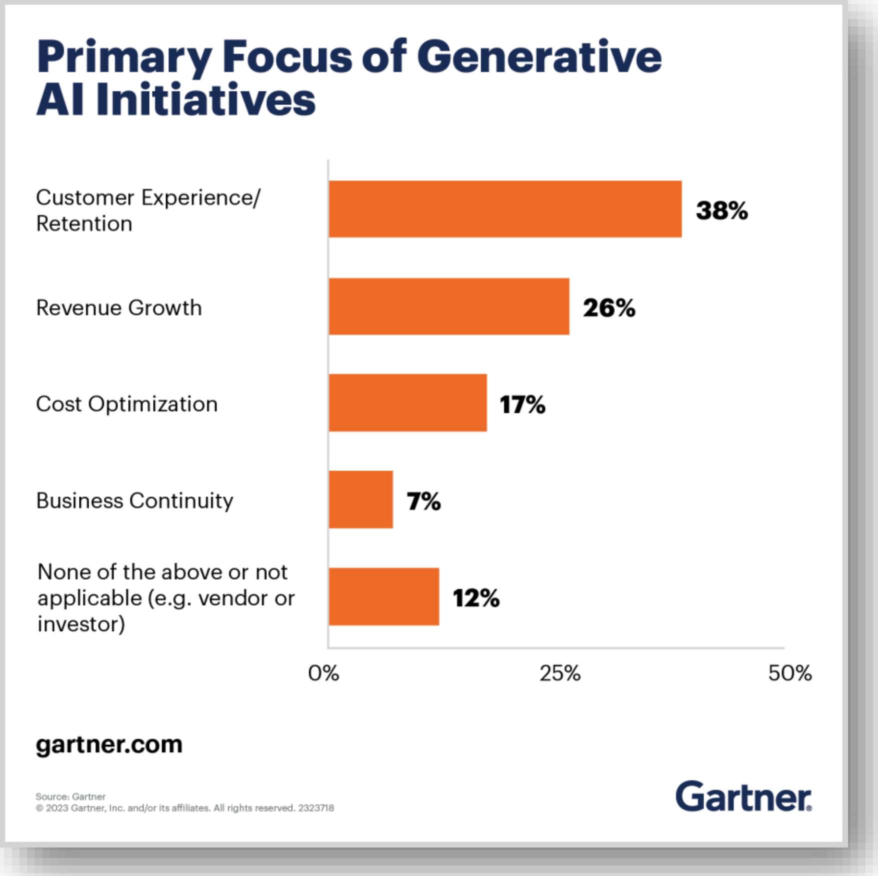
TECHORAMA

Review my session

Make AI Come to Life with OpenAI

Jeff Prosise

Why Generative AI?



Primary Focus of Generative AI Initiatives

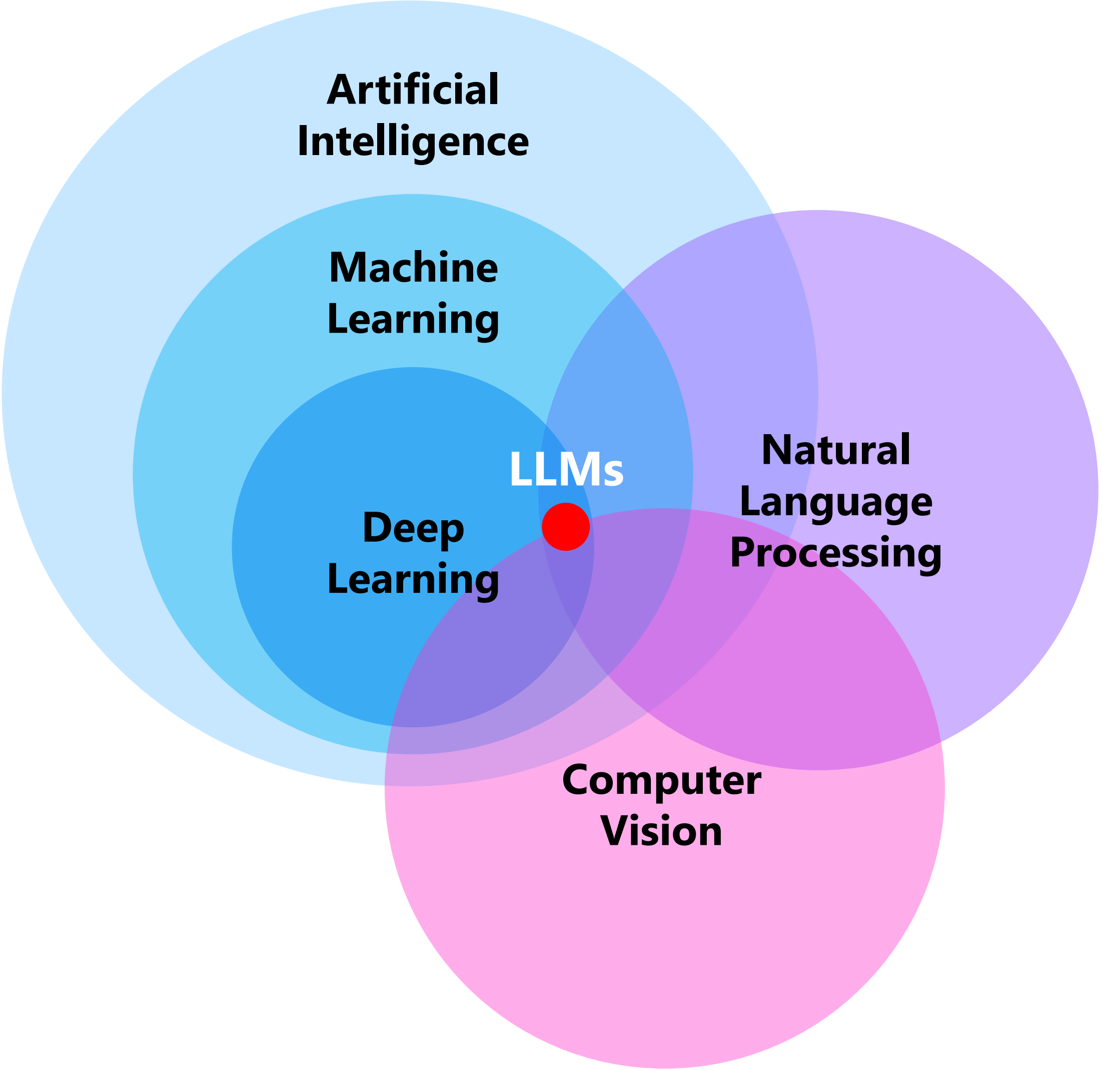
Source: Gartner
© 2023 Gartner, Inc. and/or its affiliates. All rights reserved. 2323718

