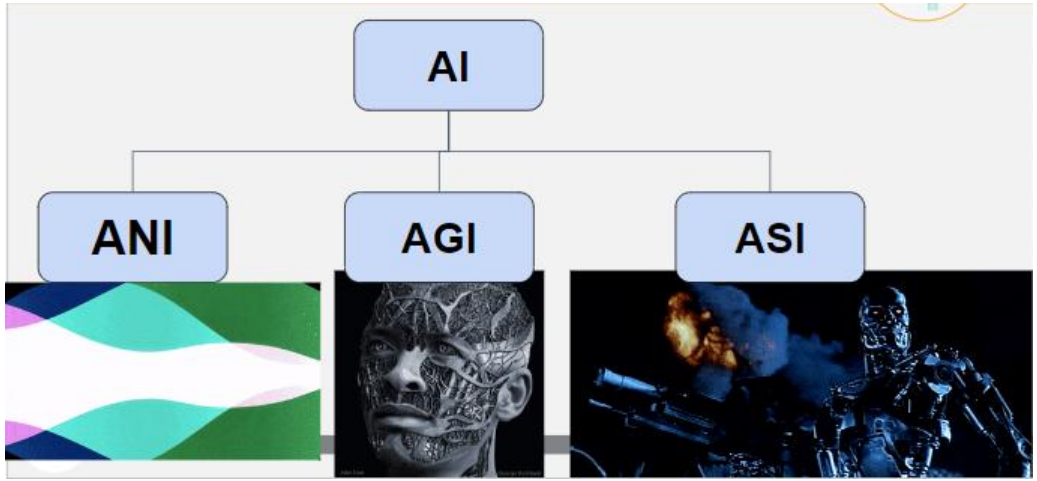
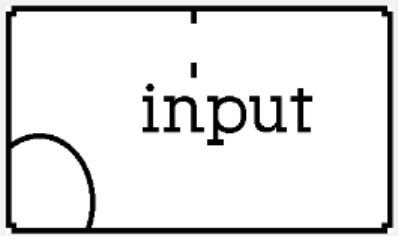
OpenAI
With ChatGPT 5

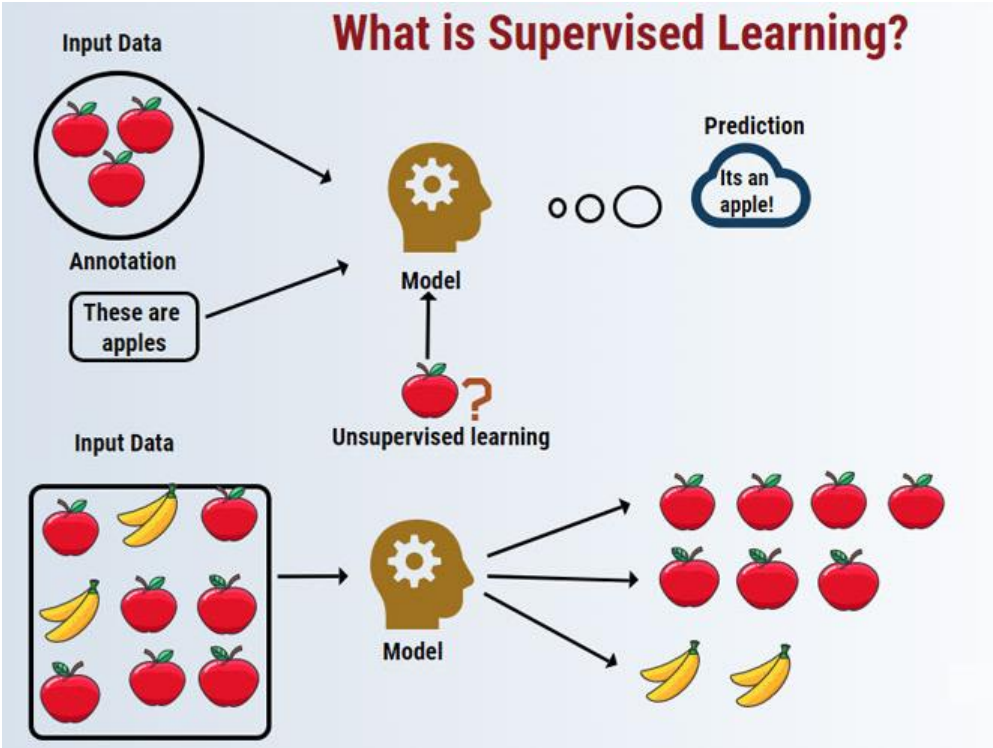
Dr. Ernesto Lee
Professor Carlos Marquez


AI
ANI
AGI
ASI

A → B

Input → Output




What is Supervised Learning?
Input Data
Annotation
These are apples
Model
Prediction
Its an apple!
Unsupervised learning
Input Data
Model

What animal is this?

How about this?

DNA


GCTGATTGG
GTATGCGCA
CTCTTAG
CTAGAGAAAT
ATGTCAGC
AGCTCAG
GCACCAAC
CCACTCAGG
ATCTGGC
CAGCCCCTCG
TCTG
TTCCCCTTT
GCCATGG
TAGTCCTCAA
CAACT
CGGCACGT
ATGGGG
GGTTACGTAG
CTCCTGT
CTAGC

CAVEMAN

What you know

- Temperature (78 degrees)
- Irrigation Levels (4” per mo)
- Fertilizer (88 tons/acre)
- Bees (2000 bees per acre)

What you WANT to know

INPUTS / DATA YOU HAVE

78 degrees (Temp)

4” (Irrigation)

88 (Fertilizer Score)

2,000 b/a (Bee’s per ac

Tree Health

????

????

????

Orange Harvest

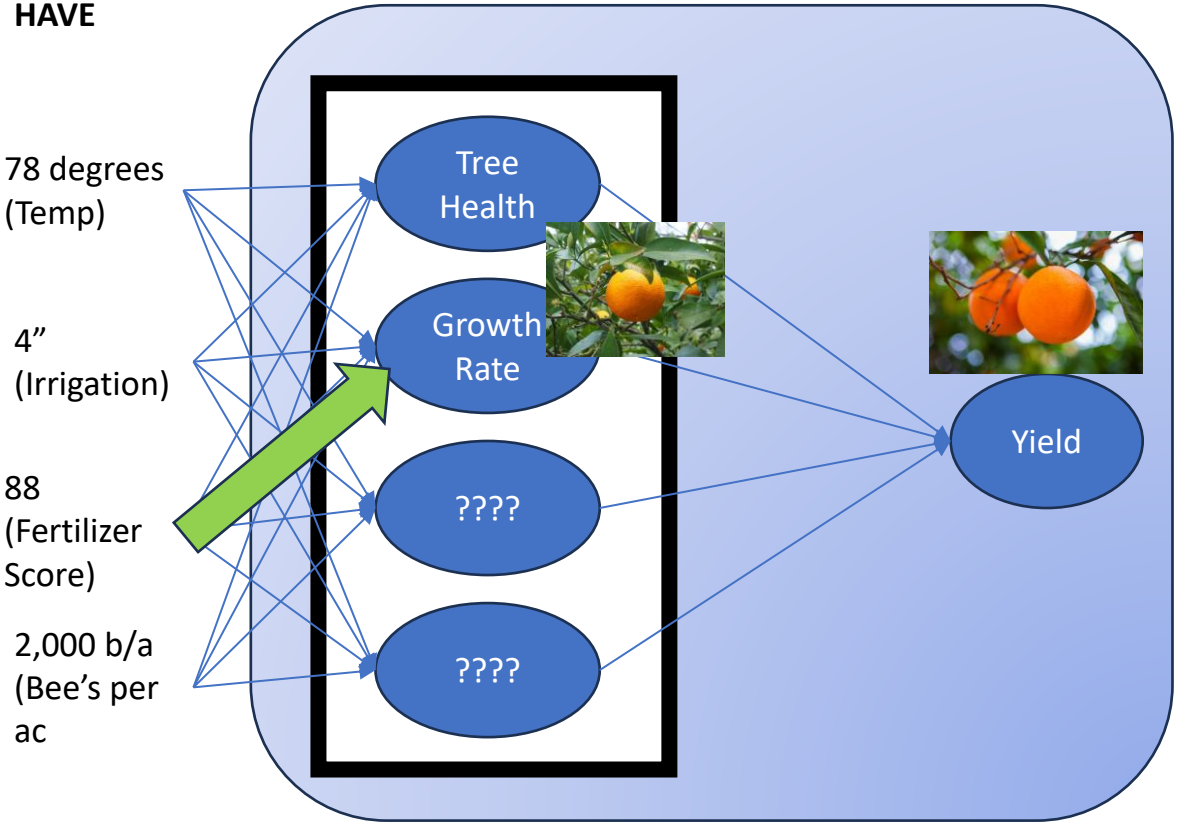
INPUTS / DATA YOU HAVE

78 degrees (Temp)

4” (Irrigation)

88 (Fertilizer Score)

2,000 b/a (Bee’s per ac



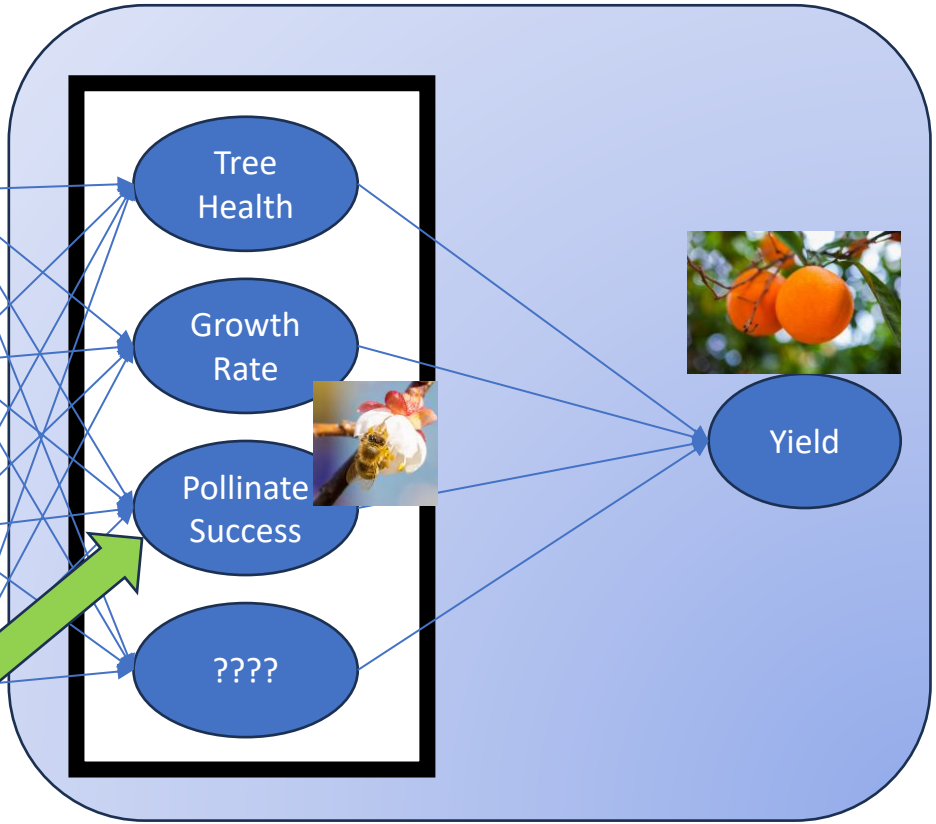
INPUTS / DATA YOU HAVE

78 degrees (Temp)

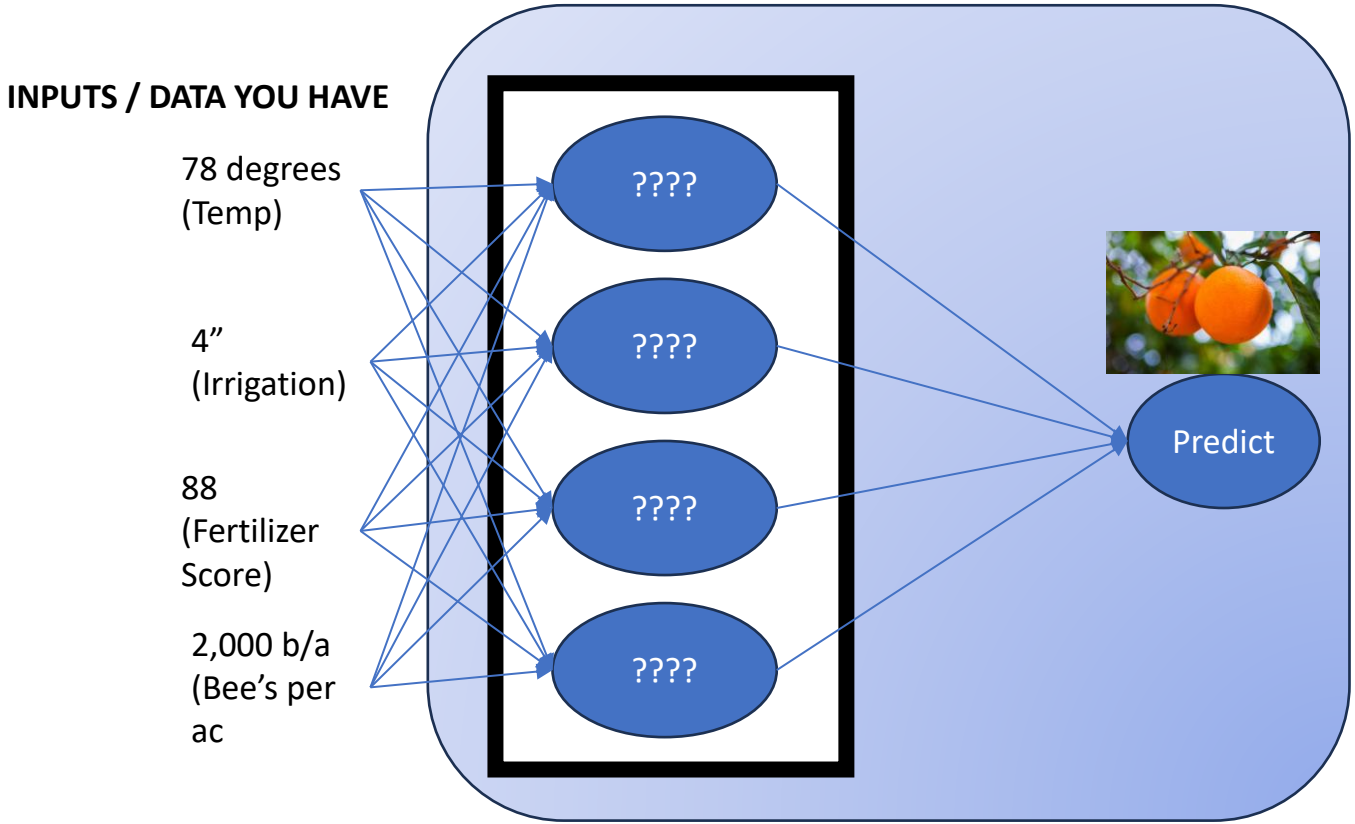
4” (Irrigation)

88 (Fertilizer Score)

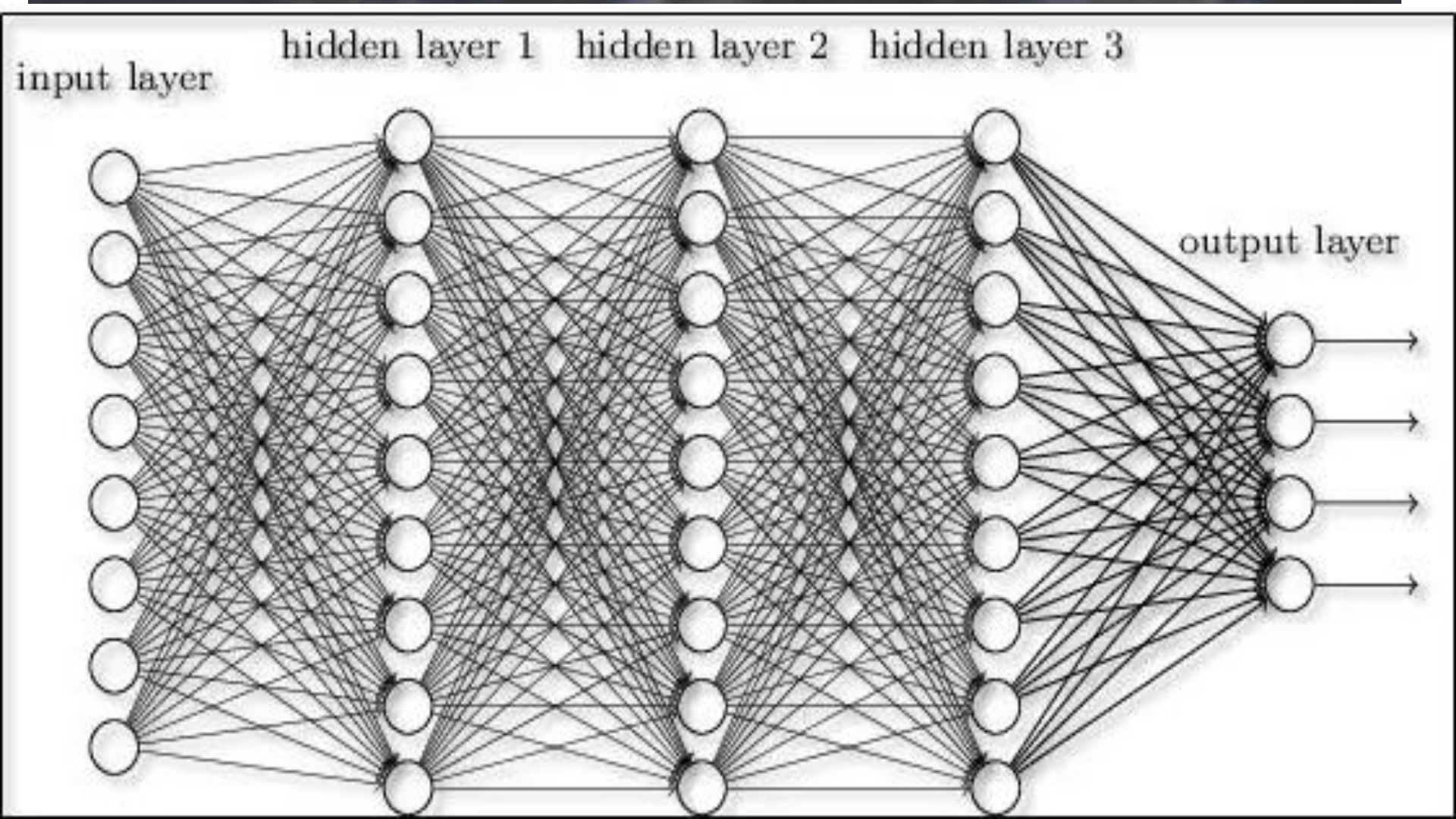
2,000 b/a (Bee’s per acre)



Here is the crazy part…

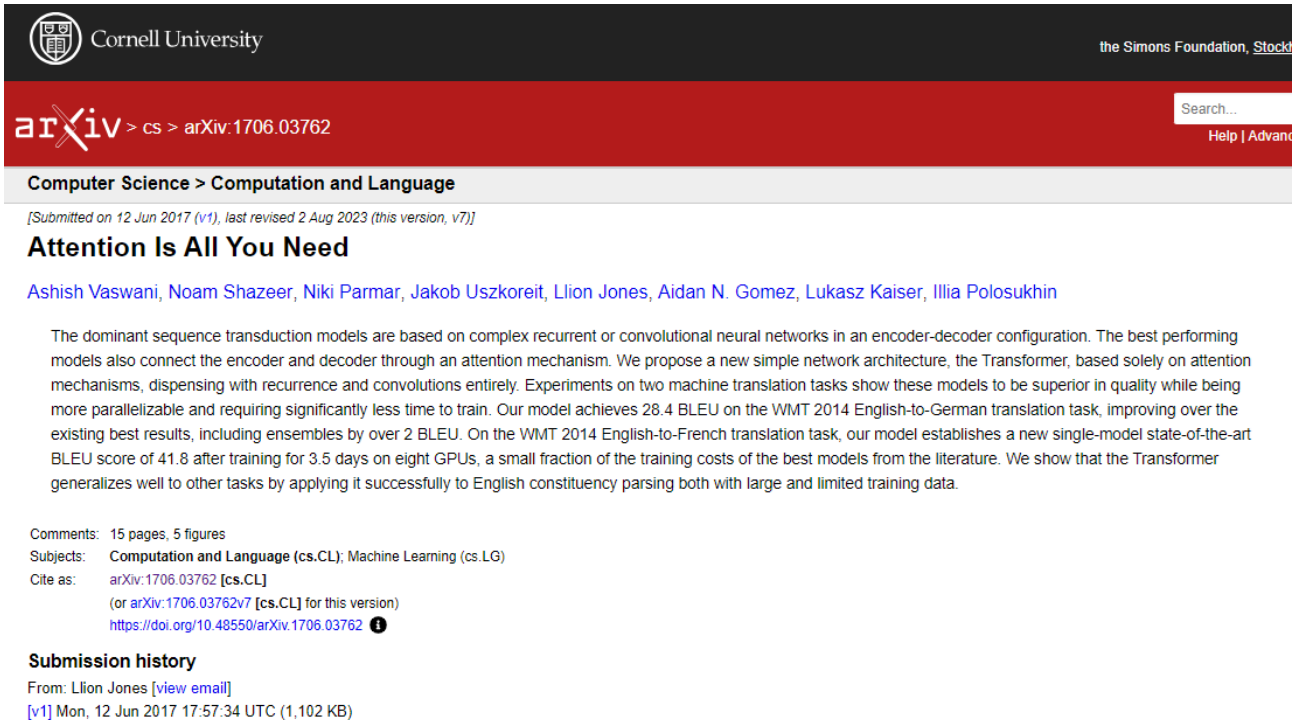


Every Board Member Contributes to the Decision


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer

https://bit.ly/attn2121

- https://arxiv.org/abs/1706.03762



Cornell University

the Simons Foundation, Stock

arXiv > cs > arXiv:1706.03762

Search...

