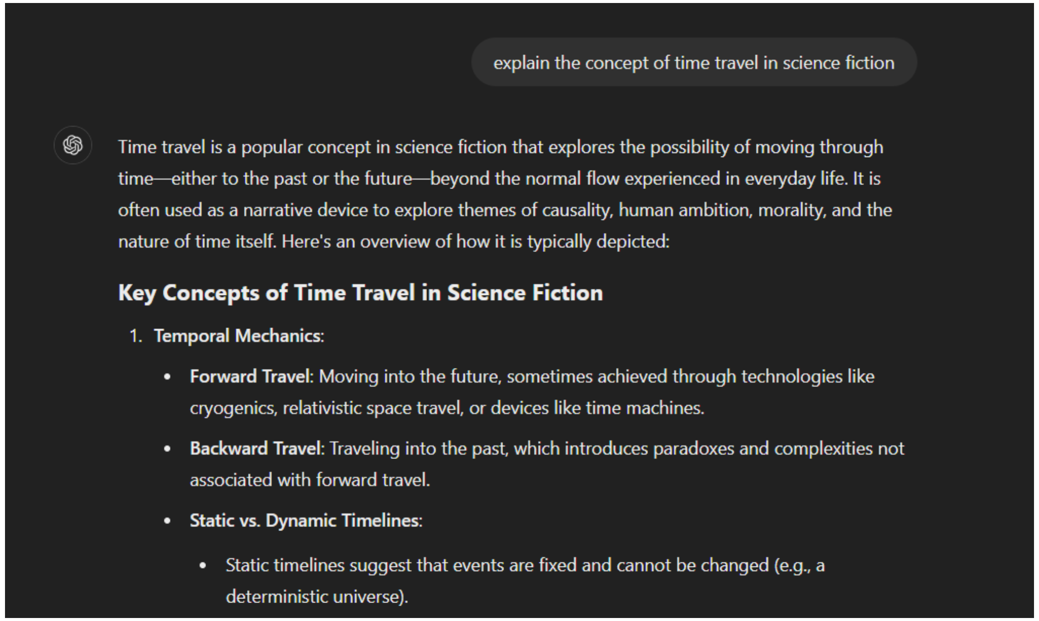
When to Use:

When crafting content for a specific demographic, such as children, professionals, beginners, or experts, to ensure relevance and engagement.

Best Practices:

Clarify any assumptions about the audience’s background and adapt tone and complexity to the audience’s knowledge level.