> “Gartner sees generative AI becoming a **general-purpose technology** with an impact similar to that of the **steam engine, electricity and the internet**. The hype will subside as the reality of implementation sets in, but the **impact of generative AI will grow** as people and enterprises discover more innovative applications for the technology in daily work and life.”

https://www.gartner.com/en/topics/generative-ai

Large Language Models (LLMs)

- Deep-learning models (neural networks) that generate text
  - All accept text as input
    - "Write a poem about deep learning"
  - Multimodal LLMs (MLLMs) accept text, images, and sometimes audio and video as input
    - "Describe what you see on this map"
- Billions of parameters trained on trillions of words



Generating Text

```
Write a short story that begins with "It was a dark and stormy night."
```

```
It was a dark and stormy night. The wind was howling through the streets, making it
difficult for anyone to stay on their feet. The rain was coming down in sheets,
drenching anyone who dared to step outside. Inside, a young woman sat huddled in a
corner of her small apartment, listening to the storm rage on outside.

She had always been afraid of storms, ever since she was a child. But tonight, there was
something different about this one. It seemed to be getting closer and closer, and she
could feel the panic rising in her chest. She tried to shake it off, telling herself
that it was just a normal storm, but she couldn't shake the feeling that something was
coming.

...
```

Translating Text

```
Translate the following into French: "What time does the train leave for Paris?"
```

```
"À quelle heure part le train pour Paris?"
```

Generating Code

```
Create a Python function that accepts an array of numbers as input, bubble sorts the
numbers, and returns a sorted array
```

```python
def bubble_sort(numbers):
    # Get the length of the array
    n = len(numbers)

    # Traverse through all the array elements
    for i in range(n-1):
        # Last i elements are already in place
        for j in range(0, n-i-1):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if numbers[j] > numbers[j+1]:
                numbers[j], numbers[j+1] = numbers[j+1], numbers[j]

    return numbers
```