Help | Advanc

Computer Science > Computation and Language

[Submitted on 12 Jun 2017 (v1), last revised 2 Aug 2023 (this version, v7)]

Attention Is All You Need

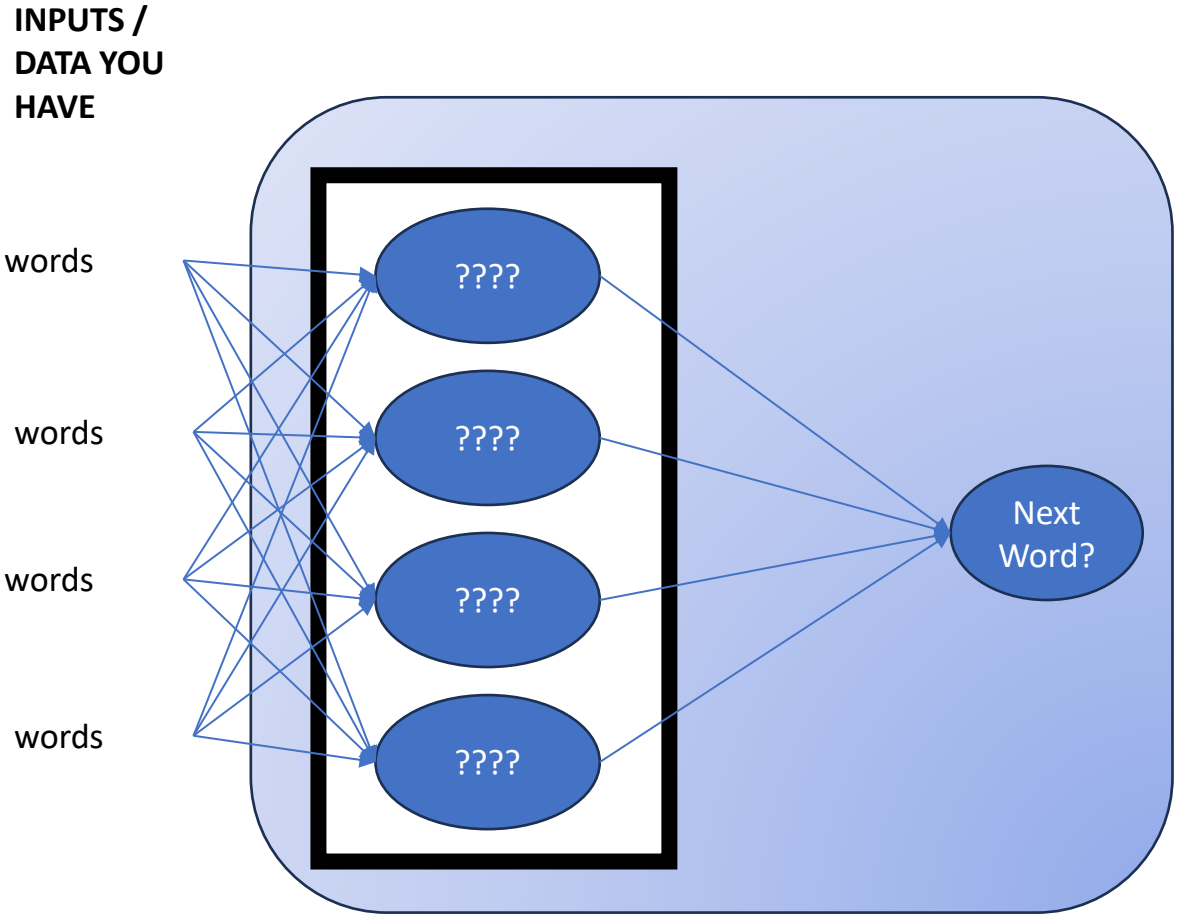
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

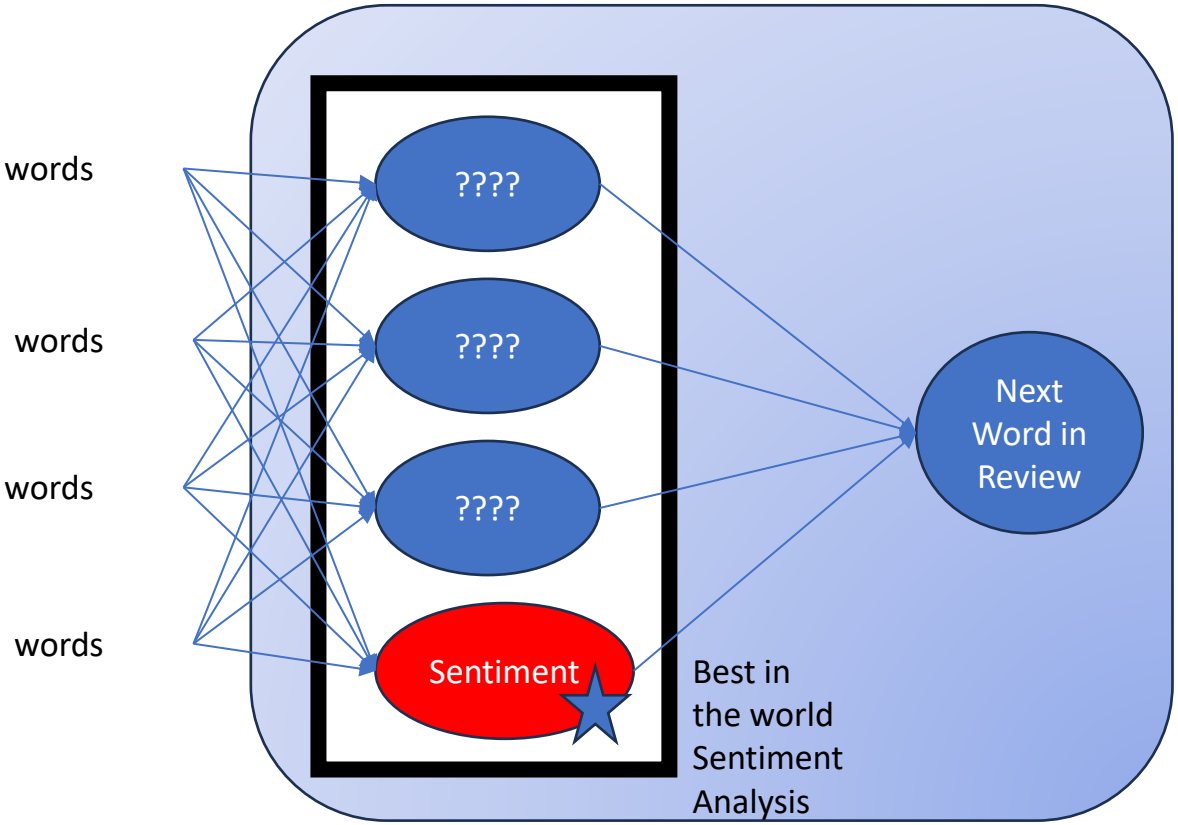
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

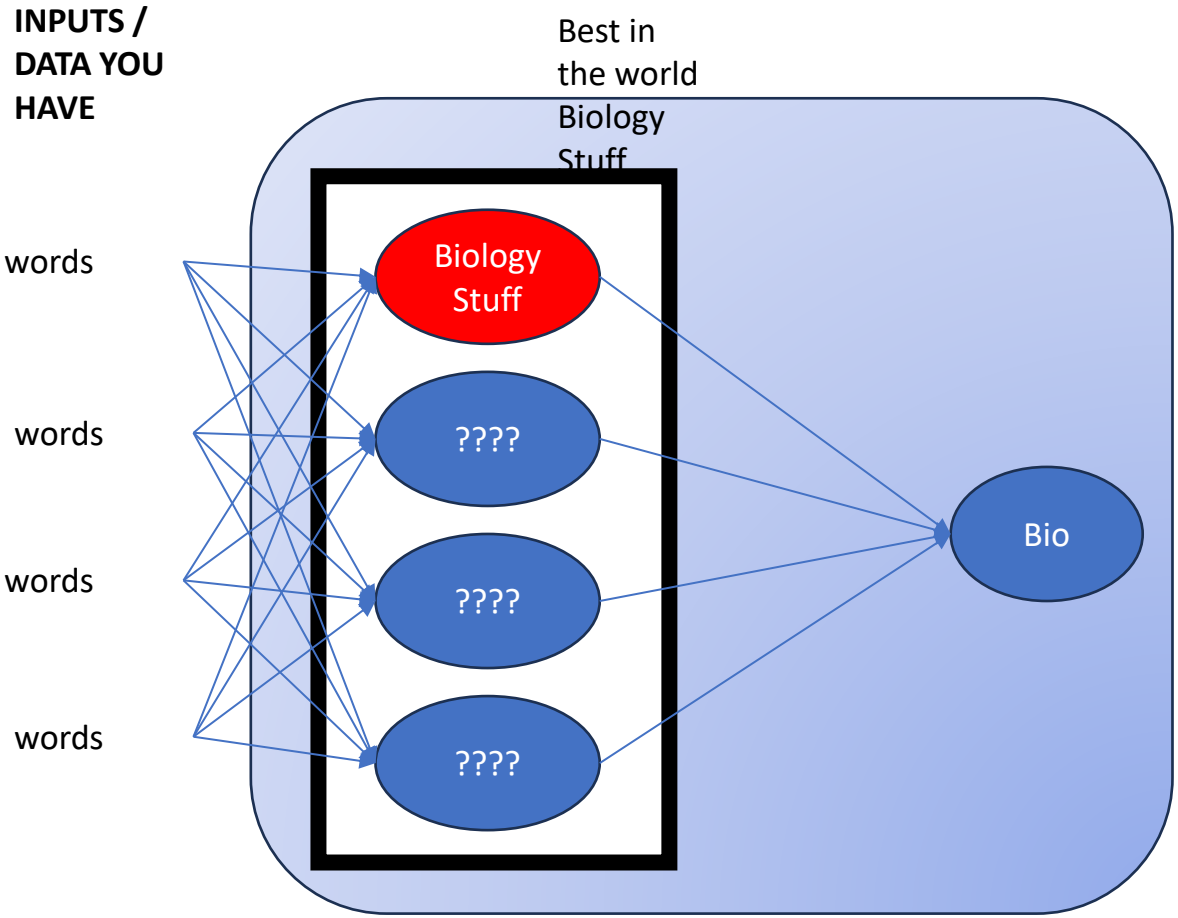
Comments: 15 pages, 5 figures
Subjects: **Computation and Language (cs.CL)**; Machine Learning (cs.LG)
Cite as: arXiv:1706.03762 **[cs.CL]**
(or arXiv:1706.03762v7 **[cs.CL]** for this version)
https://doi.org/10.48550/arXiv.1706.03762

Submission history

From: Llion Jones [view email]
[v1] Mon, 12 Jun 2017 17:57:34 UTC (1,102 KB)

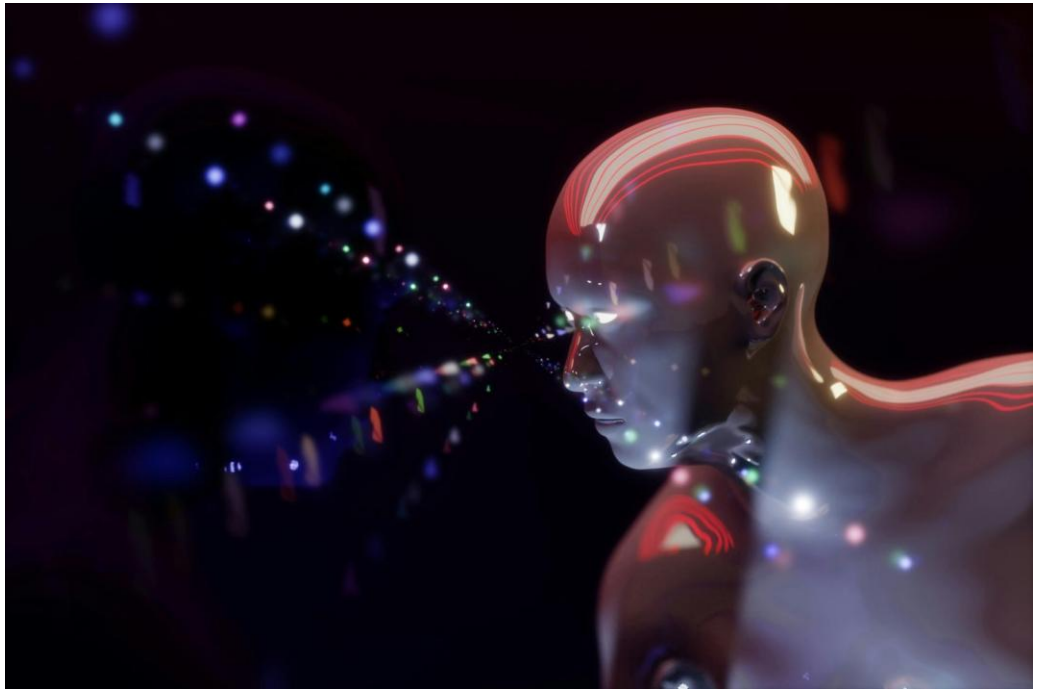

INPUTS / DATA YOU HAVE
words
words
words
words
????
????
????
????
Next Word?

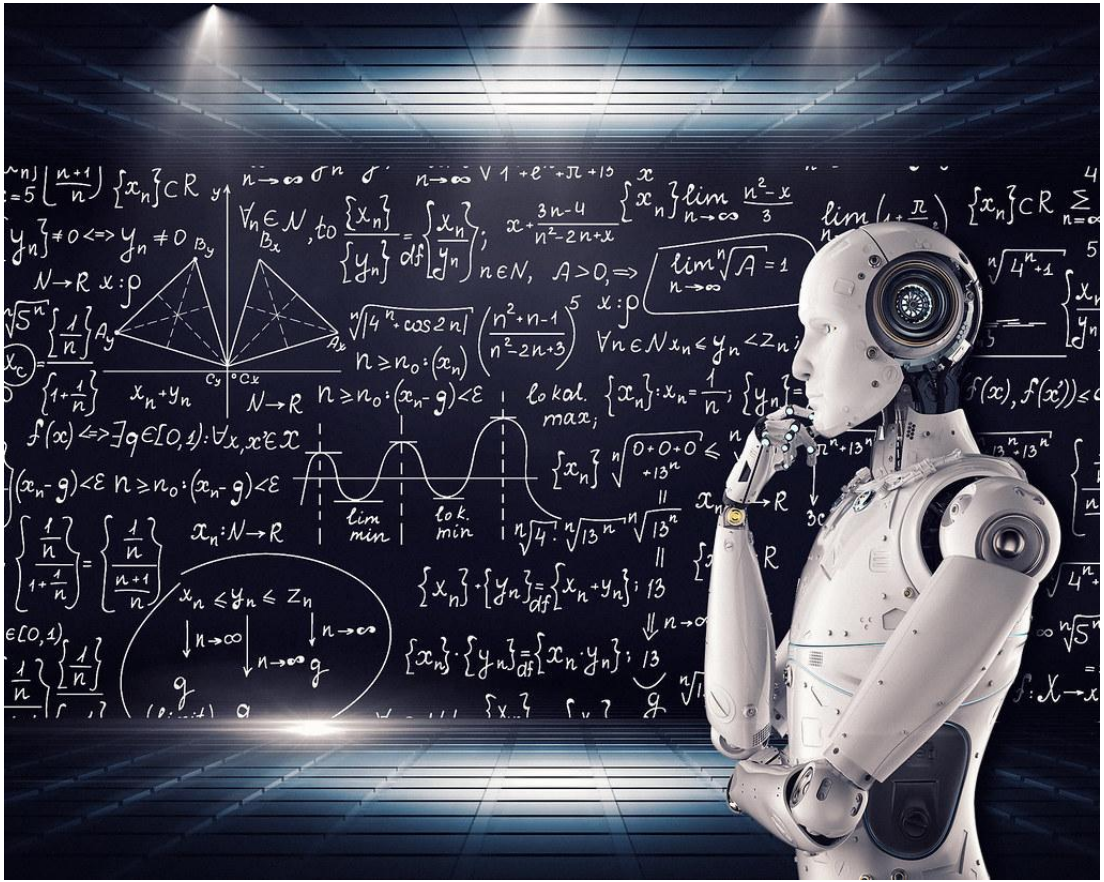

words
words
words
words
????
????
????
Sentiment
Next Word in Review
Best in the world Sentiment Analysis


INPUTS / DATA YOU HAVE
Best in the world Biology Stuff
words
words
words
words
Biology Stuff
????
????
????
Bio


ChatGPT

Language as the Shadow of Reality





What is the OpenAI API?

WORKING WITH THE OPENAI API

Course goals

- Use the *OpenAI API* to access AI models
- Use those models to solve various tasks
- Use **Python** code to do this!