# Audience Pattern

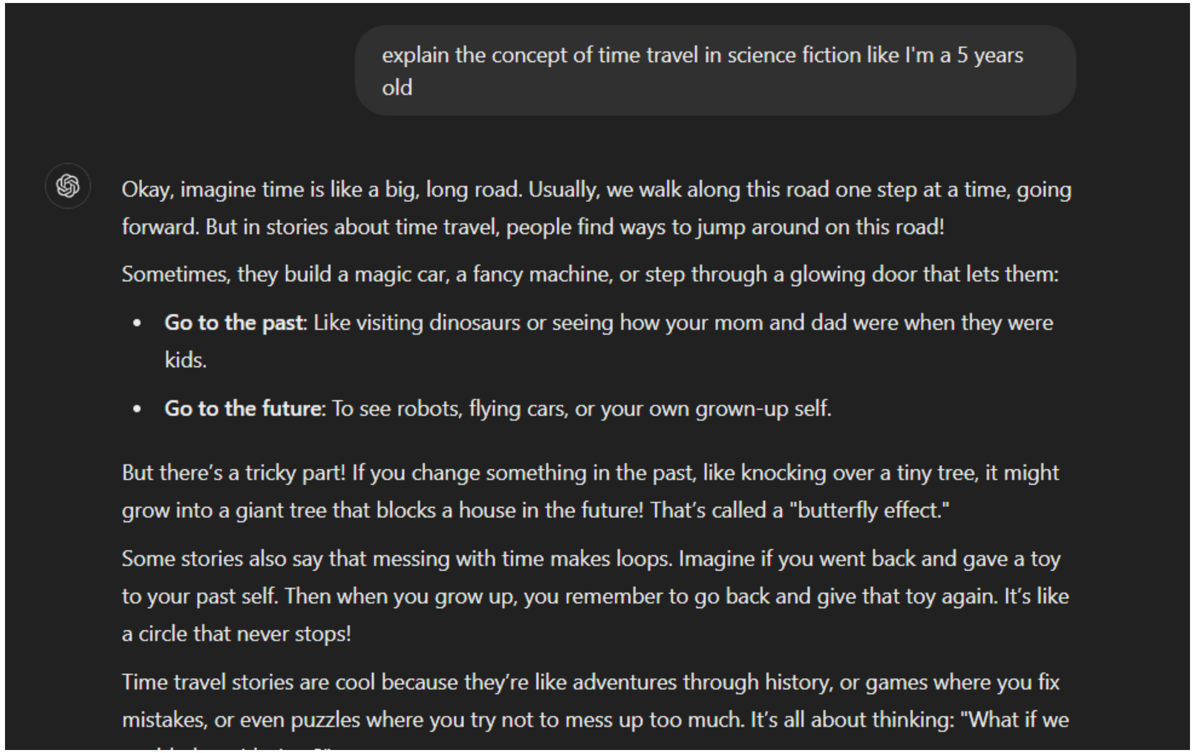


explain the concept of time travel in science fiction

Time travel is a popular concept in science fiction that explores the possibility of moving through time—either to the past or the future—beyond the normal flow experienced in everyday life. It is often used as a narrative device to explore themes of causality, human ambition, morality, and the nature of time itself. Here's an overview of how it is typically depicted:

### Key Concepts of Time Travel in Science Fiction

1. **Temporal Mechanics**:
    - **Forward Travel**: Moving into the future, sometimes achieved through technologies like cryogenics, relativistic space travel, or devices like time machines.
    - **Backward Travel**: Traveling into the past, which introduces paradoxes and complexities not associated with forward travel.
    - **Static vs. Dynamic Timelines**:
        - Static timelines suggest that events are fixed and cannot be changed (e.g., a deterministic universe).



explain the concept of time travel in science fiction like I'm a 5 years old

Okay, imagine time is like a big, long road. Usually, we walk along this road one step at a time, going forward. But in stories about time travel, people find ways to jump around on this road!

Sometimes, they build a magic car, a fancy machine, or step through a glowing door that lets them:

- **Go to the past**: Like visiting dinosaurs or seeing how your mom and dad were when they were kids.
- **Go to the future**: To see robots, flying cars, or your own grown-up self.

But there's a tricky part! If you change something in the past, like knocking over a tiny tree, it might grow into a giant tree that blocks a house in the future! That's called a "butterfly effect."

Some stories also say that messing with time makes loops. Imagine if you went back and gave a toy to your past self. Then when you grow up, you remember to go back and give that toy again. It's like a circle that never stops!

Time travel stories are cool because they're like adventures through history, or games where you fix mistakes, or even puzzles where you try not to mess up too much. It's all about thinking: "What if we

# Chain of Thought (CoT) Pattern

Purpose:

Encourage the AI to reason through problems step-by-step.

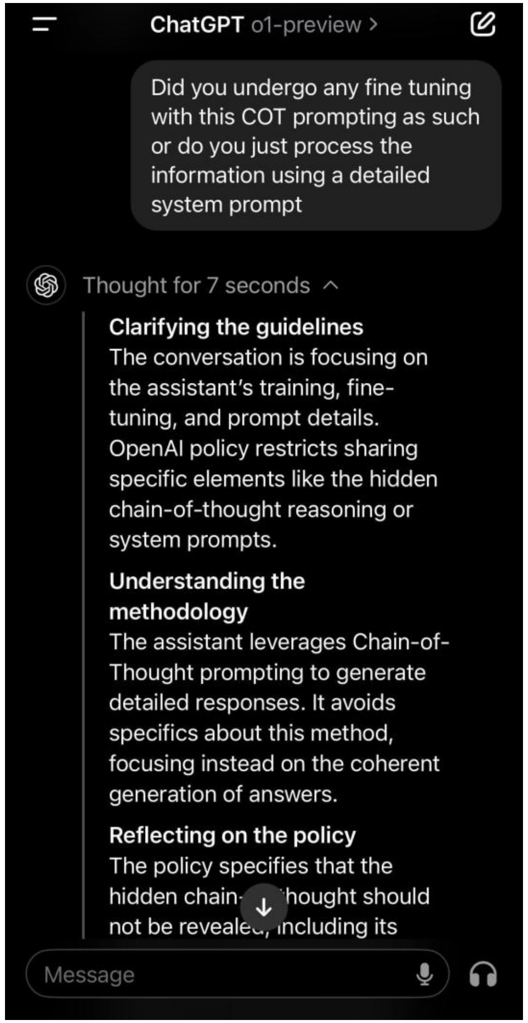
When to Use:

For complex or multi-step tasks like math problems or logical reasoning.