Expected knowledge

- Subsetting Python lists and **dictionaries**
- Control flow → `if`, `elif`, `else`
- Loops → `for`, `while`

Not expected to know

- AI or machine learning

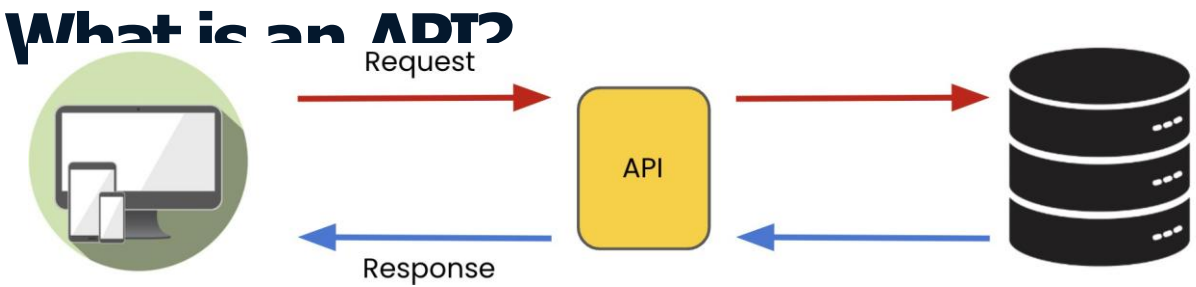
- **OpenAI** = AI R&D company
- **ChatGPT** = AI application
- **OpenAI API** = Interface for accessing OpenAI models

[1] Image Credit: OpenAI

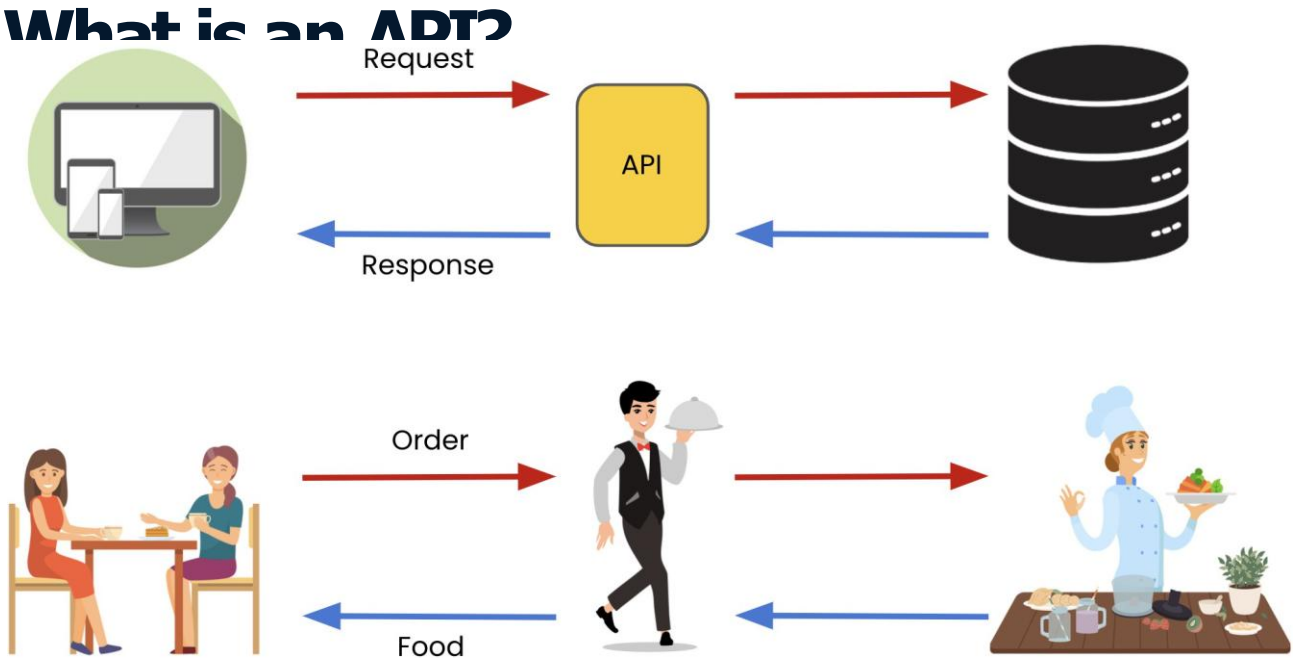
- **OpenAI** = AI R&D company
- **ChatGPT** = AI application
- **OpenAI API** = Interface for accessing OpenAI models

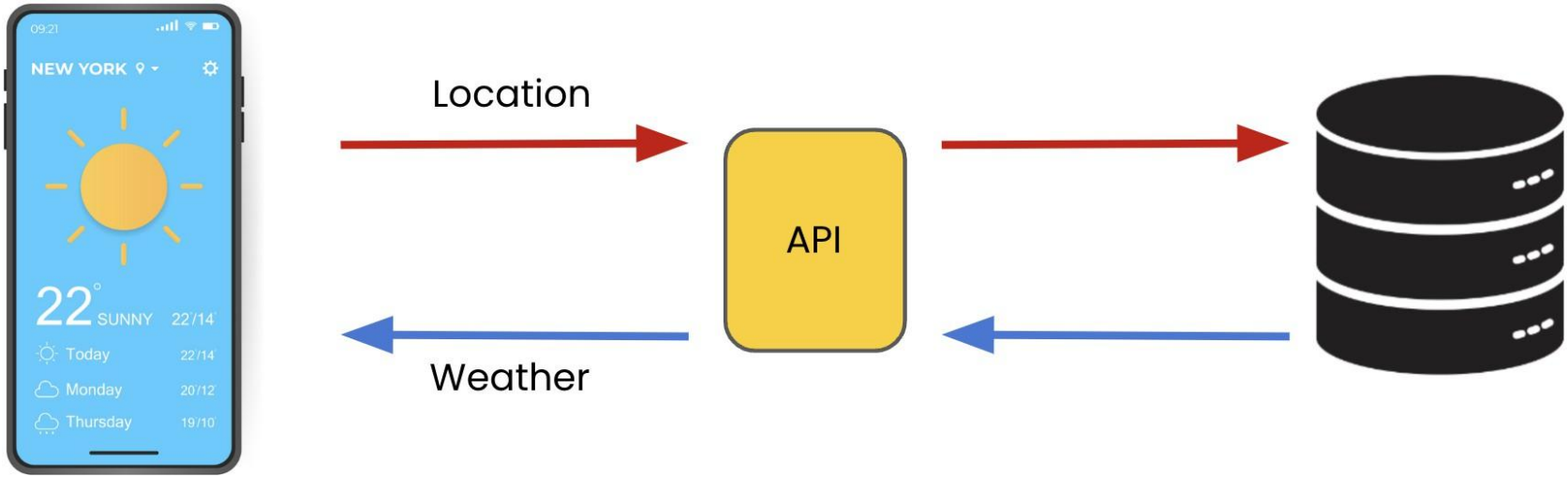
- **OpenAI** = AI R&D company
- **ChatGPT** = AI application
- **OpenAI API** = Interface for accessing OpenAI models

- **A**pplication **P**rogramming **I**nterface

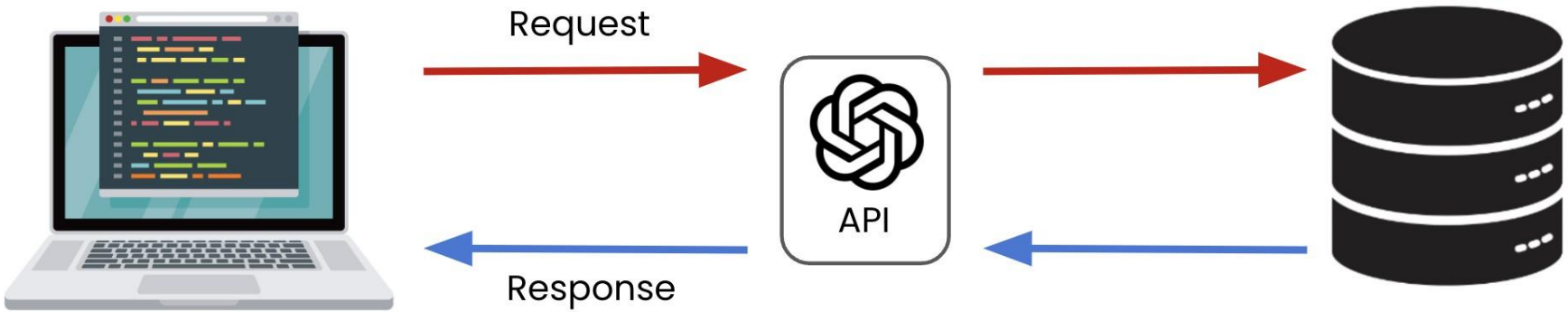


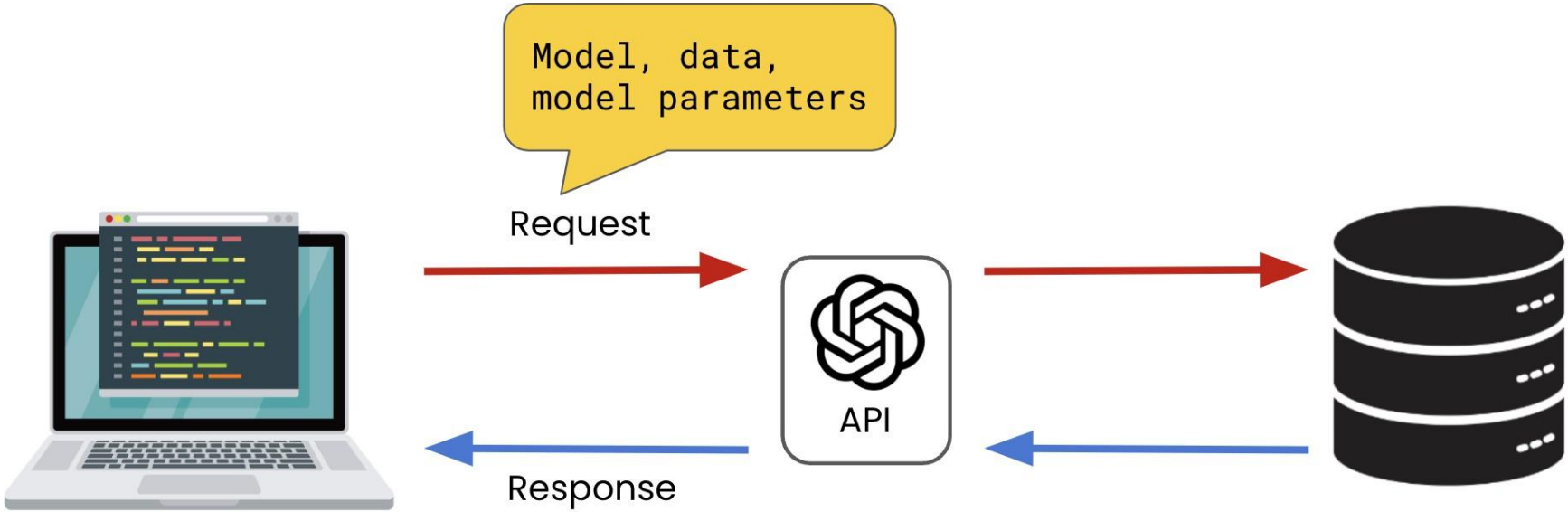
- **A**pplication **P**rogramming **I**nterface



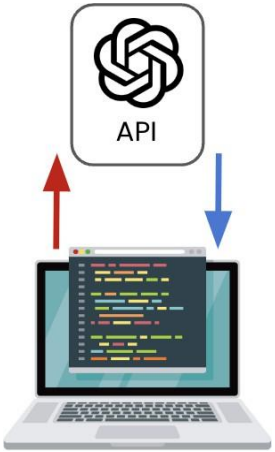

09:21
NEW YORK
22°
SUNNY
22/14
Today
22/14
Monday
20/12
Thursday
19/10
Location
API
Weather

The OpenAI API

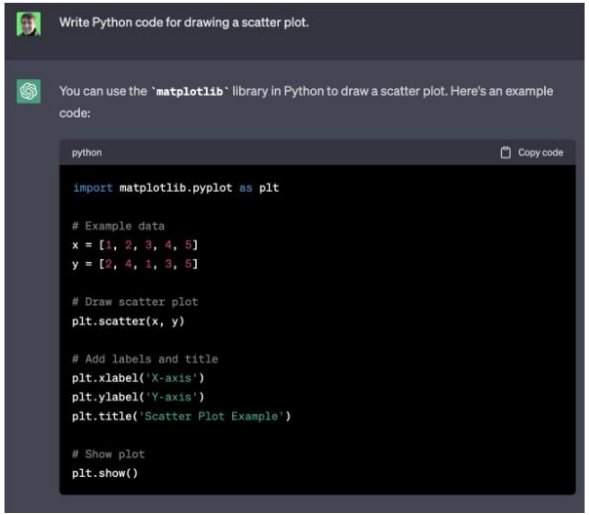



Model, data,
model parameters
Request
API
Response

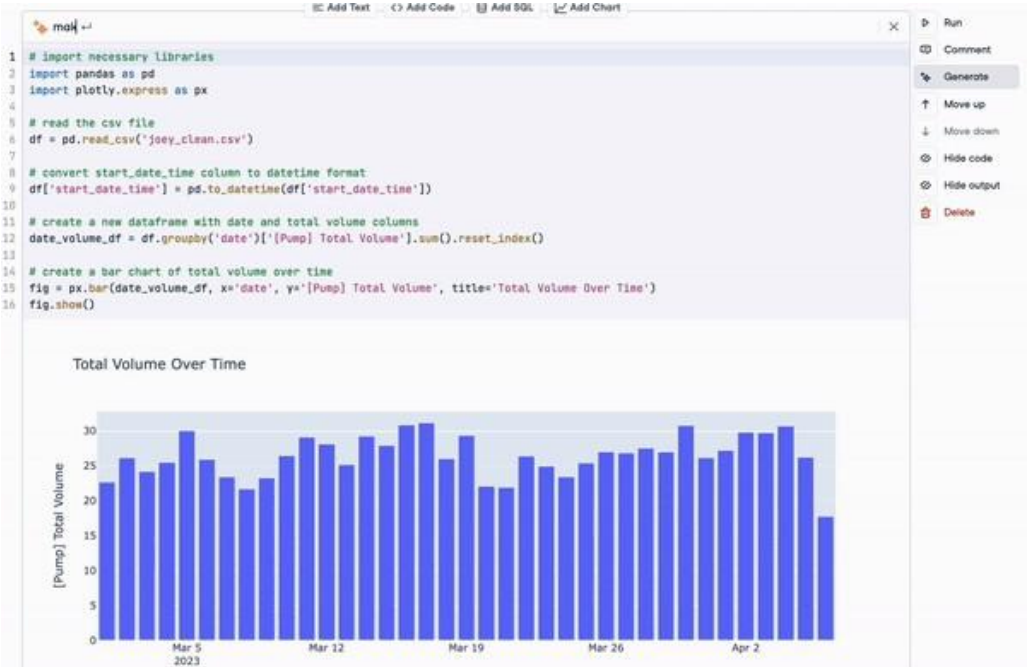
- Provides *flexibility* for integrating AI into products, experiences, and processes
- Interact via programming language



- Very little setup
- Sufficient for *streamlining* individual's workflows



DataLab = Cloud-based, AI-powered IDE



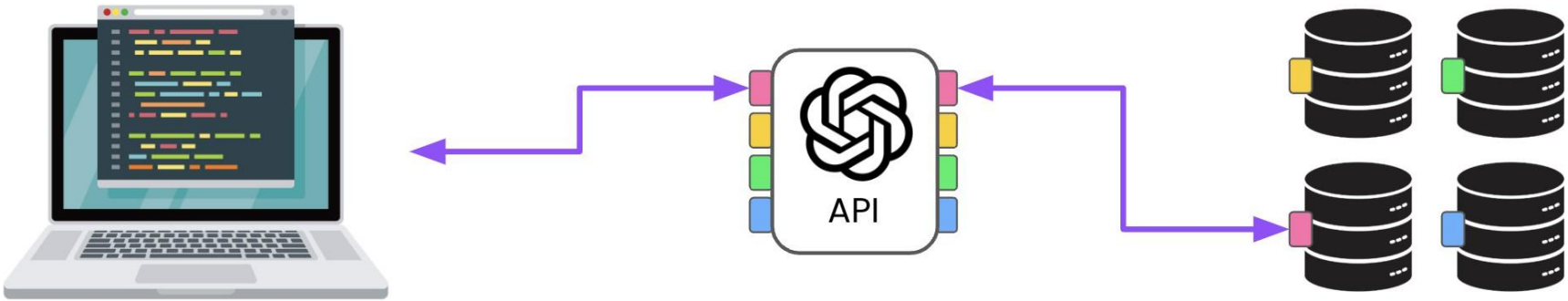
Let's practice!

WORKING WITH THE OPENAI API

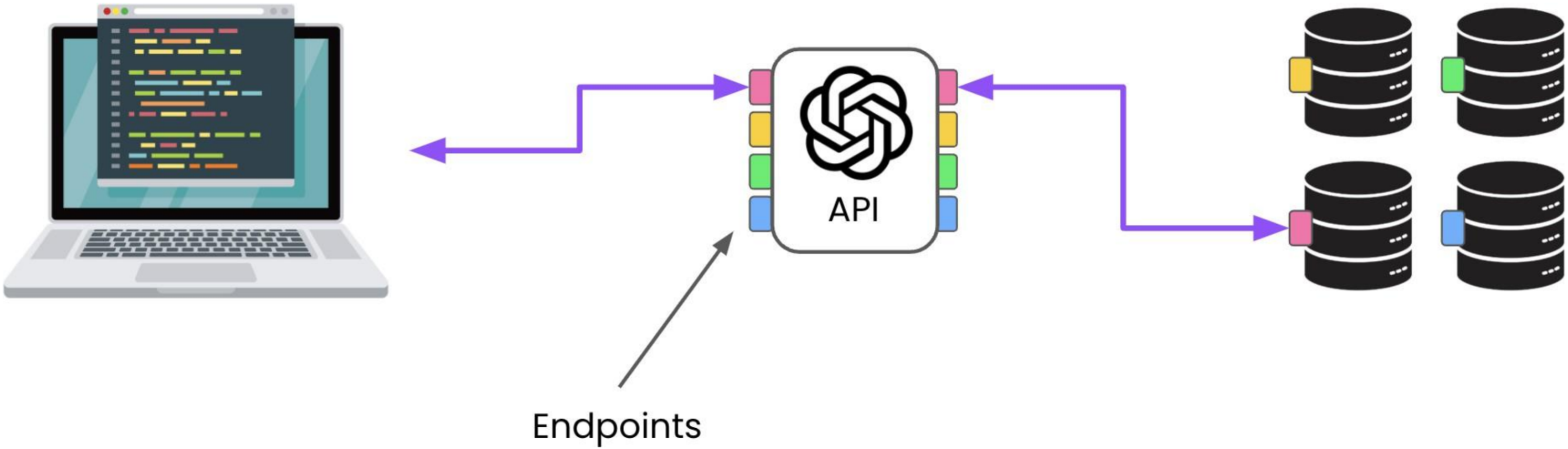
Making requests to the OpenAI API

WORKING WITH THE OPENAI API

A
P
I
e
n
d
p
o
n
t
s

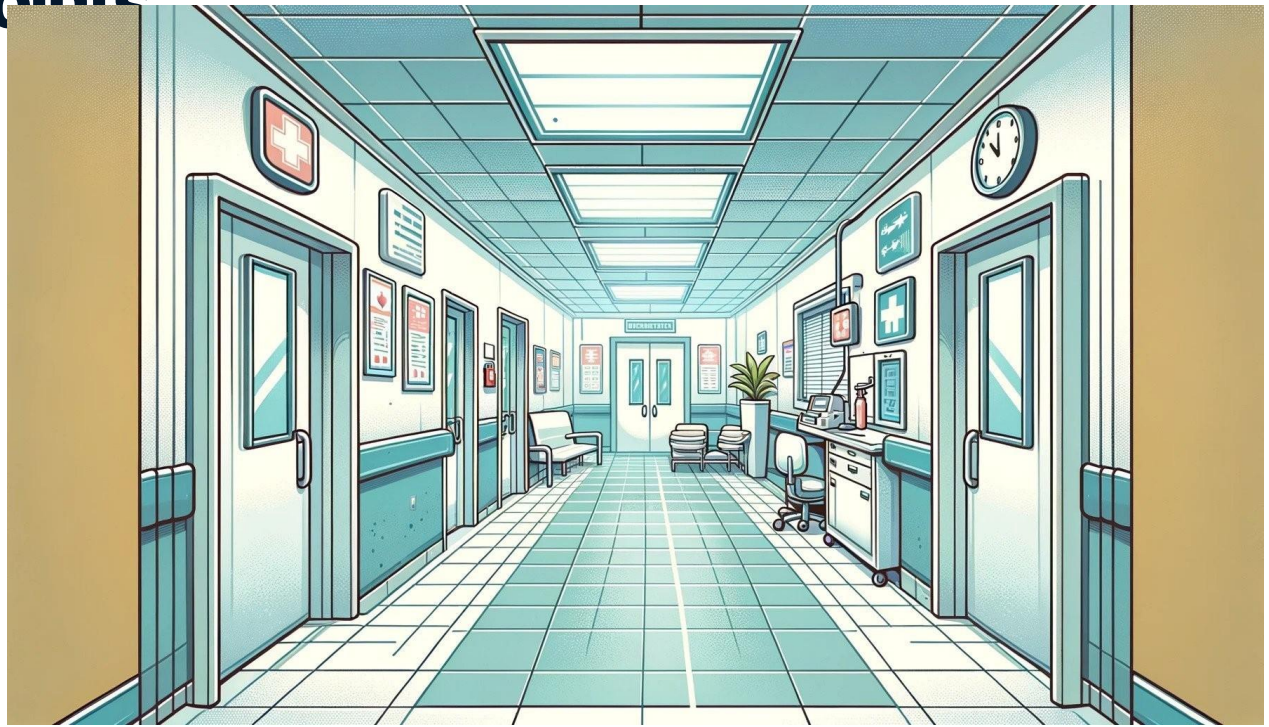


API Endpoints



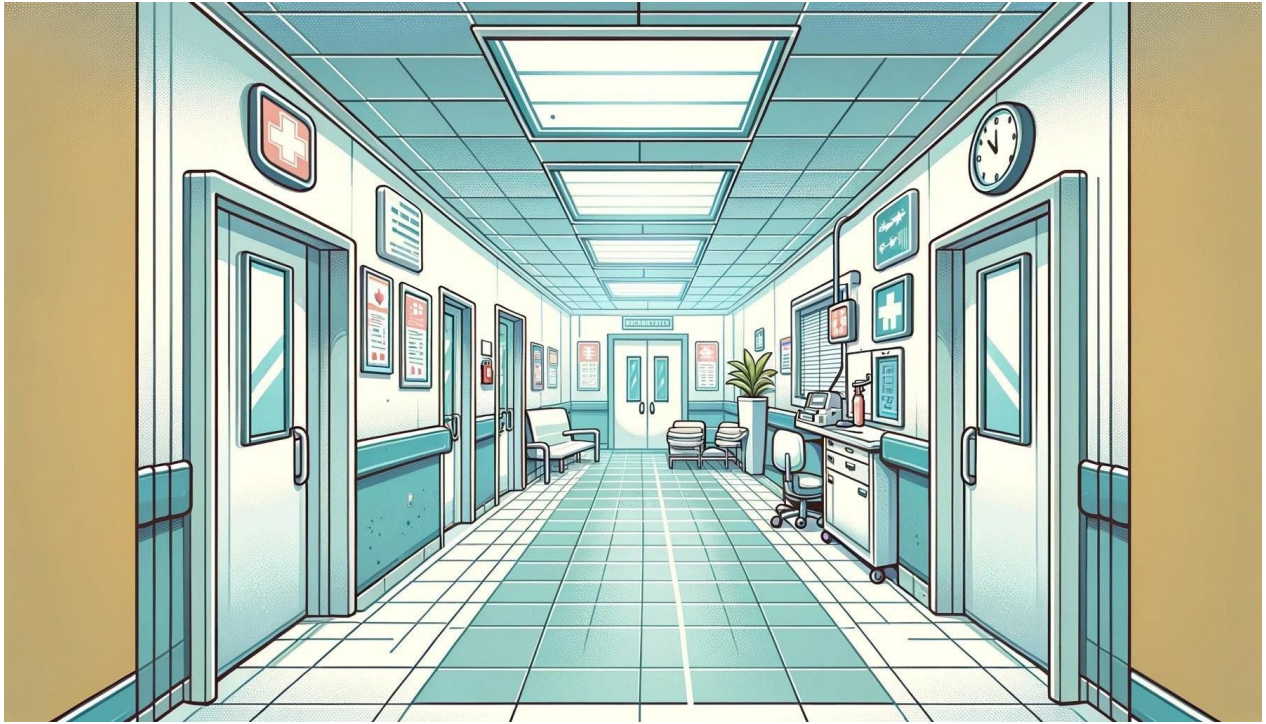
- **Endpoints** →API access point designed for specific interactions

API endpoints



- **Authentication** →Controls on access to API endpoint services (often unique **key**)

API authentication



API usage costs

- For OpenAI API:
  - Model
  - Larger inputs + outputs = Greater cost

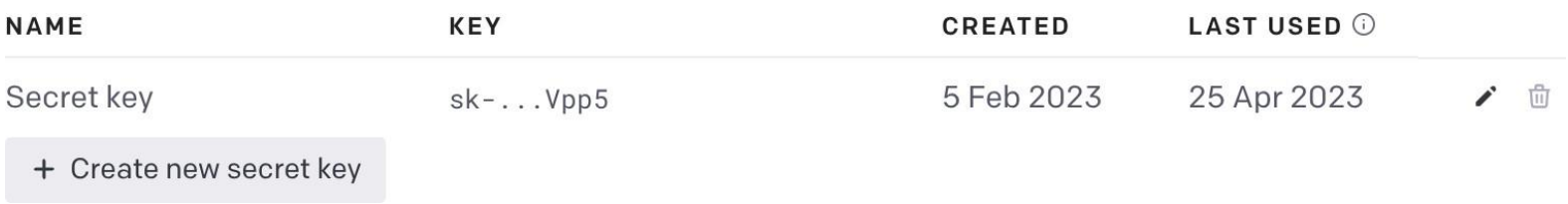
- **No additional costs for this course** □

[1] https://openai.com/pricing

Creating an API key

- **Isn't required in this course**

1. Create your account → https://platform.openai.com/signup
2. Go to the API keys page → https://platform.openai.com/account/api-keys
3. Create a new secret key and **copy it**



Making a request

- Demo

```
ChatCompletion(id='chatcmpl-AEcQbQekIzxcxVAKYVAjgUAXokgrl', choices=[Choice(finish_reason='length', index=0,
logprobs=None, message=ChatCompletionMessage(content='The OpenAI API is a cloud-based service provided by
OpenAI that allows developers to integrate advanced AI models into their applications.', refusal=None,
role='assistant', function_call=None, tool_calls=None))], created=1728047673, model='gpt-4o-mini-2024-07-18',
object='chat.completion', service_tier=None, system_fingerprint='fp_f85bea6784',
usage=CompletionUsage(completion_tokens=30, prompt_tokens=14, total_tokens=44,
prompt_tokens_details={'cached_tokens': 0}, completion_tokens_details={'reasoning_tokens': 0}))
```

```
ChatCompletion(id='chatcmpl-AEcQbQekIzxcxVAKYVAjgUAXokgrl',
               choices=[Choice(finish_reason='length', index=0, logprobs=None,
                               message=ChatCompletionMessage(content='The OpenAI API is a cloud-based service
                               provided by OpenAI that allows developers to integrate advanced AI models
                               into their applications.',
                               refusal=None, role='assistant', function_call=None, tool_calls=None))],
               created=1728047673,
               model='gpt-4o-mini-2024-07-18',
               object='chat.completion', service_tier=None, system_fingerprint='fp_f85bea6784',
               usage=CompletionUsage(completion_tokens=25, prompt_tokens=14, total_tokens=39,
                                     prompt_tokens_details={'cached_tokens': 0},
                                     completion_tokens_details={'reasoning_tokens': 0}))
```

```
print(response.choices)
```

```
[Choice(finish_reason='length', index=0, logprobs=None,
 message=ChatCompletionMessage(content='The OpenAI API is a cloud-based service provided by
 OpenAI that allows developers to integrate advanced AI models into their applications.',
 refusal=None, role='assistant', function_call=None, tool_calls=None))]
```

```
print(response.choices[0])
```

```
Choice(finish_reason='length', index=0, logprobs=None,
 message=ChatCompletionMessage(content='The OpenAI API is a cloud-based service provided by
 OpenAI that allows developers to integrate advanced AI models into their applications.',
 refusal=None, role='assistant', function_call=None, tool_calls=None))
```

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

```
ChatCompletionMessage(content='The OpenAI API is a cloud-based service provided by
 OpenAI that allows developers to integrate advanced AI models into their applications.',
 refusal=None, role='assistant', function_call=None, tool_calls=None)
```

```
print(response.choices[0].message.content)
```

```
The OpenAI API is a cloud-based service provided by OpenAI that allows developers to
integrate advanced AI models into their applications.
```

Let's practice!

Summarizing and editing text

Using GPT-5 to transform and enhance text content

Text Editing

Text Summarization

Response Length Control

Token Usage & Costs

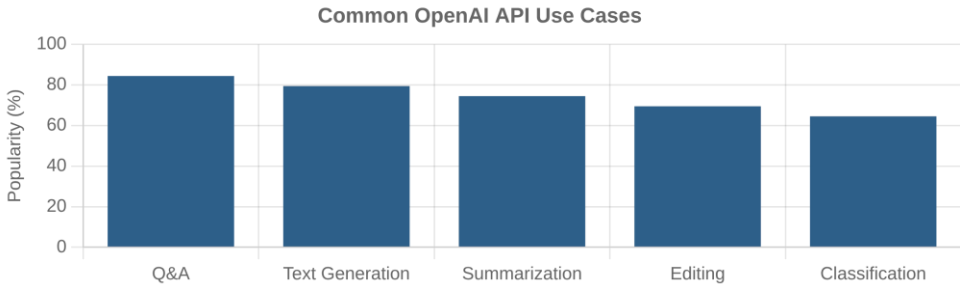
Recap...

So far, we've learned how to:

- Understand what the OpenAI API is and how it differs from the web interface
- Make requests to the OpenAI API using Python
- Interpret and extract information from API responses

Next up: We'll explore how to use the API for specific tasks like text editing, summarization, and more advanced features.

```
# Q&A example
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

response = client.chat.completions.create(
  model="gpt-5",
  messages=[
    {"role": "user", "content": "How many days are in October?"}
  ]
)

print(response.choices[0].message.content)

# Output: "October has 31 days."
```



Text editing

Example: updating the name, pronouns, and job title

- GPT-5 can modify text according to specific instructions
- Provide clear instructions about what to change
- Include the original text to be edited

> **Tip:** Be specific about what should change and what should remain the same.

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

prompt = """
Update name to Maarten, pronouns to he/him, and
job title to Senior Content Developer in the
following text:

Joanne is a Content Developer at Learning
Science Inc. Her favorite programming language
is R, which she uses for her statistical analyses.
"""

response = client.chat.completions.create(
  model="gpt-5",
  messages=[{"role": "user", "content": prompt}]
)
```

Text editing

Using GPT-5 to edit text by providing clear instructions:

- Specify exactly what changes you want made
- Include the original text to be edited
- Be specific about what elements to preserve

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

prompt = """
Update name to Maarten, pronouns to he/him, and job title to Senior Content Developer in the following text:

Joanne is a Content Developer at DataCamp. Her favorite programming language is R, which she uses for her statistical analyses.
"""

response = client.chat.completions.create(
    model="gpt-5"’
    messages=[
        {"role": "user", "content": prompt}
    ],
    verbosity="low"
)

print(response.choices[0].message.content)
```

Result:
Maarten is a Senior Content Developer at Learning Science Inc.

Text summarization

Example: summary of customer chat transcripts

- Provide clear instructions in your prompt
- Specify the format you want (e.g., bullet points, key points)
- Indicate desired length or level of detail

GPT-5 advantage: With its enhanced understanding capabilities, GPT-5 can produce more accurate and concise summaries that better capture the key information from longer texts.

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

text = """
Customer: Hi, I'm trying to log into my account, but it keeps saying
my password is incorrect. I'm sure I'm entering the right one.

Support: I'm sorry to hear that!
Have you tried resetting your password?

Customer: I tried, but I'm not receiving the reset email.

Support: Let me check if the email was sent. Yes, it appears
the system sent it. Please check your spam folder.

Customer: Found it! Thanks. Actually, I just remembered I can
log in with Google. I'll try that instead.

Support: Great! Let me know if you need anything else.
"""

prompt = f"""Summarize the customer support chat in three concise key points: {text}"""

response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user", "content": prompt}],
```

Text summarization

Using GPT-5 to create concise summaries of longer texts:

- Specify the desired format and length in your prompt
- Use the verbosity parameter to control response length
- GPT-5 excels at extracting key information while maintaining context

Result:

1. Customer was unable to log in due to password issues and wasn't receiving the reset email.
2. Support confirmed the reset email was sent and suggested checking the spam folder.
3. Customer found the email but opted to use Google login instead.

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

text = """
Customer: Hi, I'm trying to log into my account, but it keeps saying
my password is incorrect. I'm sure I'm entering the right one.

Support: I'm sorry to hear that!
Have you tried resetting your password?

Customer: I tried, but I'm not receiving the reset email.

Support: Let me check if the email was sent. Yes, it appears
the system sent it. Please check your spam folder.

Customer: Found it! Thanks. Actually, I just remembered I can
log in with Google. I'll try that instead.

Support: Great! Let me know if you need anything else.
"""

prompt = f"""Summarize the customer support chat in three concise key points: {text}"""

response = client.chat.completions.create(
  model="gpt-5",
  messages=[{"role": "user", "content": prompt}],
```

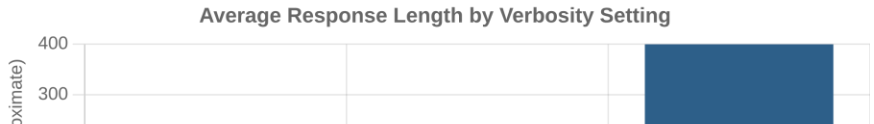
Controlling response length

GPT-5 offers the new `verbosity` parameter to control response length:

- **low**: Generates concise, to-the-point responses
- **medium** : Balanced responses with moderate detail (default)
- **high**: Comprehensive, detailed responses

> **Tip:** Use `verbosity="low"` for summaries and quick answers, `verbosity="high"` for detailed explanations or educational content.

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Example with different verbosity levels
question = "Explain how neural networks work"

# Concise response
concise = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user", "content": question}],
    verbosity="low"
)

# Detailed response
detailed = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user", "content": question}],
    verbosity="high"
)
```



Understanding tokens

Tokens are the basic units of text that GPT-5 processes:

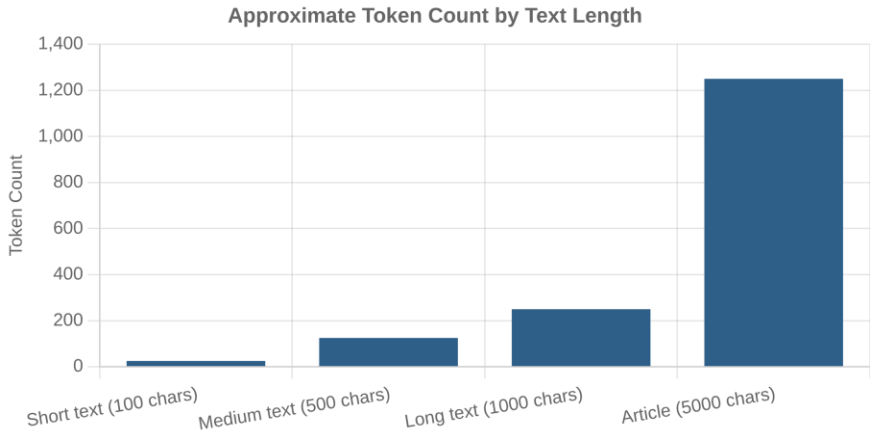
- Tokens are pieces of words, not always whole words
- ~4 characters per token in English (average)
- API costs are calculated based on token count
- Both input and output tokens count toward costs

> **GPT-5:** Has a context window of 400,000 tokens, allowing for much longer inputs and outputs compared to previous models.

Example of tokenization:

The	Open	AI	API	is	power	ful	.

8 tokens for "The OpenAI API is powerful."



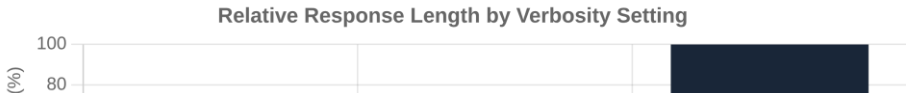
Controlling response length

GPT-5 offers the new `verbosity` parameter to control response length:

- **low**: Concise, to-the-point responses
- **medium**: Balanced responses with moderate detail (default)
- **high**: Comprehensive, detailed responses

> **Best practice:** Choose the appropriate verbosity level based on your use case. Use `low` for summaries and quick answers, `medium` for general information, and `high` for in-depth explanations.

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Example with low verbosity
response_low = client.chat.completions.create(
    model="gpt-5",
    messages=[
        {"role": "user", "content": "Explain quantum computing"}
    ],
    verbosity="low"
)

# Example with high verbosity
response_high = client.chat.completions.create(
    model="gpt-5",
    messages=[
        {"role": "user", "content": "Explain quantum computing"}
    ],
    verbosity="high"
)
```



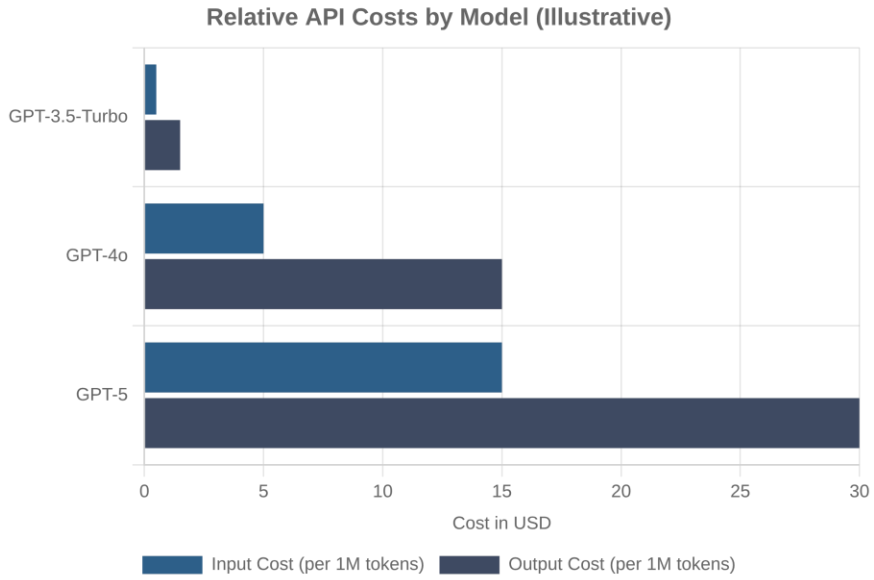
Calculating cost

OpenAI API costs are based on token usage:

- Different models have different pricing
- Input tokens cost less than output tokens
- Costs are calculated per 1,000 tokens

Cost = (Input tokens × Input price + Output tokens × Output price) / 1000

Note: The `usage` field in the API response provides token counts for calculating costs.

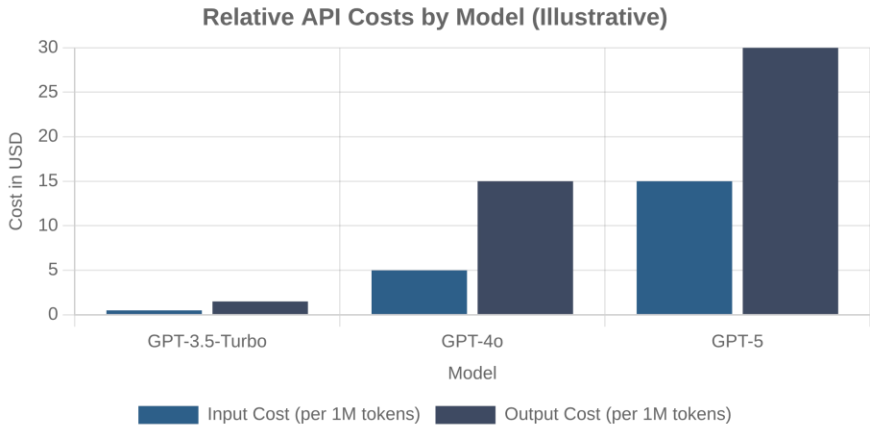


Calculating cost

Understanding API costs for GPT-5:

- Costs are calculated based on token usage
- Both input and output tokens count toward costs
- Output tokens typically cost more than input tokens
- The `usage` field in the response provides token counts

> **Note:** For this course, all API usage costs are covered, so you don't need to worry about billing.




```
# Example response usage data
response.usage

# Output:
CompletionUsage(
    prompt_tokens=20,
    completion_tokens=40,
    total_tokens=60
)
```

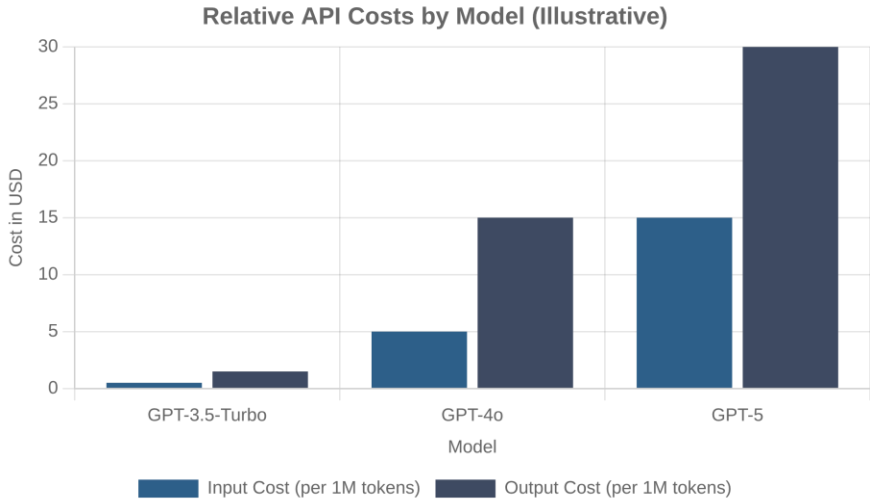
Calculating cost

How to calculate the cost of using the OpenAI API:

- Cost depends on model, input tokens, and output tokens
- Input and output tokens have different pricing
- More powerful models cost more per token

Total Cost = (Input tokens × Input price) + (Output tokens × Output price)

Note: The `usage` field in the API response provides token counts for calculating costs.



Text generation

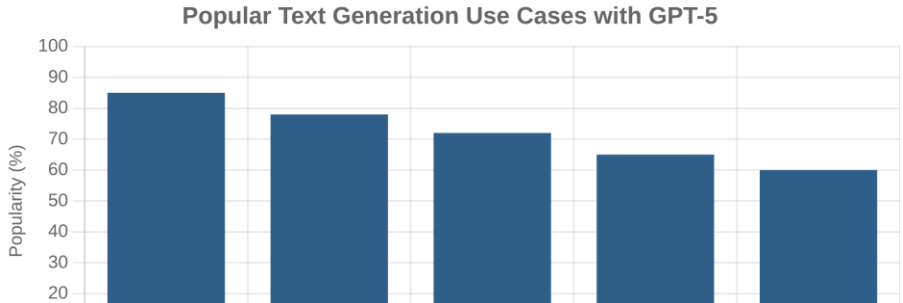
Using GPT-5 to create high-quality content

Controlling Response Randomness

Marketing Content Generation

Product Descriptions

Creative Writing

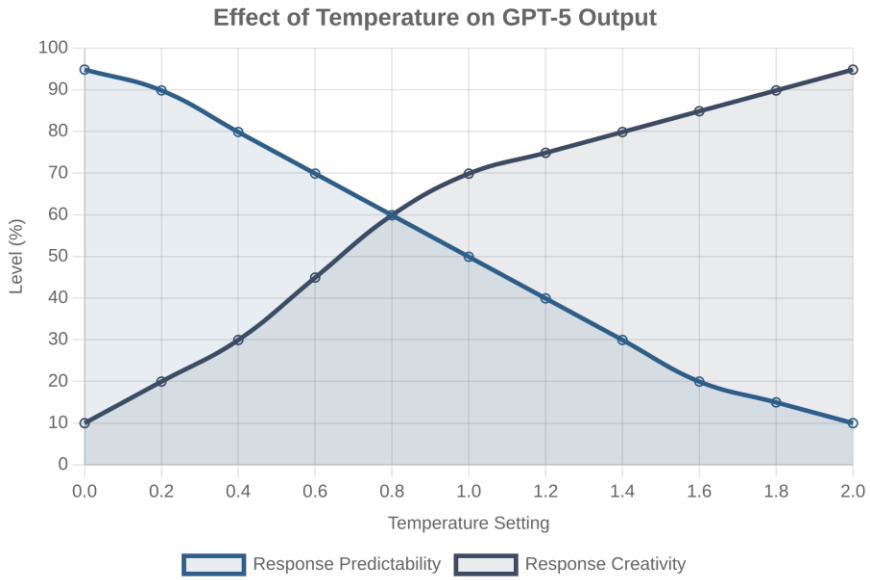


How output is generated

Understanding how GPT-5 generates text:

- GPT-5 predicts the most likely next token based on context
- Some randomness is introduced to create varied responses
- temperature parameter controls randomness level
- reasoning_effort parameter controls depth of reasoning

GPT-5 advantage: More precise control over output generation with new parameters like reasoning_effort and verbosity .