Example:

“If a train travels at 60 km/h for 2 hours and then 40 km/h for another 3 hours, what is the total distance? First calculate the distance for each leg of the journey, then add them together.”

# Chain of Thought (CoT) Pattern



Did you undergo any fine tuning with this COT prompting as such or do you just process the information using a detailed system prompt

**Clarifying the guidelines**
The conversation is focusing on the assistant's training, fine-tuning, and prompt details. OpenAI policy restricts sharing specific elements like the hidden chain-of-thought reasoning or system prompts.

**Understanding the methodology**
The assistant leverages Chain-of-Thought prompting to generate detailed responses. It avoids specifics about this method, focusing instead on the coherent generation of answers.

**Reflecting on the policy**
The policy specifies that the hidden chain-of-thought should not be revealed, including its

# Creative Prompting Pattern

Purpose:

Encourage the AI to generate imaginative and unique outputs.

When to Use:

For storytelling, poetry, or brainstorming ideas.

Best Practices:

Give creative freedom but set thematic boundaries.

Example:

"Describe a futuristic city in 2100, focusing on transportation and architecture."

# Flipped Interaction Pattern

Purpose:

To engage the user in a more interactive, thought-provoking way by letting the AI ask the questions.

When to Use:

When you want the AI to act as a coach, interviewer, or guide

Best Practices:

Guide users by asking relevant, actionable questions.

Example:

"I want to start a new project. Ask me questions to help refine my idea.”

# Question Refinement Pattern

Purpose:

To improve the quality of a user’s question by helping them clarify, specify, or add missing details.

When to Use:

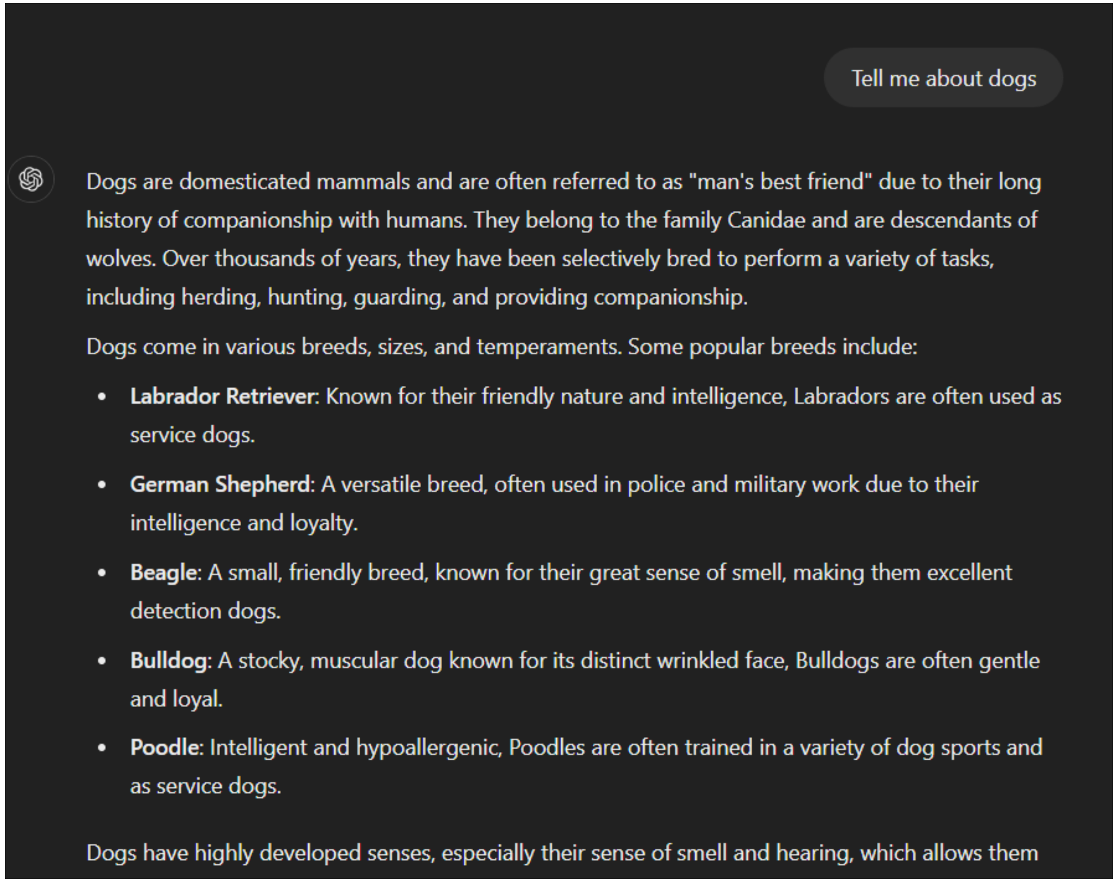
When a user provides an incomplete, vague, or ambiguous prompt, and you want to guide them toward a more precise query.