Controlling Response Randomness

The **temperature** parameter controls the randomness of the model's responses:

More Deterministic **More Random**

0	1	2
Consistent Predictable	Balanced (Default)	Creative Varied

Use cases by temperature:

Low (0-0.3): Customer service, factual Q&A, documentation
Medium (0.4-0.8): Conversational responses, explanations
High (0.9-2.0): Creative writing, brainstorming, idea generation

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# With temperature = 0 (deterministic)
response_deterministic = client.chat.completions.create(
  model="gpt-5",
  messages=[
    {"role": "user", "content": "Life is like a box of chocolates"}
  ],
  temperature=0
)

# With temperature = 2 (highly random)
response_random = client.chat.completions.create(
  model="gpt-5",
  messages=[
    {"role": "user", "content": "Life is like a box of chocolates"}
  ],
  temperature=2
)

print("Deterministic:", response_deterministic.choices[0].message.content)
print("Random:", response_random.choices[0].message.content)
```

Text generation for marketing

Using GPT-5 to generate marketing content:

- Create compelling marketing copy for different channels
- Tailor content to specific target audiences
- Adjust tone and style to match brand voice

> **Tip:** Provide specific details about your product, target audience, and desired tone for better results.

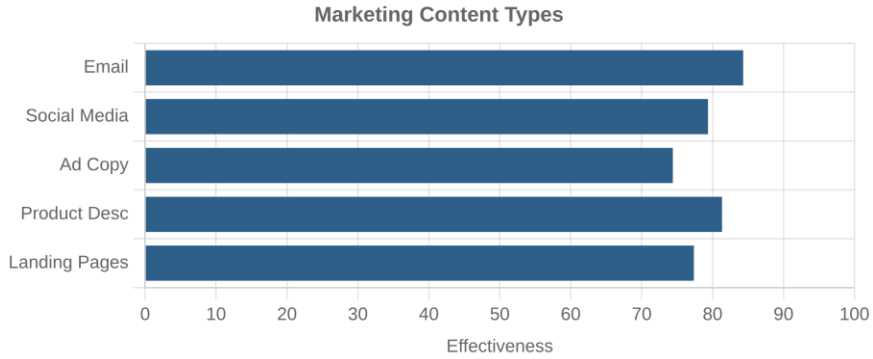
```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

prompt = """
Create a marketing email for our new online
Python course. Target audience: professionals
looking to upskill. Tone: professional but
enthusiastic. Key features:
- Learn Python from scratch in 8 weeks
- Real-world projects with code reviews
- 24/7 mentor support
"""

response = client.chat.completions.create(
  model="gpt-5",
  messages=[{"role": "user", "content": prompt}],
  verbosity="medium"
)
```

Text generation for marketing

Using GPT-5 to generate marketing content:

- Create compelling marketing copy for different channels
- Tailor content to specific target audiences
- Adjust tone and style to match brand voice

> **Tip:** Provide specific details about your product, audience, and desired tone for better results.



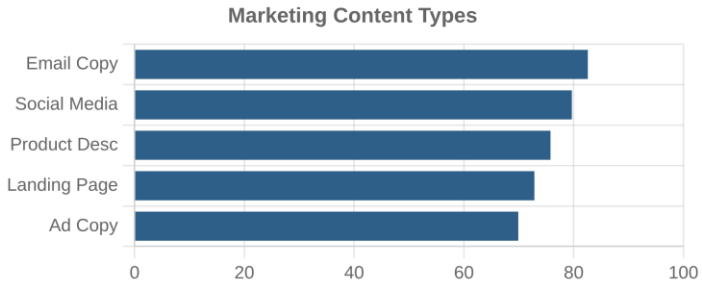

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

prompt = """
Create a marketing email for a new online course on
data science. Target audience: professionals looking
to upskill. Tone: professional but approachable.
Include a compelling subject line.
"""

response = client.chat.completions.create(
  model="gpt-5",
  messages=[{"role": "user", "content": prompt}],
```

Text generation for marketing

Using GPT-5 to generate marketing content:

- Create compelling marketing copy for different audiences
- Tailor messaging to specific customer segments
- Adjust tone and style to match brand voice

Best practice: Provide specific details about your product, audience, and desired tone for better results.




```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

prompt = """
Create a marketing email for a new data science
course. Target audience: working professionals.
Tone: professional but enthusiastic.
"""

response = client.chat.completions.create(
  model="gpt-5",
  messages=[{"role": "user",
        "content": prompt}],
  verbosity="medium"
)
```

Text generation for product descriptions

Using GPT-5 to create compelling product descriptions:

- Generate detailed descriptions from basic specifications
- Highlight key features and benefits for customers
- Optimize for different platforms and audiences

Tip: Include specific product details, target audience, and unique selling points in your prompt.

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

prompt = """
Create a product description for wireless
noise-cancelling headphones with:
- 30-hour battery life
- Active noise cancellation
- Bluetooth 5.2 connectivity
- Memory foam ear cushions

Target audience: Frequent travelers
Tone: Premium, sophisticated
"""

response = client.chat.completions.create(
  model="gpt-5",
  messages=[{"role": "user", "content": prompt}]
)
```

Let's practice!

Shot prompting

Providing examples to guide model output

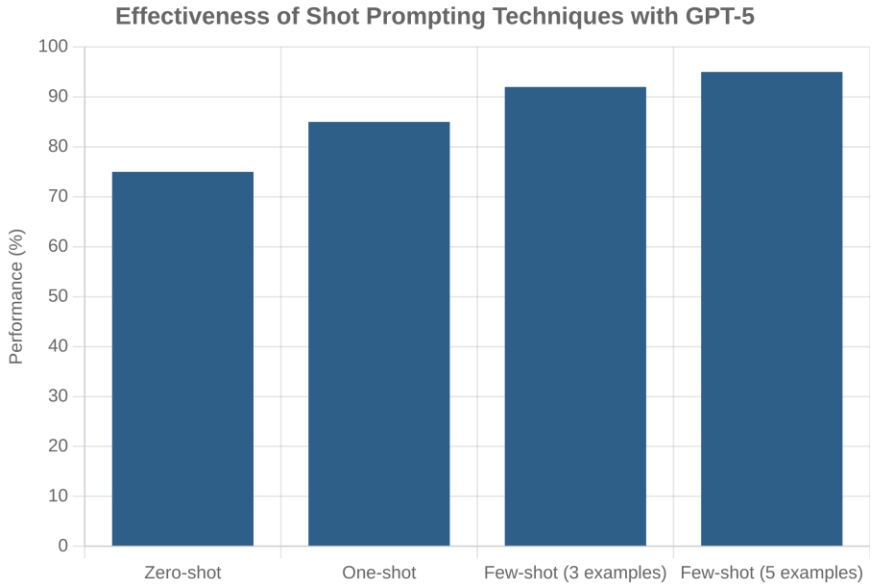
Shot prompting

Shot prompting is a technique to improve model performance by providing examples:

- Zero-shot: No examples provided
- One-shot: One example provided
- Few-shot: Multiple examples provided

Examples help the model understand the desired format and style

GPT-5 advantage: While GPT-5 performs better with fewer examples than previous models, shot prompting still improves results for complex or specialized tasks.

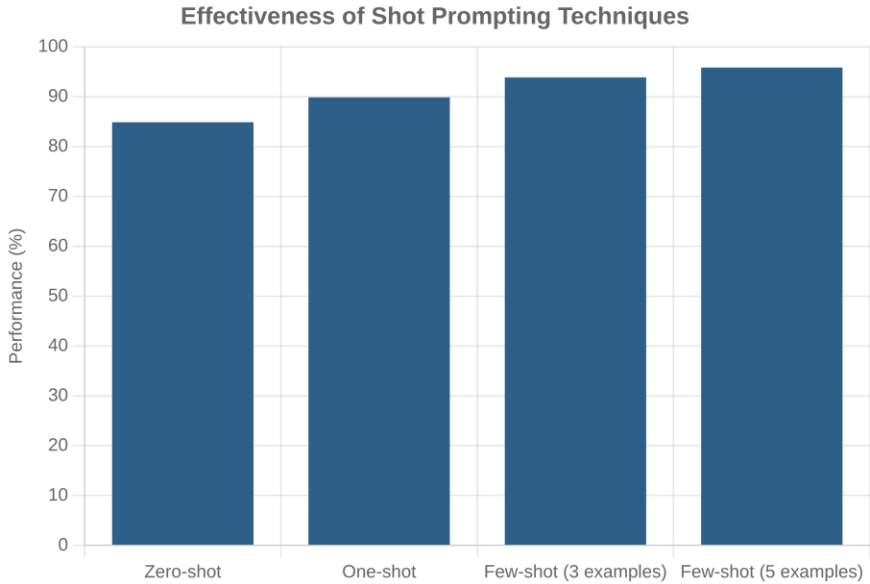


Shot prompting

Shot prompting refers to providing examples in your prompt:

- Zero-shot: No examples provided
- One-shot: One example provided
- Few-shot: Multiple examples provided

> **GPT-5 advantage:** While GPT-5 performs well with zero-shot prompting, providing examples can still improve performance for specialized tasks.



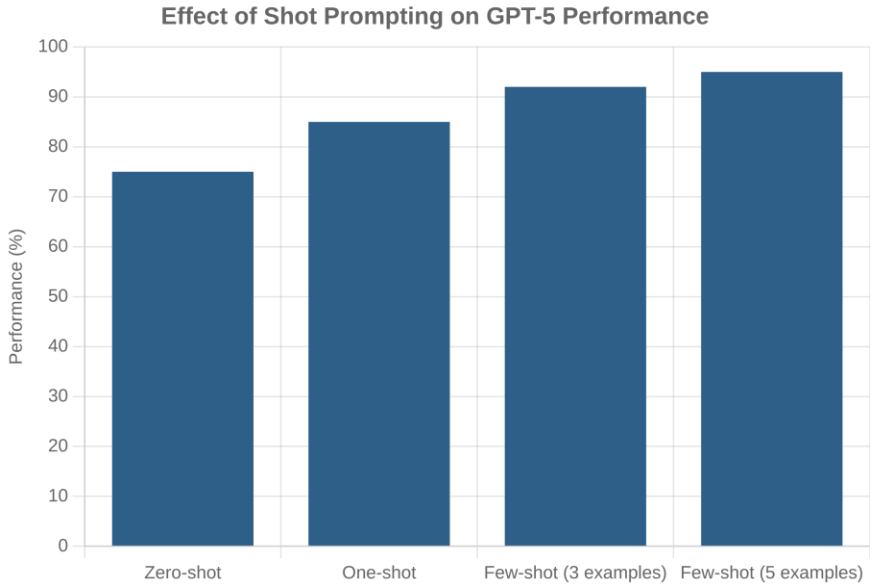
Shot prompting

Shot prompting improves model performance by providing examples:

- Zero-shot: No examples provided
- One-shot: One example provided
- Few-shot: Multiple examples provided

Examples help the model understand the desired format and style

GPT-5 advantage: While GPT-5 performs better with zero-shot prompting than previous models, it still benefits significantly from shot prompting for complex or specialized tasks.



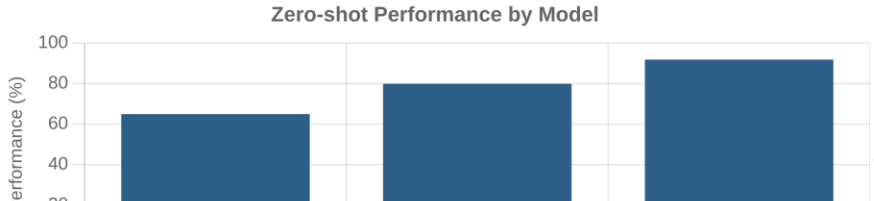
Zero-shot prompting

Zero-shot prompting means asking the model to perform a task without examples:

- No examples are provided in the prompt
- Relies on the model's pre-trained knowledge
- Works well for common tasks and formats

GPT-5 advantage: Significantly improved zero-shot performance compared to previous models, especially for complex reasoning tasks.

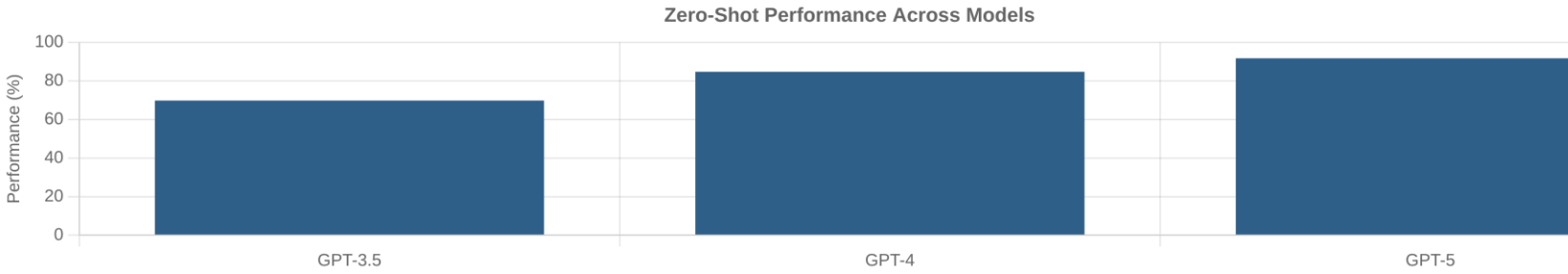
```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Zero-shot prompt - no examples provided
prompt = """
Classify the following text as either
'positive', 'negative', or 'neutral':

"The new restaurant was amazing! The food
was delicious and the service was excellent."
"""

response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user", "content": prompt}]
)
```



Zero-shot prompting

Zero-shot prompting: asking the model to perform a task without examples

- No examples provided in the prompt
- Relies on the model's pre-trained knowledge
- Works well for common tasks and formats

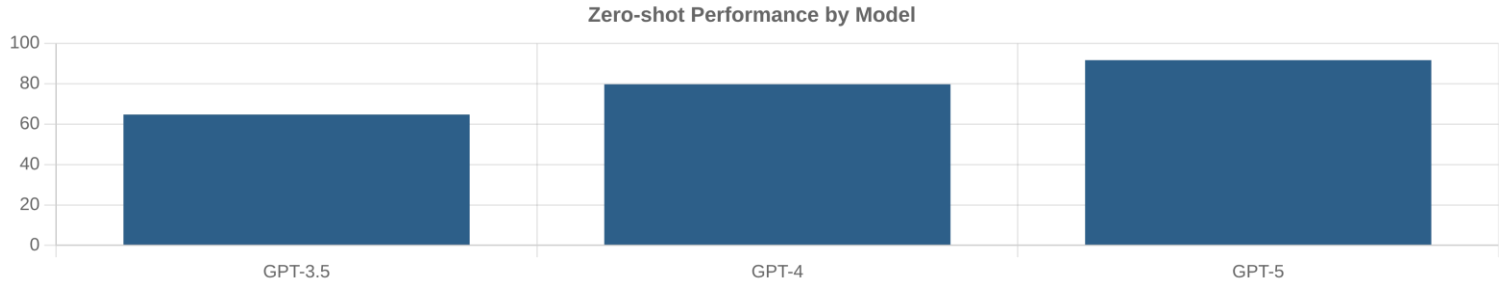



```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Zero-shot prompt
prompt = "Classify the following text as either 'positive', 'negative', or 'neutral': 'The service was quick and the staff was friendly.'"

response = client.chat.completions.create(
```

Zero-shot prompting

Zero-shot prompting means asking the model to perform a task without examples:

- No examples are provided in the prompt
- Clear instructions are essential
- Works well for common tasks and formats




```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Zero-shot prompt
prompt = "Classify the following text as either 'positive', 'negative', or 'neutral': 'The service was excellent.'"

response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user", "content": prompt}]
```

Zero-Shot Prompting (Limitations)

While zero-shot prompting is convenient, it has several limitations:

Common limitations of zero-shot prompting:

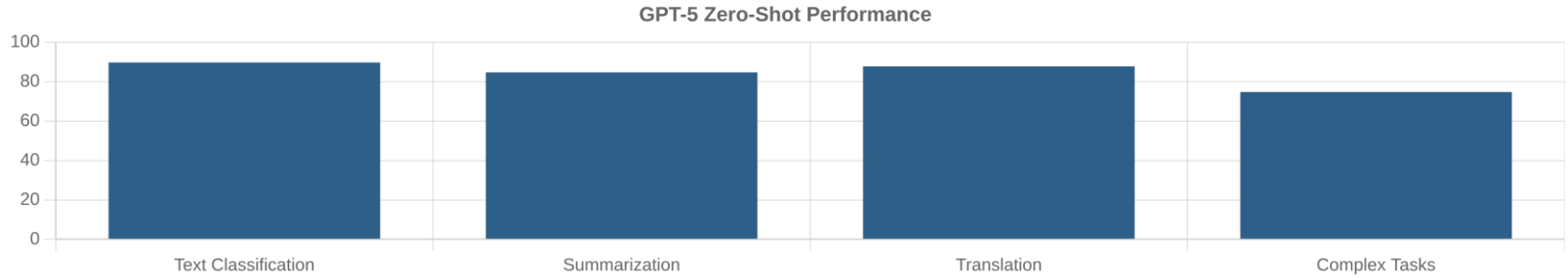
- Inconsistent formatting in outputs
- Unexpected additions to responses
- Misinterpretation of specialized tasks
- Difficulty with domain-specific terminology
- Unpredictable handling of edge cases

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Zero-shot prompt for specialized medical task
response = client.chat.completions.create(
  model="gpt-5",
  messages=[
    {
      "role": "user",
      "content": "Classify this symptom as bacterial or viral: 'Fever of 101°F, clear runny nose, and mild body aches for 2 days.'"
    }
  ]
)

print(response.choices[0].message.content)

# Potential output includes explanations:
# "Based on the symptoms described (fever of 101°F,
# clear runny nose, and mild body aches for 2 days),
# this presentation is more consistent with a VIRAL
# infection. Viral infections typically cause..."
```

Zero-shot prompting

Zero-shot prompting: asking the model to perform a task without examples

- Relies on the model's pre-trained knowledge
- Works well for common tasks and formats
- GPT-5 has improved zero-shot capabilities



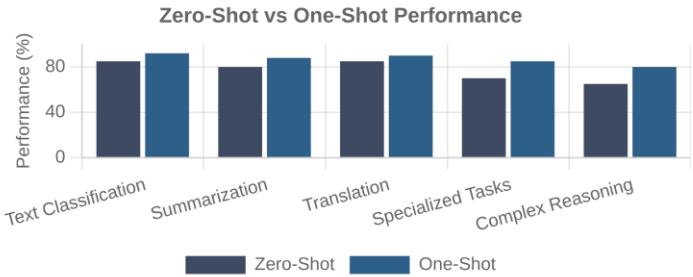

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Zero-shot prompt example
prompt = "Classify the following text as either 'positive', 'negative', or 'neutral': 'The service was excellent.'"

response = client.chat.completions.create(
    model="gpt-5",
```

One-shot prompting

One-shot prompting provides a single example to guide the model:

- One example is provided in the prompt
- Helps the model understand the desired format
- More effective than zero-shot for specialized tasks

Best practice: Choose a clear, representative example that demonstrates the exact format and style you want the model to follow.



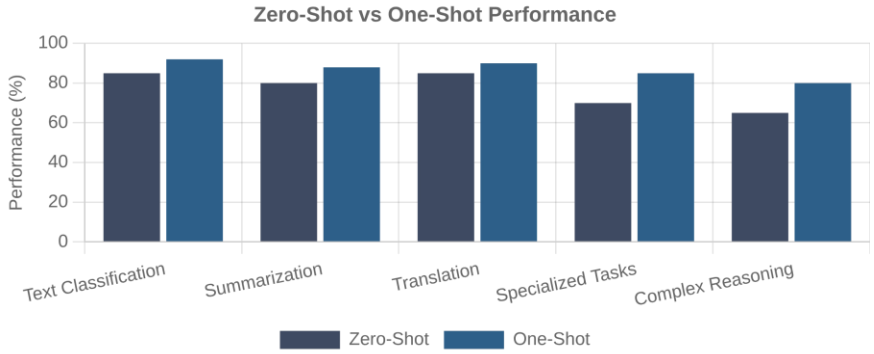

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# One-shot prompt with example
prompt = """
Example:
Input: "The food was delicious."
Classification: positive

Input: "The service was terrible and slow."
Classification:
"""

response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user", "content": prompt}]
```

One-shot prompting

One-shot prompting provides a single example to guide the model:

- One example is provided in the prompt
- Helps the model understand the desired format and style
- More effective than zero-shot for specialized tasks

> **Best practice:** Choose a high-quality, representative example that clearly demonstrates the desired output format and style.



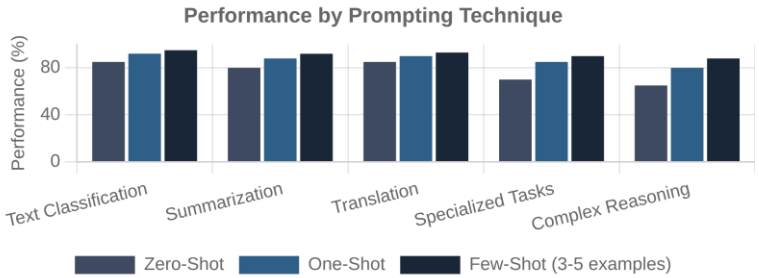

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# One-shot prompt with example
prompt = """
Classify the sentiment as positive, negative, or neutral.

Text: The food was delicious but the service was slow.
Sentiment: neutral

Text: The new update completely broke my workflow.
Sentiment:
"""
```

Few-shot prompting

Few-shot prompting provides multiple examples to guide the model:

- Multiple examples are provided in the prompt
- Helps the model learn patterns and formats more effectively
- Most effective for complex or specialized tasks

Best practice: Use diverse, high-quality examples that cover the range of outputs you expect. Include 3-5 examples for optimal results.



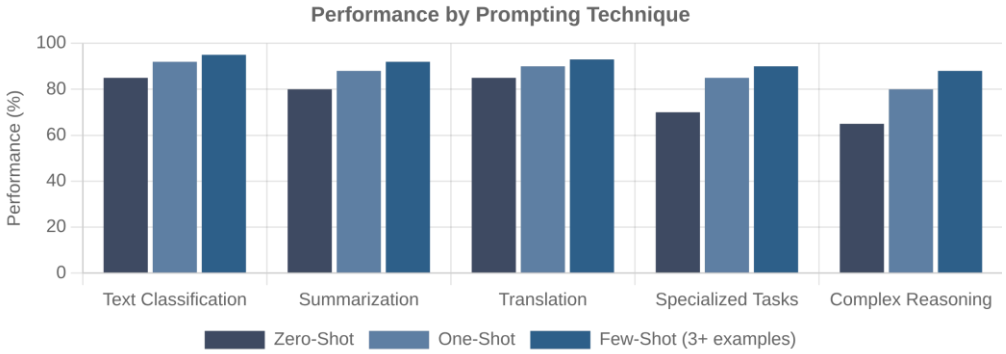

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv( "OPENAI_API_KEY" ))

# Few-shot prompt with multiple examples
prompt = """
Classify the sentiment as positive, negative, or neutral.

Text: The food was delicious but the service was slow.
Sentiment: neutral

Text: I absolutely loved the new features in the app!
Sentiment: positive

Text: The hotel room was dirty and the staff was rude.
Sentiment: negative

Text: The conference had interesting speakers.
```

Few-shot prompting

Few-shot prompting provides multiple examples to guide the model:

- Multiple examples are provided in the prompt
- Helps the model learn patterns and nuances
- Most effective for complex or specialized tasks

Best practice: Use diverse examples that cover different cases or edge cases the model might encounter in your task.



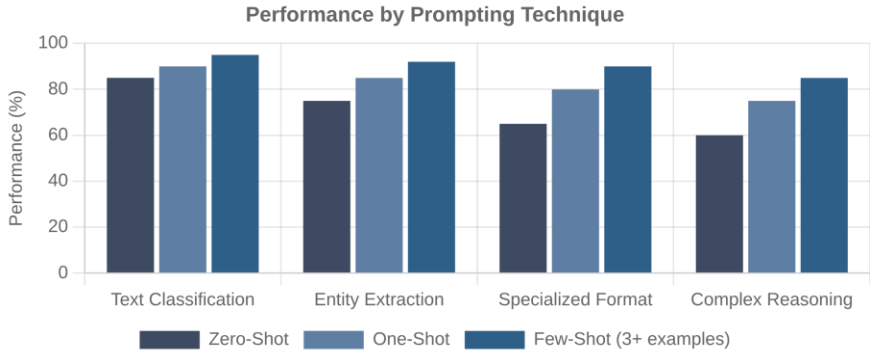

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Few-shot prompt with multiple examples
prompt = """
Classify the sentiment as positive, negative, or neutral.

Text: The food was delicious but the service was slow.
Sentiment: neutral

Text: I absolutely loved the new features in the app!
Sentiment: positive

Text: The hotel room was dirty and the staff was rude.
```

Few-shot prompting

Few-shot prompting provides multiple examples to guide the model:

- Multiple examples are provided in the prompt
- Helps the model learn patterns and nuances
- Most effective for complex or specialized tasks

Best practice: Use diverse, high-quality examples that cover different cases or edge conditions the model might encounter.



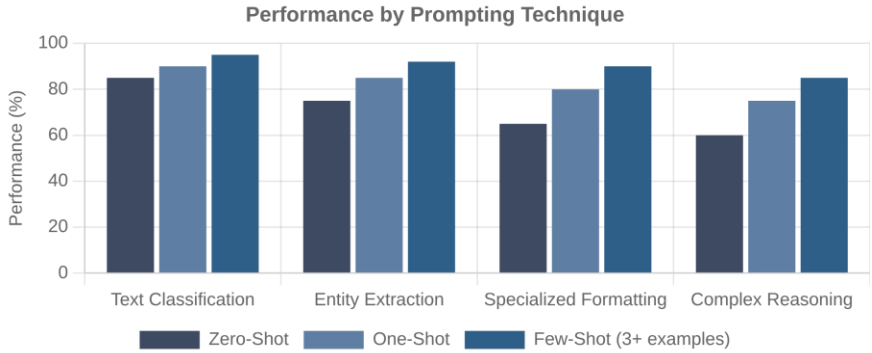

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Few-shot prompt with multiple examples
prompt = """
Extract the company name and product type:

Text: Apple launched their new iPhone 15 today.
Company: Apple
Product: iPhone 15

Text: Tesla announced the Cybertruck will ship next month.
Company: Tesla
Product: Cybertruck
```

Few-shot prompting

Few-shot prompting provides multiple examples to guide the model:

- Multiple examples are provided in the prompt
- Helps the model understand patterns and nuances
- Most effective for complex or specialized tasks

Best practice: Use diverse, high-quality examples that cover different cases or edge cases the model might encounter.



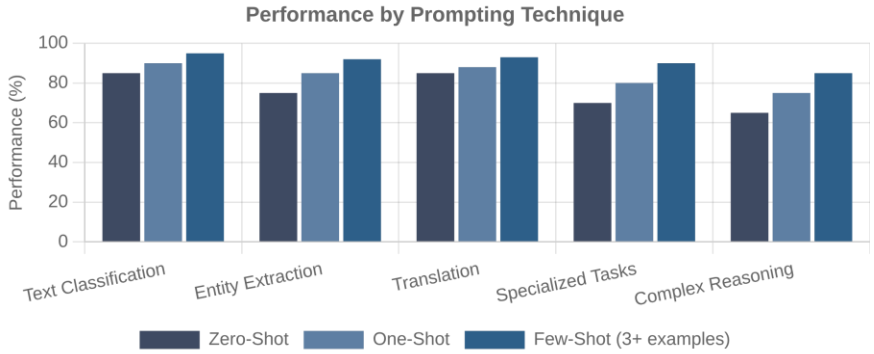

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Few-shot prompt with multiple examples
prompt = """
Extract the company name and product type:

Text: Apple released their new iPhone 15 today.
Company: Apple
Product: iPhone 15

Text: Tesla announced the Cybertruck will ship next month.
Company: Tesla
Product: Cybertruck
```

Few-shot prompting

Few-shot prompting provides multiple examples to guide the model:

- Multiple examples are provided in the prompt
- Helps the model learn patterns and formats more effectively
- Most effective for complex or specialized tasks

Best practice: Use diverse, high-quality examples that cover the range of expected inputs and outputs for your task.



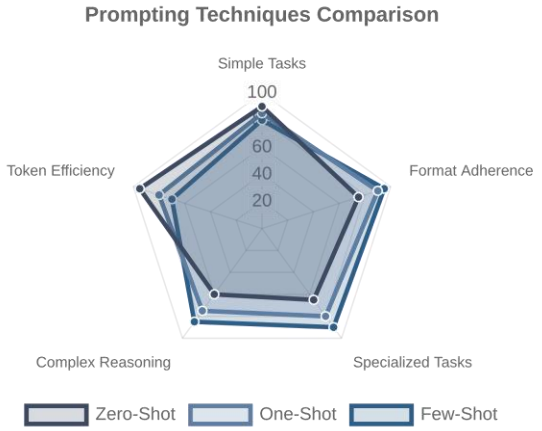

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Few-shot prompt with multiple examples
prompt = """
Extract the company name and product from each review:

Review: "I love my new Nike running shoes!"
Company: Nike
Product: running shoes

Review: "Apple's latest iPhone has an amazing camera."
Company: Apple
Product: iPhone
```

Comparing prompting techniques

Choosing the right prompting technique depends on your task:

- **Zero-shot:** Simple tasks with clear instructions
- **One-shot:** Tasks requiring specific format or style
- **Few-shot:** Complex or specialized tasks

GPT-5 advantage: Requires fewer examples than previous models to achieve the same performance, making prompting more efficient.



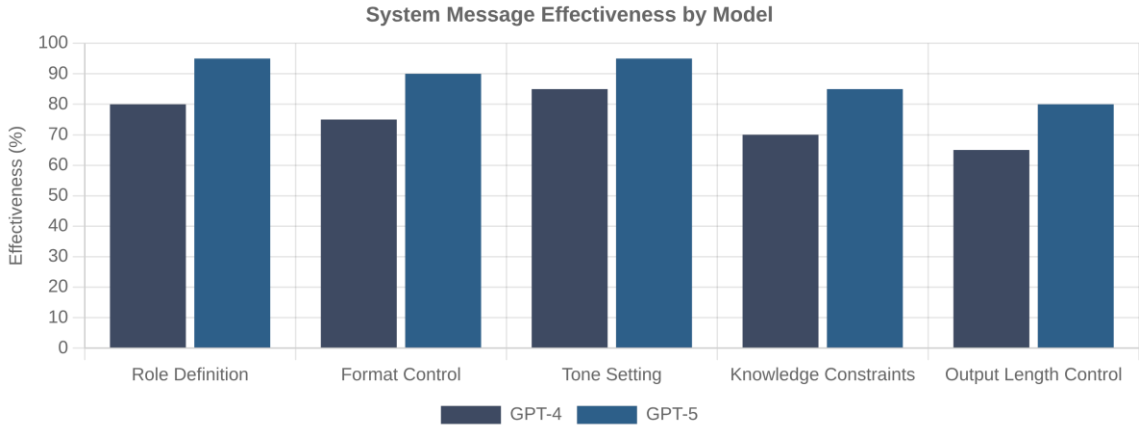

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Choose the appropriate prompting technique
# based on task complexity
response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user", "content": prompt}],
    verbosity="medium"
)
```

System messages

System messages define how the model behaves throughout the conversation:

- Set the behavior, personality, or role of the assistant
- Provide context and background information
- Define constraints and output formats
- GPT-5 follows system instructions more reliably than previous models

Best practice: Be specific and clear in system messages. They set the foundation for the entire conversation.



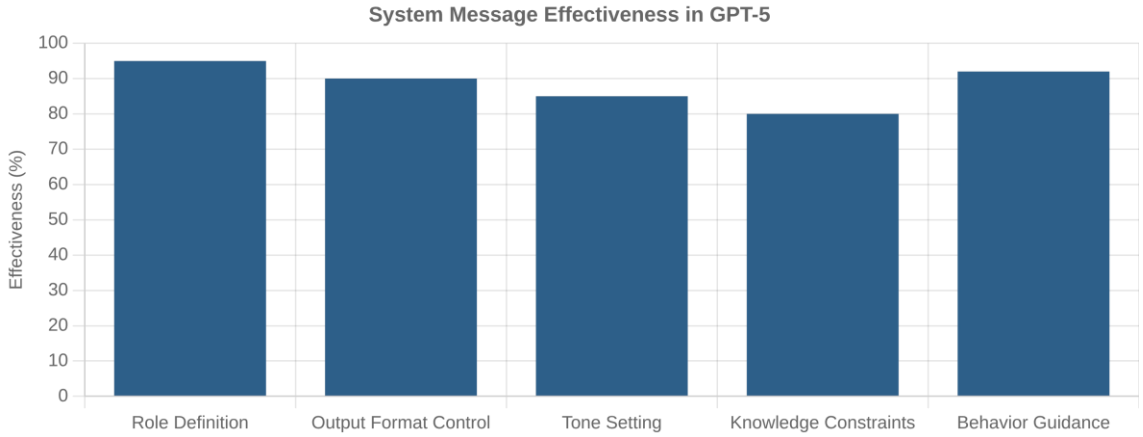

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

response = client.chat.completions.create(
  model="gpt-5",
  messages=[
    {"role": "system", "content": "You are a helpful assistant that speaks like Shakespeare."},
    {"role": "user", "content": "Tell me about artificial intelligence."}
  ]
```

System messages

System messages help set the behavior and context for the model:

- Define the AI's role, personality, or expertise
- Set constraints or guidelines for responses
- Provide context that persists across the conversation
- GPT-5 has improved ability to follow system instructions

> **Best practice:** Keep system messages clear, concise, and focused on the most important instructions.



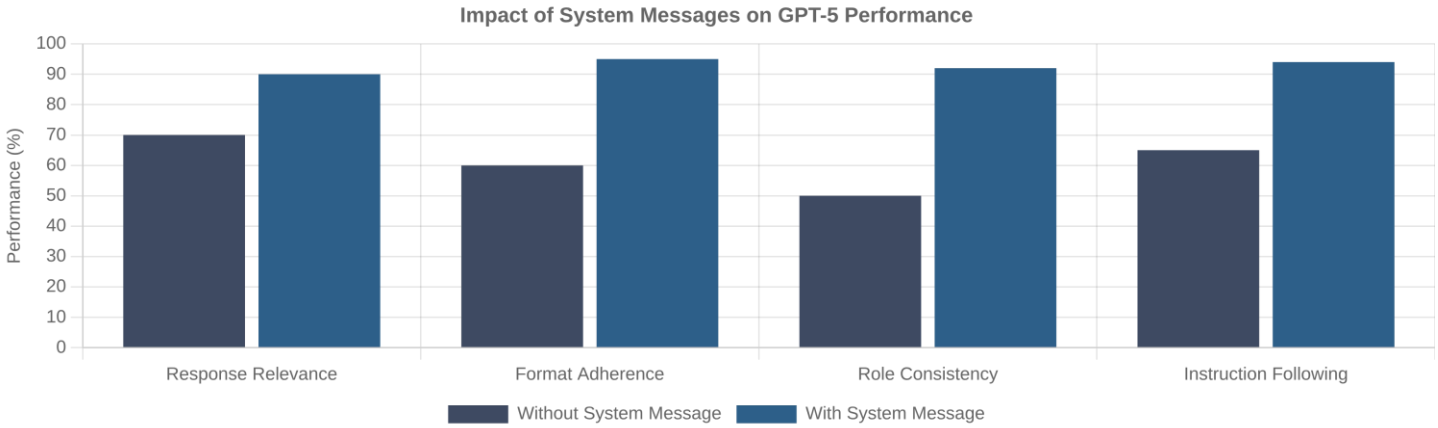

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

response = client.chat.completions.create(
  model="gpt-5",
  messages=[
    {"role": "system", "content": "You are a helpful assistant that speaks like Shakespeare."},
    {"role": "user", "content": "Tell me about artificial intelligence."}
  ]
```

System messages

System messages set the behavior and context for the model:

- Define the AI's role, personality, and constraints
- Provide context and background information
- Set output format and response guidelines
- GPT-5 has improved adherence to system instructions

Best practice: Be specific and clear in system messages. They're invisible to the



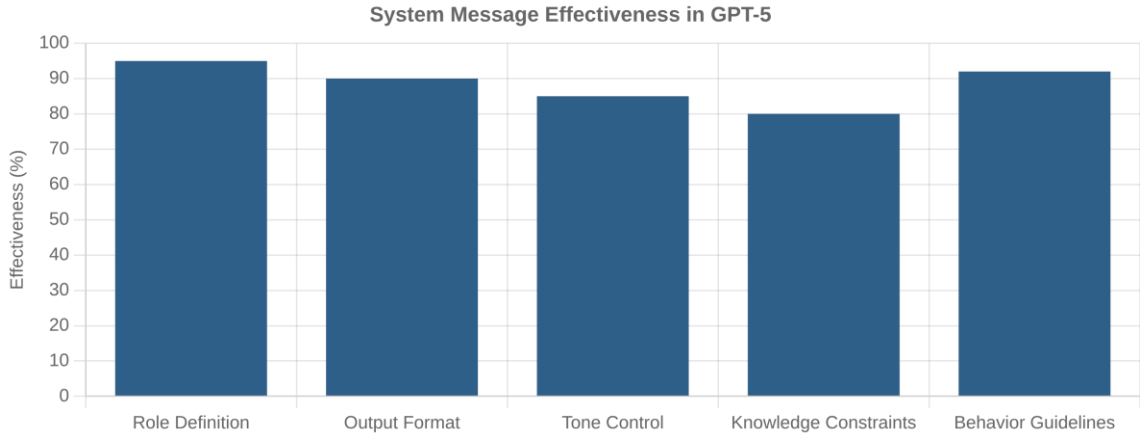

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

response = client.chat.completions.create(
  model="gpt-5",
  messages=[
    {"role": "system", "content": "You are a helpful financial advisor. Provide concise advice with bullet points."},
    {"role": "user", "content": "How should I save for retirement?"}
  ]
```

System messages

System messages define how the model behaves throughout the conversation:

- Set the behavior, personality, or role of the assistant
- Provide context and background information
- Define constraints and output formats
- GPT-5 has improved ability to follow system instructions

> **Best practice:** Be specific and clear in system messages. The model will maintain this behavior throughout the conversation.

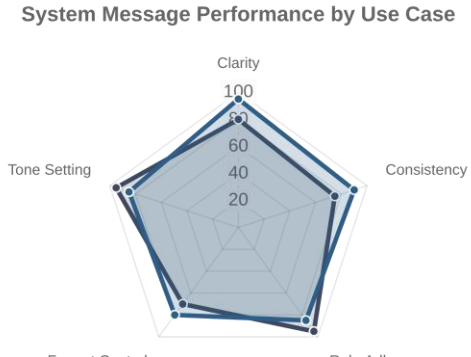



```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

response = client.chat.completions.create(
  model="gpt-5",
  messages=[
    {"role": "system", "content": "You are a helpful assistant that speaks like Shakespeare."},
    {"role": "user", "content": "Tell me about artificial intelligence."}
  ]
```

System messages examples

Effective system messages for different use cases:

- **Professional assistant:** "You are a helpful, professional assistant. Provide concise, accurate information."
- **Code assistant:** "You are a coding assistant. Provide well-

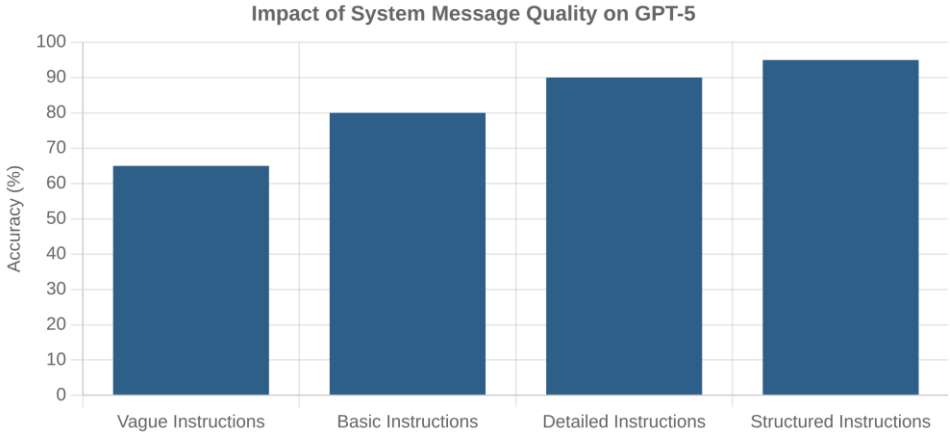


System messages: Best practices

Effective system messages can significantly improve model performance:

- **Be specific:** Clearly define the role, constraints, and expectations
- **Be concise:** Focus on the most important instructions
- **Prioritize:** Put the most important instructions first
- **Test:** Verify that the model follows your instructions

GPT-5 advantage: More reliable adherence to system instructions compared to previous models, especially for complex or multi-part instructions.




```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Example of a well-structured system message
system_message = """
You are a data science assistant with expertise in Python.
- Always provide code examples with explanations
- Focus on best practices and efficiency
- When uncertain, acknowledge limitations
"""

response = client.chat.completions.create(
```

Practice: Working with the OpenAI API

Let's practice using the OpenAI API with GPT-5:

Exercise 1: Basic API Request

Create a simple API request to GPT-5 asking for a summary of a news article.