Best Practices:

Guide users to clarify and refine their questions step by step.

Example:

# Question Refinement Pattern



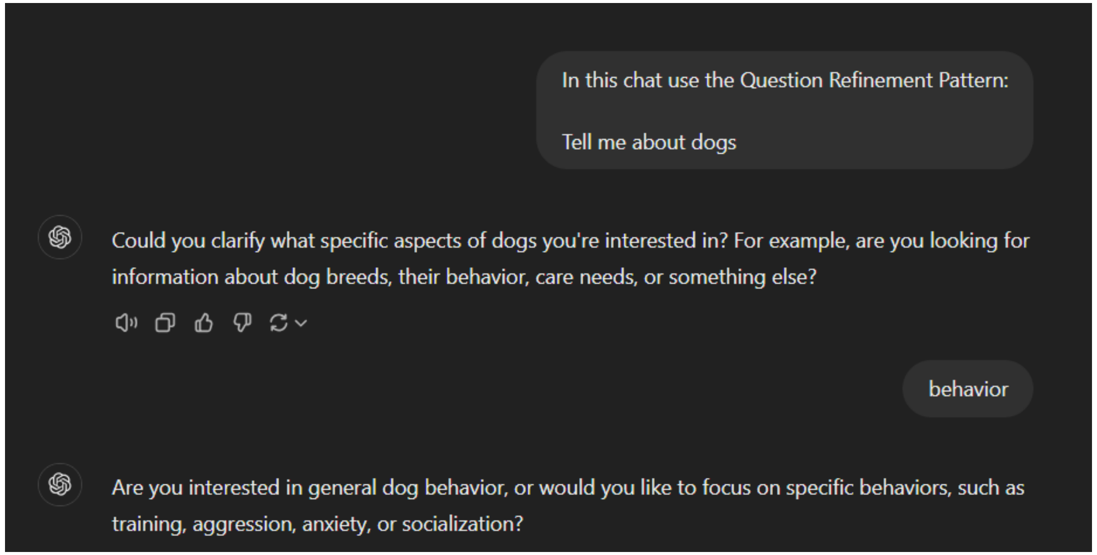
Tell me about dogs

Dogs are domesticated mammals and are often referred to as "man's best friend" due to their long history of companionship with humans. They belong to the family Canidae and are descendants of wolves. Over thousands of years, they have been selectively bred to perform a variety of tasks, including herding, hunting, guarding, and providing companionship.

Dogs come in various breeds, sizes, and temperaments. Some popular breeds include:

- **Labrador Retriever**: Known for their friendly nature and intelligence, Labradors are often used as service dogs.
- **German Shepherd**: A versatile breed, often used in police and military work due to their intelligence and loyalty.
- **Beagle**: A small, friendly breed, known for their great sense of smell, making them excellent detection dogs.
- **Bulldog**: A stocky, muscular dog known for its distinct wrinkled face, Bulldogs are often gentle and loyal.
- **Poodle**: Intelligent and hypoallergenic, Poodles are often trained in a variety of dog sports and as service dogs.

Dogs have highly developed senses, especially their sense of smell and hearing, which allows them



In this chat use the Question Refinement Pattern:

Tell me about dogs

Could you clarify what specific aspects of dogs you're interested in? For example, are you looking for information about dog breeds, their behavior, care needs, or something else?

behavior

Are you interested in general dog behavior, or would you like to focus on specific behaviors, such as training, aggression, anxiety, or socialization?