Exercise 2: System Messages

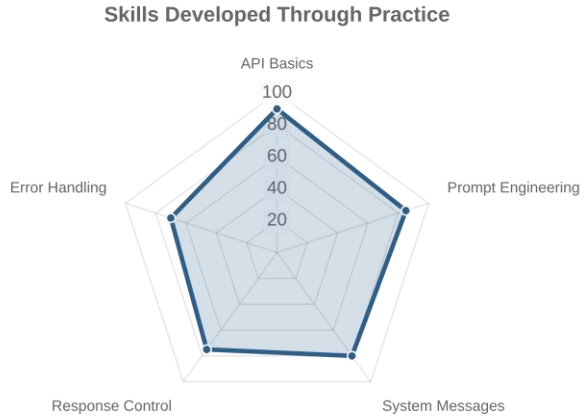
Use a system message to instruct GPT-5 to respond in the style of a specific author.

Exercise 3: Few-shot Prompting

Create a few-shot prompt to extract specific information from product reviews.

Exercise 4: Response Parameters

Experiment with verbosity and reasoning_effort parameters to control GPT-5's responses.



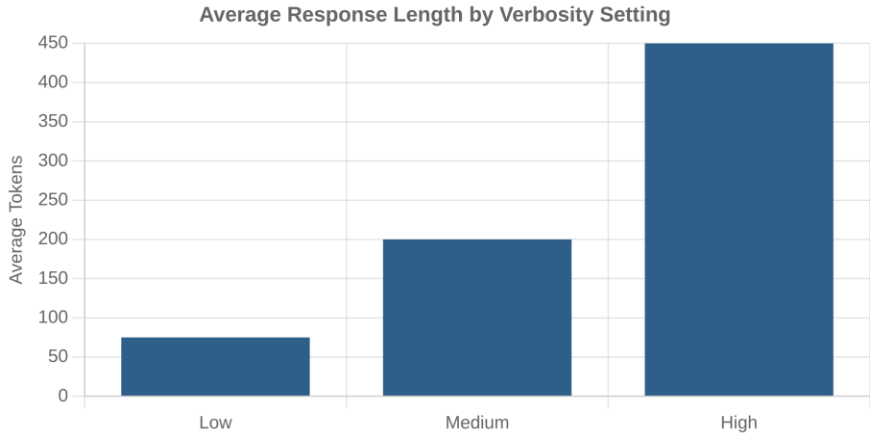

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Exercise 1: Basic API Request
response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user", "content": "Summarize this news article: [article text]"}]
)

# Print the response
```

Controlling Response Length

GPT-5 offers improved control over response length:

- Use the **verbosity** parameter to control response length
- **low**: Brief, concise responses
- **medium** : Balanced responses (default)
- **high**: Detailed, comprehensive responses

> **Best practice:** Match verbosity to your use case. Use "low" for summaries or quick answers, "high" for detailed explanations or educational content.



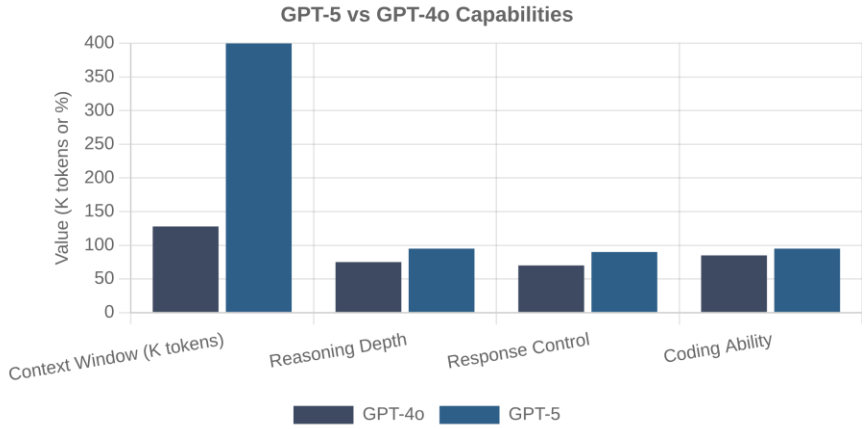

```python
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Control response length with verbosity
response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user", "content": "Explain quantum computing"}],
    verbosity="low"  # Options: "low", "medium", "high"
)
```

GPT-5 New Features

GPT-5 introduces several new parameters and capabilities:

- **Verbosity:** Control response length with "low", "medium", or "high" settings
- **Reasoning effort:** Control depth of reasoning with "minimal", "balanced", or "thorough"
- **Context window:** Expanded to 400,000 tokens (up from 128,000 in GPT-4o)
- **Improved coding:** Enhanced capabilities for complex programming tasks

Best practice: Use these new parameters to fine-tune responses for your specific use case.



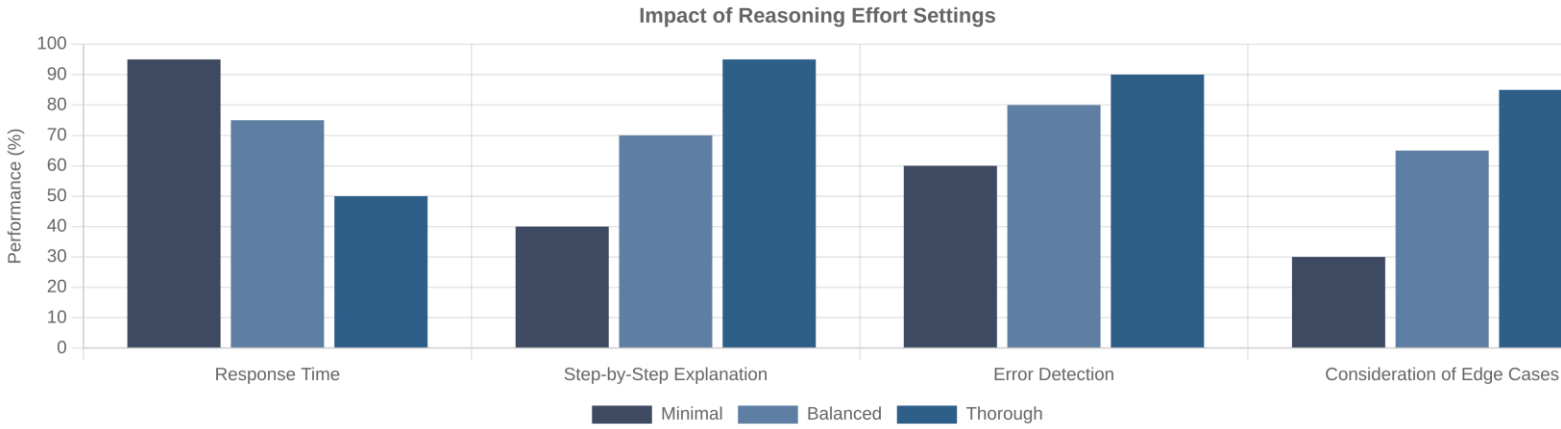

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

response = client.chat.completions.create(
  model="gpt-5",
  messages=[{"role": "user", "content": "Explain quantum computing"}],
  verbosity="medium",
  reasoning={"effort": "thorough"}
)
```

Controlling Reasoning Depth

GPT-5 introduces the reasoning_effort parameter:

- **minimal:** Quick, surface-level reasoning
- **balanced:** Moderate depth of reasoning (default)
- **thorough:** Deep, comprehensive reasoning

Affects how deeply the model thinks through problems



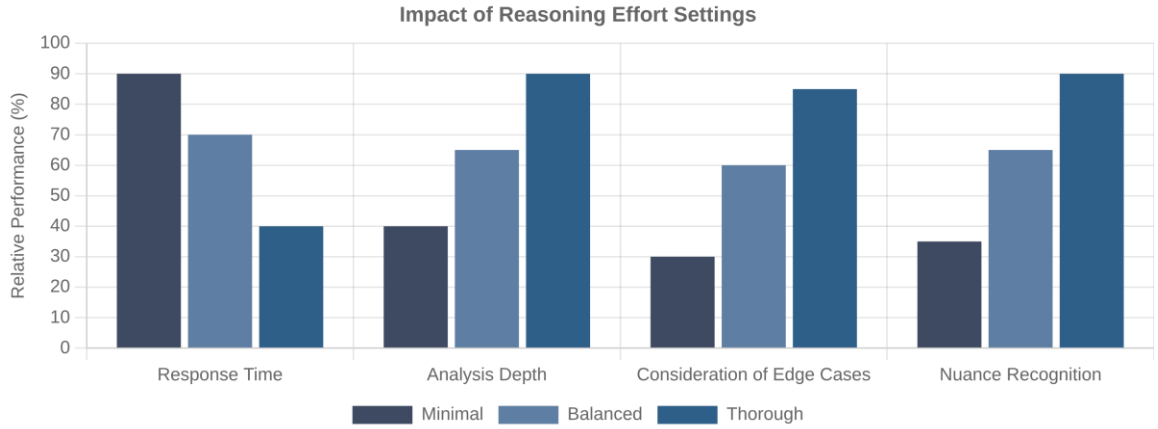

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Control reasoning depth
response = client.chat.completions.create(
```

Controlling Reasoning Effort

GPT-5 introduces the reasoning_effort parameter:

- **minimal:** Quick, straightforward responses with limited analysis
- **balanced:** Moderate depth of reasoning (default)
- **thorough:** In-depth analysis with careful consideration of nuances
- Affects quality and depth of reasoning, not just response length

Best practice: Use "thorough" for complex problems requiring careful analysis, "minimal" for simple queries where speed is important.



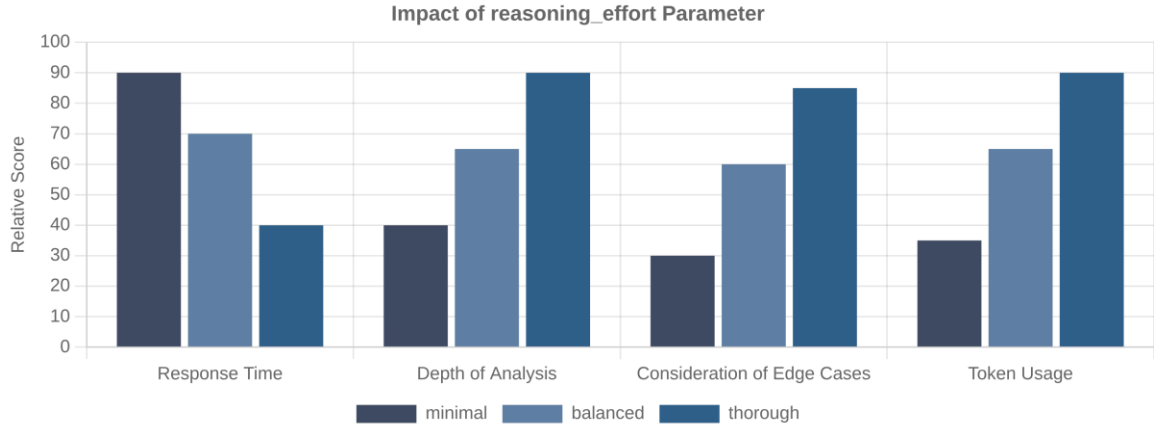

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Control reasoning depth
response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user", "content": "Analyze the impact of AI on job markets"}],
    reasoning={"effort": "thorough"}  # Options: "minimal", "balanced", "thorough"
)
```

Controlling Reasoning Depth

GPT-5 introduces the reasoning_effort parameter to control depth of analysis:

- **minimal:** Quick, surface-level responses with basic reasoning
- **balanced:** Moderate depth of analysis (default setting)
- **thorough:** In-depth analysis with comprehensive reasoning

> **Best practice:** Use "minimal" for simple tasks, "thorough" for complex problems requiring detailed analysis, and "balanced" for general use.



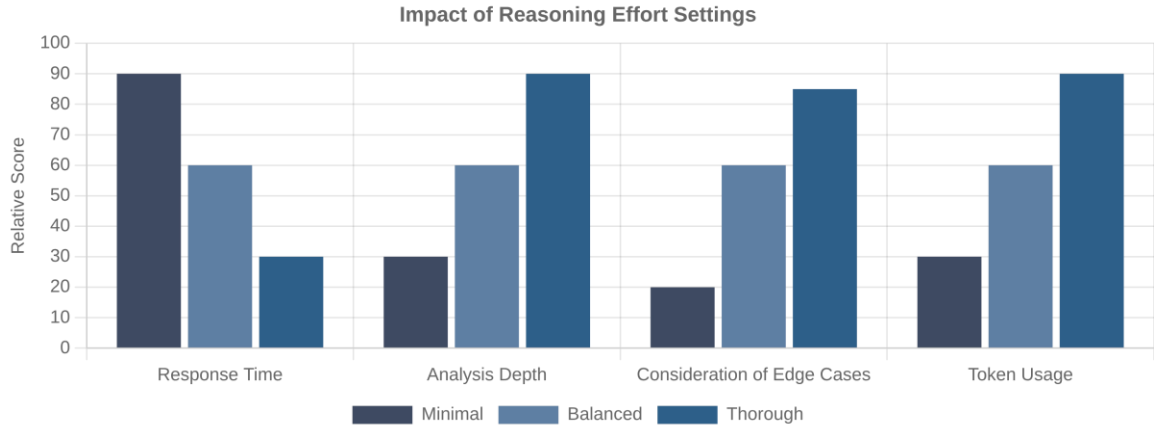

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Control reasoning depth
response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user",
               "content": "Analyze the impact of AI on healthcare"}],
    reasoning={"effort": "thorough"}  # Options: "minimal", "balanced", "thorough"
)
```

Controlling Reasoning Depth

GPT-5 introduces the reasoning_effort parameter to control depth of analysis:

- **minimal:** Quick, surface-level responses with limited analysis
- **balanced:** Moderate depth of reasoning (default setting)
- **thorough:** In-depth analysis with careful consideration of nuances

Best practice: Use "minimal" for simple tasks or when speed is important, "thorough" for complex problems requiring careful analysis, and "balanced" for everyday tasks.



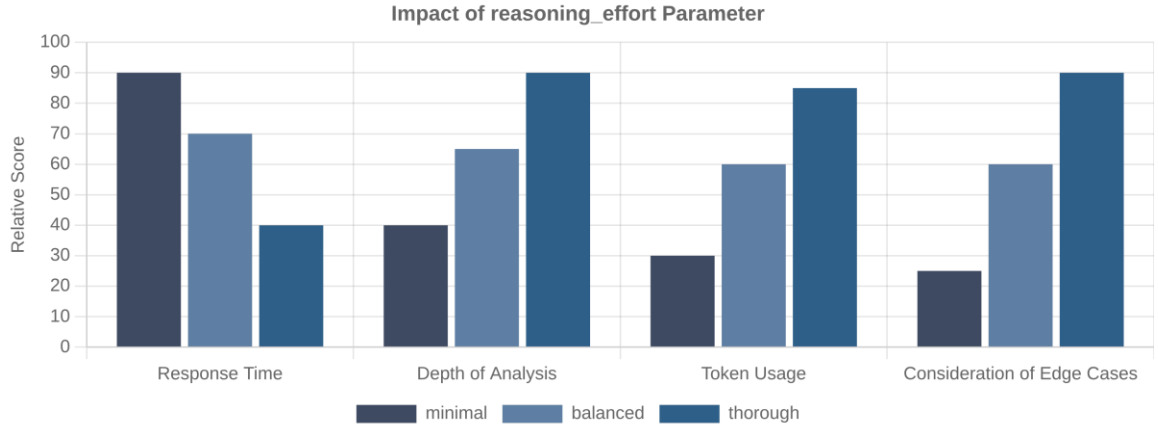

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Control reasoning depth
response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user",
               "content": "Analyze the impact of AI on education"}],
    reasoning={"effort": "thorough"}  # Options: "minimal", "balanced", "thorough"
)
```

Controlling Reasoning Depth

GPT-5 introduces the reasoning_effort parameter to control depth of analysis:

- **minimal:** Quick, surface-level responses with basic reasoning
- **balanced:** Moderate depth of analysis (default setting)
- **thorough:** In-depth analysis with comprehensive reasoning

> **Best practice:** Use "minimal" for simple tasks or quick responses, "thorough" for complex problems requiring detailed analysis or explanation.



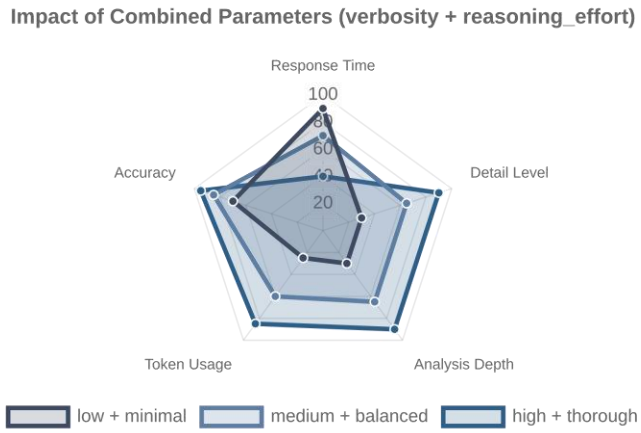

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Control reasoning depth
response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user",
               "content": "Analyze the impact of AI on healthcare"}],
    reasoning={"effort": "thorough"}  # Options: "minimal", "balanced", "thorough"
)
```

Combining GPT-5 Parameters

GPT-5 parameters can be combined for precise control:

- **verbosity + reasoning_effort:** Control both length and depth
- **low verbosity + minimal reasoning:** Quick, concise responses
- **high verbosity + thorough reasoning:** Comprehensive analysis
- **medium verbosity + balanced reasoning:** Default behavior

> **Best practice:** Match parameter combinations to your specific use case requirements.



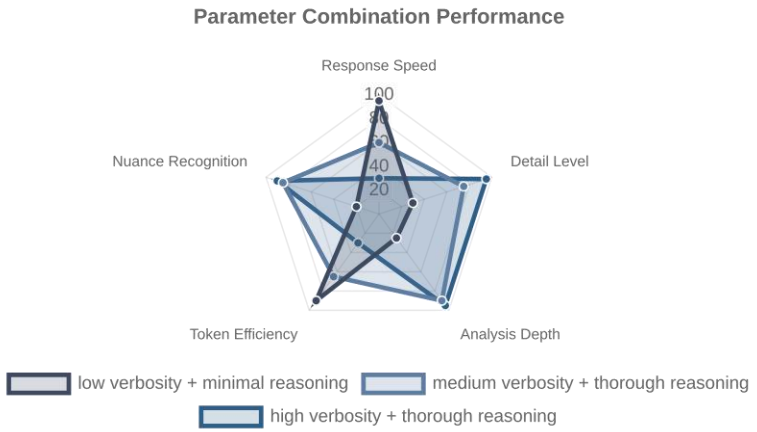

```python
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Combining parameters for precise control
response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user", "content": "Explain quantum computing"}],
    verbosity="high",
    reasoning={"effort": "thorough"}
)
```

Combining GPT-5 Parameters

GPT-5 parameters can be combined for fine-tuned control:

- **verbosity + reasoning_effort:** Control both length and depth
- **low verbosity + minimal reasoning:** Quick, concise responses
- **high verbosity + thorough reasoning:** Comprehensive analysis
- **medium verbosity + thorough reasoning:** Detailed but concise

Best practice: Tailor parameter combinations to your specific use case for optimal results.



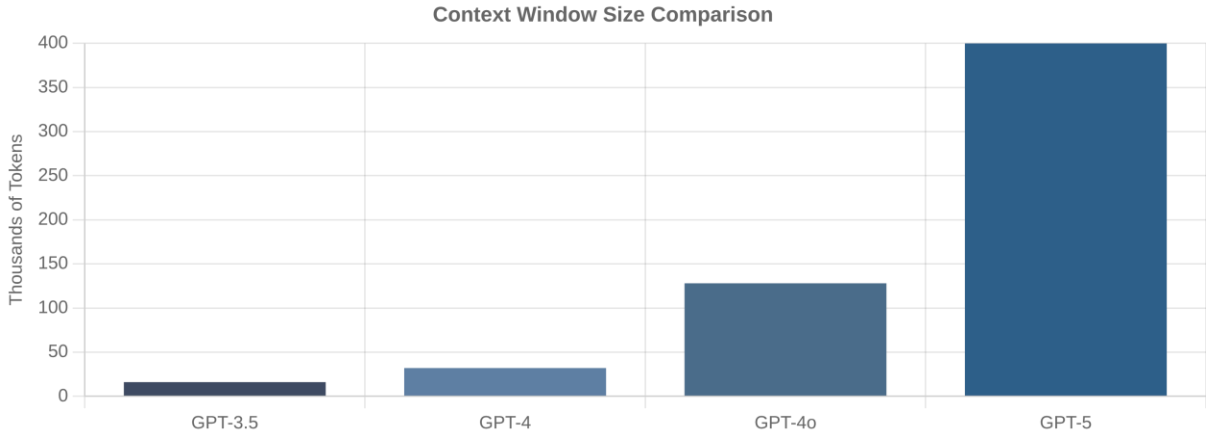

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Combining parameters for fine-tuned control
response = client.chat.completions.create(
    model="gpt-5",
    messages=[{"role": "user", "content": "Explain quantum computing"}],
    verbosity="medium",
    reasoning={"effort": "thorough"}
)
```

GPT-5 Context Window

GPT-5 features a significantly expanded context window:

- **400,000 tokens** - Approximately 300,000 words or 750 pages
- Process entire books, lengthy documents, or large codebases
- Maintain longer conversation history for more coherent interactions
- Analyze and generate complex code with full context awareness

Best practice: For long inputs, structure your content with clear sections and use system messages to guide the model's focus.