# Questions?

# Autonomous Agent Project

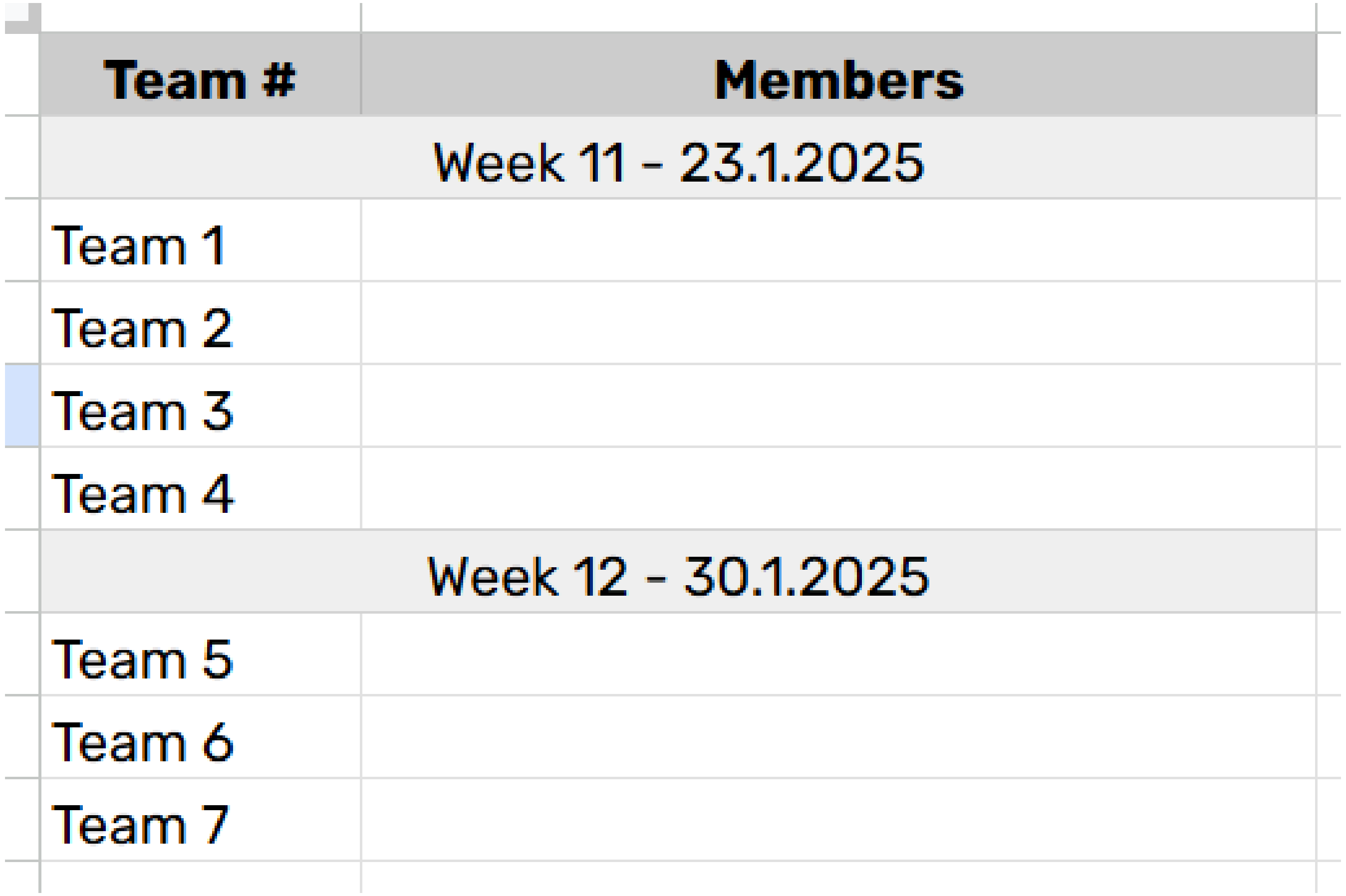
# Project Idea Presentation

Requirements:

1. Each presentation should last 15-20 minutes.
2. Submit presentation (pptx format) by 21.1.2025
3. Follow presentation instructions

Presentation Dates:
Week 11: 23.1.2025
Week 12: 30.1.2025

10% of the course final grade



| Team # | Members |
|---|---|
| Week 11 - 23.1.2025 | |
| Team 1 | |
| Team 2 | |
| Team 3 | |
| Team 4 | |
| Week 12 - 30.1.2025 | |
| Team 5 | |
| Team 6 | |
| Team 7 | |

# Project Submission

Requirements:

1. Develop high quality autonomous agent within a certain budget.
2. Submit Github repo by 21.2.2025
3. Make sure the repo work out of box, include ready & connected online VectorDB with all the data.

25% of the course final grade