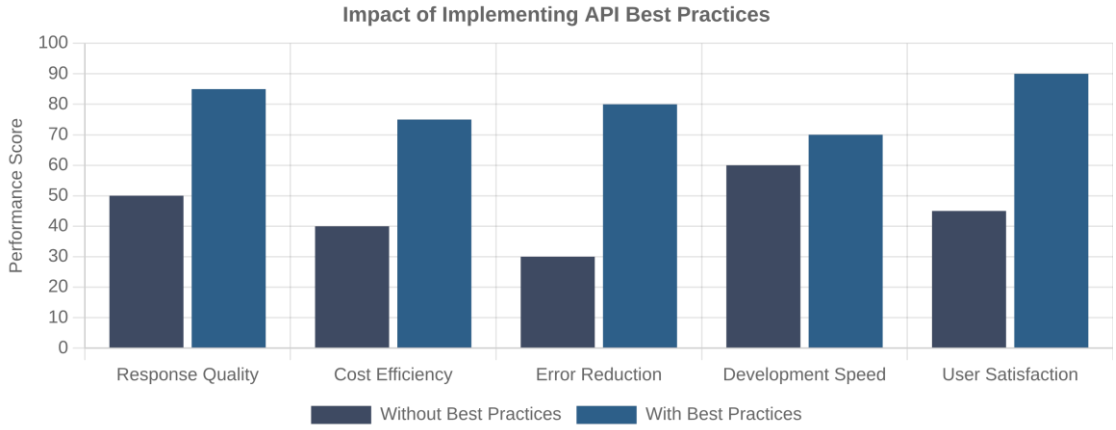

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Example: Processing a large document
with open("large_document.txt", "r") as f:
    document_text = f.read()

response = client.chat.completions.create(
    model="gpt-5",
    messages=[
        {"role": "system", "content": "Summarize the key points from this document."},
```

GPT-5 API Best Practices

Optimize your GPT-5 API usage with these best practices:

- **Be specific in prompts:** Clear instructions yield better results
- **Use appropriate parameters:** Match verbosity and reasoning_effort to your use case
- **Leverage system messages:** Define role and constraints clearly
- **Handle errors gracefully:** Implement robust error handling
- **Monitor token usage:** Track and optimize for cost efficiency

> **Remember:** The quality of your inputs significantly impacts the quality of outputs. Invest time in crafting effective prompts and system messages.



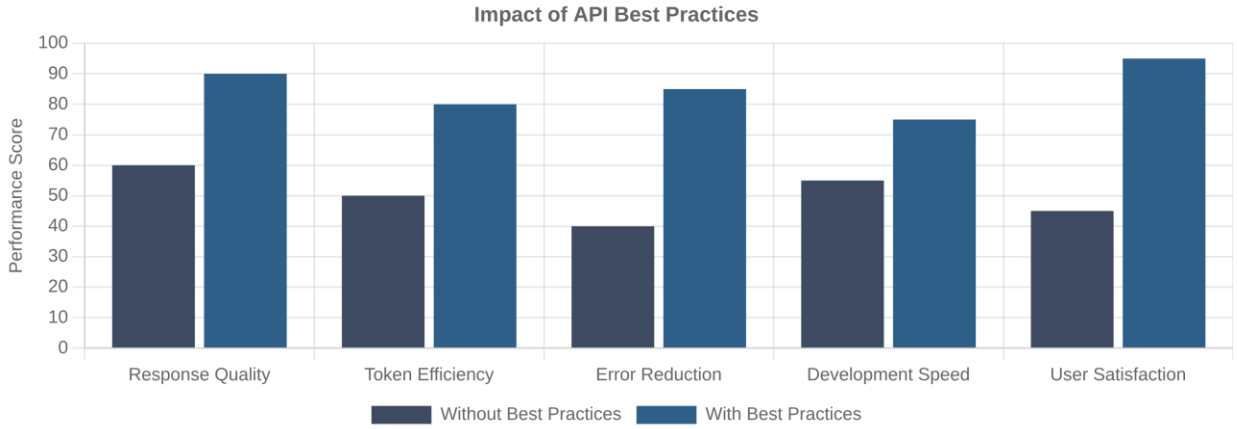

```python
import os
from openai import OpenAI
from tenacity import retry, stop_after_attempt, wait_random_exponential

# Best practice: Implement retry logic
@retry(wait=wait_random_exponential(min=1, max=60), stop=stop_after_attempt(6))
def get_completion(prompt, system_message=None):
    client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

    messages = []
    if system_message:
        messages.append({"role": "system", "content": system_message})
```

GPT-5 API Best Practices

Optimize your GPT-5 API usage with these best practices:

- **Be specific in prompts:** Clear instructions yield better results
- **Use appropriate parameters:** Match verbosity and reasoning_effort to your use case
- **Leverage system messages:** Define role and constraints clearly
- **Maintain conversation context:** Include relevant message history
- **Implement error handling:** Gracefully handle API errors and rate limits

Remember: GPT-5 is more capable but still requires thoughtful prompting and parameter selection for optimal results.



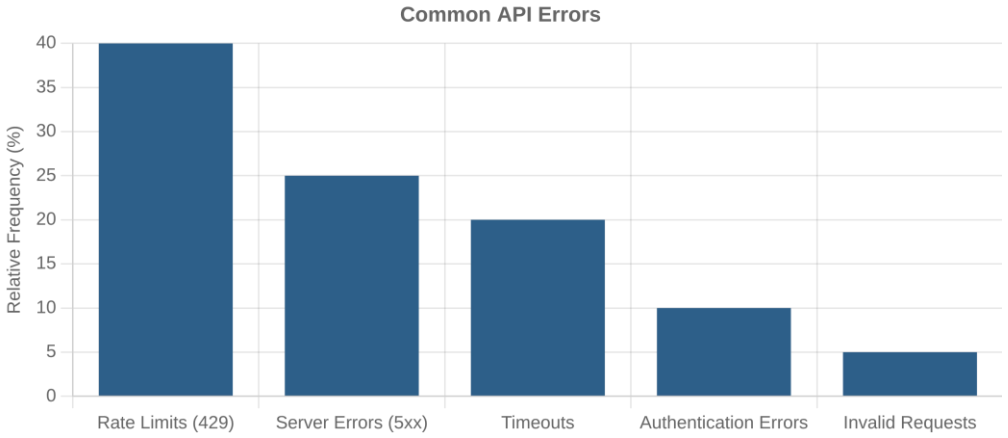

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Example of best practices implementation
try:
    response = client.chat.completions.create(
        model="gpt-5",
        messages=[
            {"role": "system", "content": "You are a helpful data science assistant."},
            {"role": "user", "content": "Analyze this dataset and identify key trends."}
        ],
```

Error Handling with the GPT-5 API

Implement robust error handling for reliable applications:

- **Rate limits:** Handle 429 errors with exponential backoff
- **Server errors:** Implement retries for 5xx errors
- **Timeouts:** Set appropriate request timeouts
- **Retry strategy:** Use libraries like tenacity for Python

Best practice: Always implement proper error handling to ensure your application remains robust even when the API encounters issues.

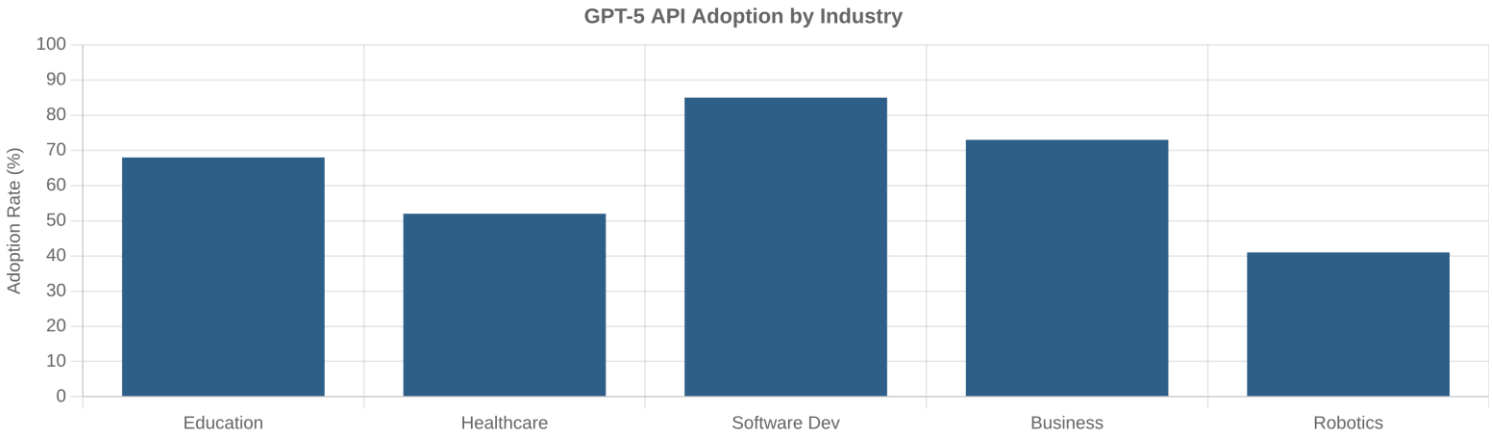



```
import os
from openai import OpenAI
from tenacity import retry, stop_after_attempt, wait_random_exponential

# Implement retry logic with exponential backoff
@retry(wait=wait_random_exponential(min=1, max=60),
    stop=stop_after_attempt(6))
def get_completion(prompt):
  client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

    try:
    response = client.chat.completions.create(
      model="gpt-5",
```

GPT-5 API Applications

GPT-5 enables powerful applications across industries:

- **Education:** Personalized tutoring, content generation, and assessment
- **Healthcare:** Medical documentation, research assistance, patient support
- **Software Development:** Code generation,

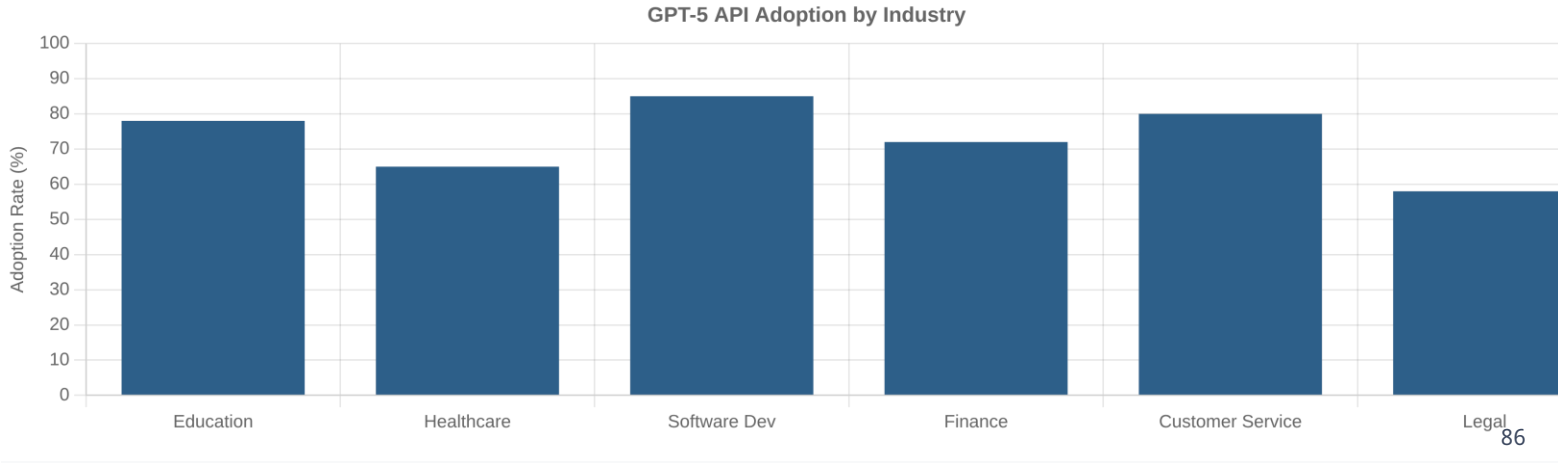



```
import os
from openai import OpenAI
```

GPT-5 API Applications

GPT-5 enables powerful applications across industries:

- **Education:** Personalized tutoring and adaptive learning systems
- **Healthcare:** Medical research assistance and patient communication
- **Software Development:** Code generation,



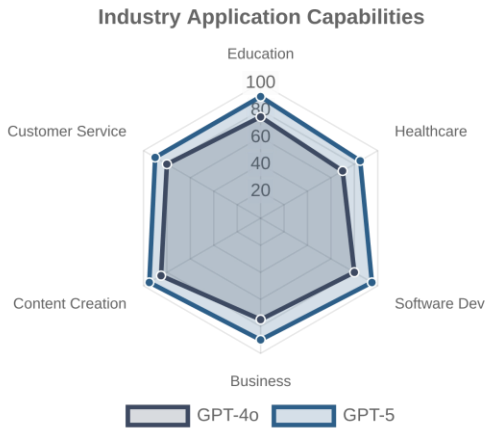
Coding a Conversation

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Initialize conversation with system message
messages = [
    {"role": "system", "content": "You are a helpful and knowledgeable assistant."}
]

# Define a list of user questions for the conversation
user_qs = [
    "Why is Python so popular among data scientists?",
    "Can you summarize your previous response in one sentence?"
]

# Loop through each question to build a conversation
for question in user_qs:
    # Add the user question to messages
    messages.append({"role": "user", "content": question})

    # Get response from the model
    response = client.chat.completions.create(
        model="gpt-5",
        messages=messages
    )

    # Extract the assistant's message
```

GPT-5 API Applications

GPT-5 enables powerful applications across industries:

- **Education:** Personalized tutoring, content generation, and assessment
- **Healthcare:** Medical documentation, research assistance, and patient support
- **Software Development:** Code

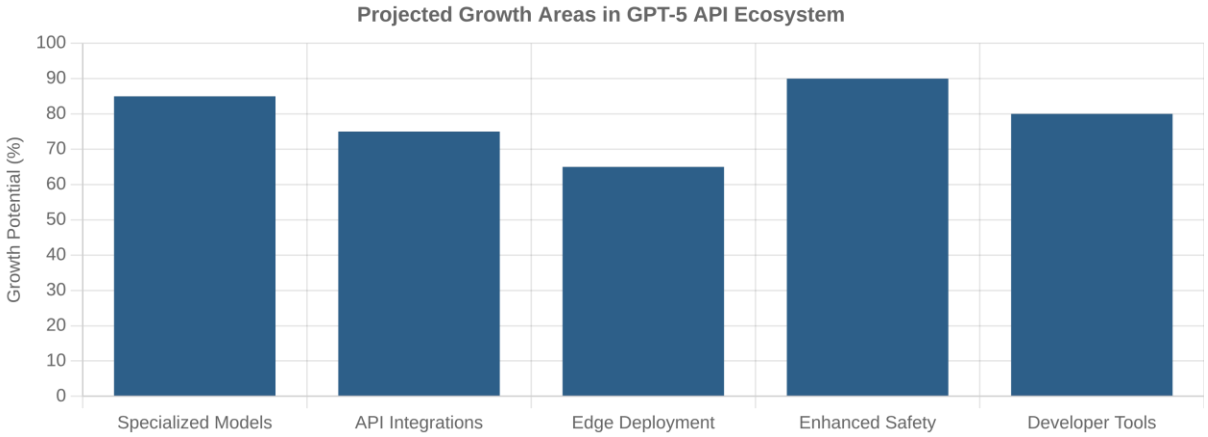


Future Trends: GPT-5 API

Key trends to watch in the GPT-5 API ecosystem:

- **Specialized models:** Domain-specific versions of GPT-5 for healthcare, legal, finance
- **API integrations:** Deeper integration with other services and platforms
- **Edge deployment:** Optimized versions for mobile and edge devices
- **Enhanced safety:** More robust content filtering and safety measures
- **Developer tools:** Improved SDKs, monitoring, and debugging capabilities

Key insight: The future of GPT-5 API development will focus on specialization, integration, and accessibility.



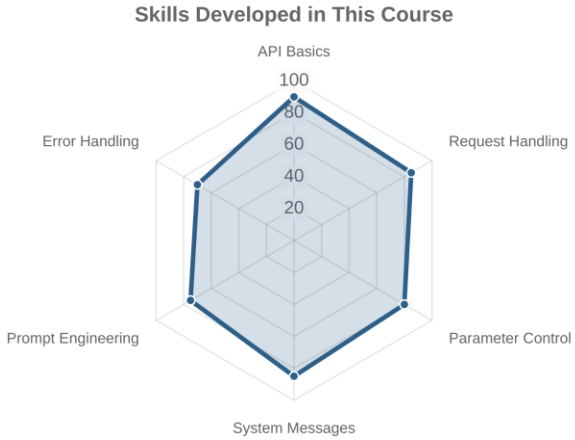

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Example: Future specialized healthcare model
response = client.chat.completions.create(
    model="gpt-5-medical",  # Hypothetical future model
    messages=[
        {"role": "system", "content": "You are a specialized medical assistant."},
        {"role": "user", "content": "Summarize the latest research on mRNA vaccines."}
    ],
    verbosity="medium"
)
```

Course Summary

Key concepts covered in this course:

- **OpenAI API Basics:** Understanding the API structure and authentication
- **Making API Requests:** Creating and sending requests to the GPT-5 API
- **Parameter Control:** Using verbosity and reasoning_effort parameters
- **System Messages:** Controlling model behavior with system instructions
- **Prompt Engineering:** Creating effective prompts for optimal results

Next steps: Apply these concepts to build your own AI-powered applications using the OpenAI API.



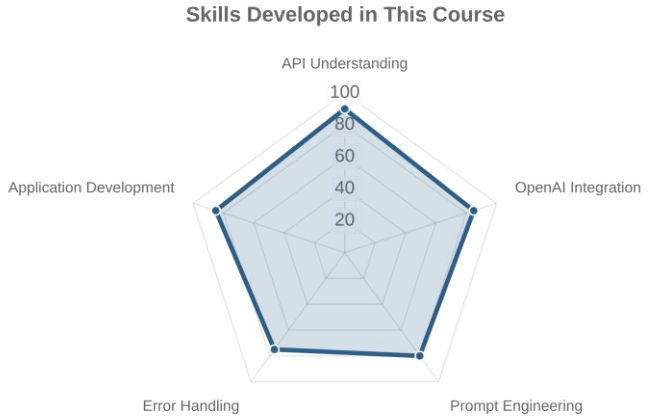

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Example combining multiple concepts
response = client.chat.completions.create(
    model="gpt-5",
    messages=[
        {"role": "system", "content": "You are a helpful AI assistant."},
        {"role": "user", "content": "Summarize the key benefits of using the OpenAI API."}
    ],
    verbosity="medium",
)
```

Course Summary

Key takeaways from this course:

- **Understanding APIs:** How APIs work as interfaces between applications
- **OpenAI API:** Structure, authentication, and making requests
- **GPT-5 Features:** Verbosity, reasoning_effort, and expanded context window
- **Practical Applications:** Text generation, summarization, and editing
- **Best Practices:** Error handling, prompt engineering, and system messages

Next steps: Apply these skills to build your own AI-powered applications using the OpenAI API.




```
import os
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# Example combining multiple concepts from the course
response = client.chat.completions.create(
    model="gpt-5",
    messages=[
        {"role": "system", "content": "You are a helpful AI assistant."},
        {"role": "user", "content": "Summarize the key points about APIs."}
    ],
    verbosity="medium",
```