Translating Code

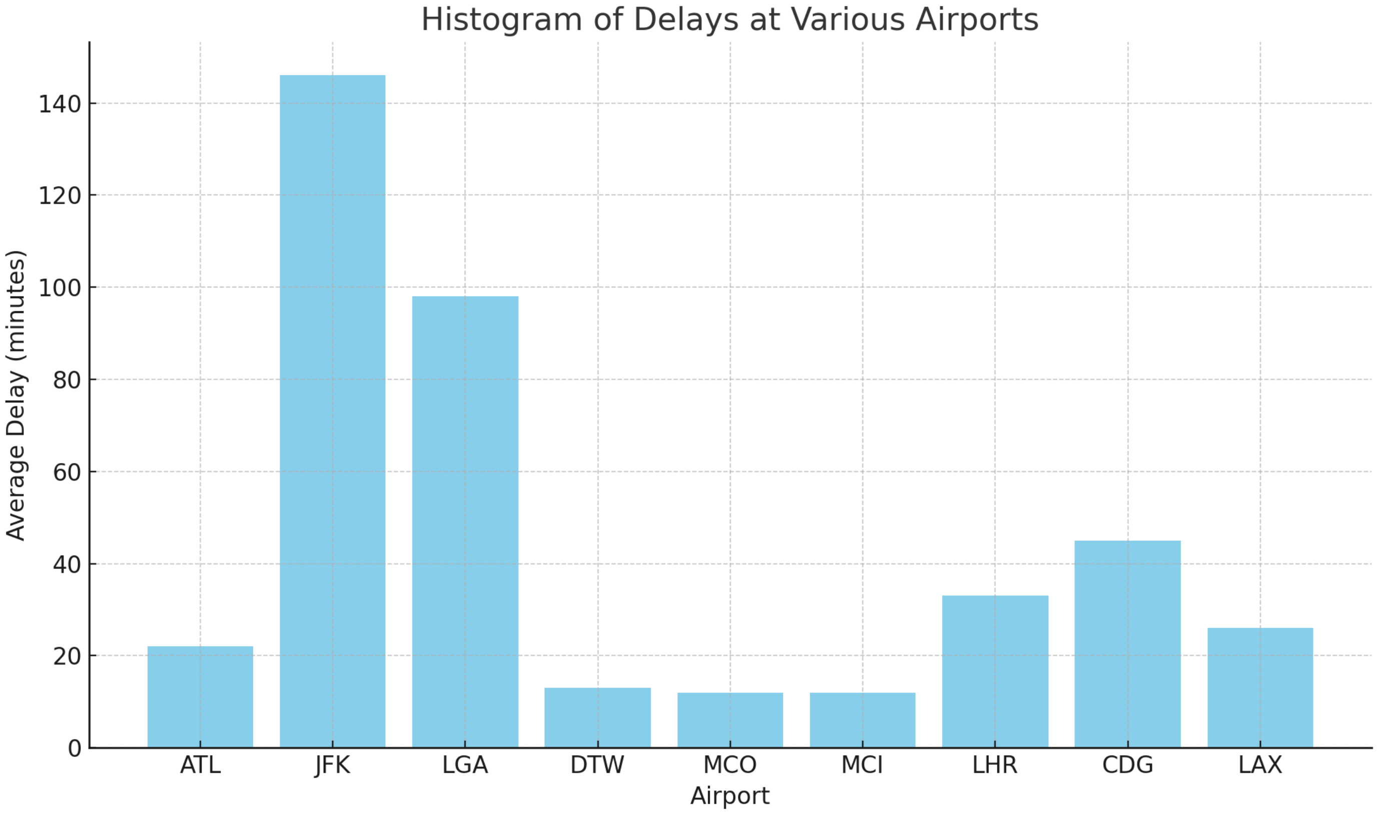
```
Translate the bubble_sort function into FORTRAN
```

```
FUNCTION bubble_sort(arr, n) RESULT(result)
   ...
  DO i = 1, n
    DO j = 1, n-i-1
      IF (arr(j) > arr(j+1)) THEN
        temp = arr(j)
        arr(j) = arr(j+1)
        arr(j+1) = temp
      END IF
    END DO
  END DO

  result = arr
END FUNCTION bubble_sort
```

Generating Charts and Graphs

```
Generate a histogram from the data below showing delays at various airports
```

Airport	Average Delay
ATL	22
JFK	146
LGA	98
DTW	13
MCO	12
MCI	12
LHR	33
CDG	45
LAX	26



Analyzing Charts

```
Explain the chart on the preceding slide
```

```
This chart is a histogram depicting the average delays (in minutes) at various airports.
Here are some key observations:

- JFK has the highest average delay, approaching 140 minutes.
- LGA follows with a slightly lower average delay, around 100 minutes.
- Most other airports, such as ATL, DTW, MCO, and MCI, have much lower average delays,
  typically around 20 minutes or lower.
- CDG and LHR, two major international airports, show moderate delays (CDG with around
  70 minutes and LHR with about 50 minutes)

The light blue bars indicate the magnitude of the delay for each airport, and the title
of the chart indicates it's a "Histogram of Delays at Various Airports."
```

Demo

LLMs in Action

OpenAI GPT Models

- Available by REST API with an OpenAI account
- SDKs available for Python, Node.js, .NET, Java, Rust, and more

GPT-3.5 Turbo

Groundbreaking LLM with **175B parameters**. Fast and cost-effective. Also known as **ChatGPT**.

GPT-4o

Multimodal model ("o" for "omni") with **128K context window**. Knowledge cutoff of October 2023.

GPT-4o mini

Distilled version of GPT-4o with **128K context window**. Costs more than **30 times less than GPT-4o**.

GPT-4o1

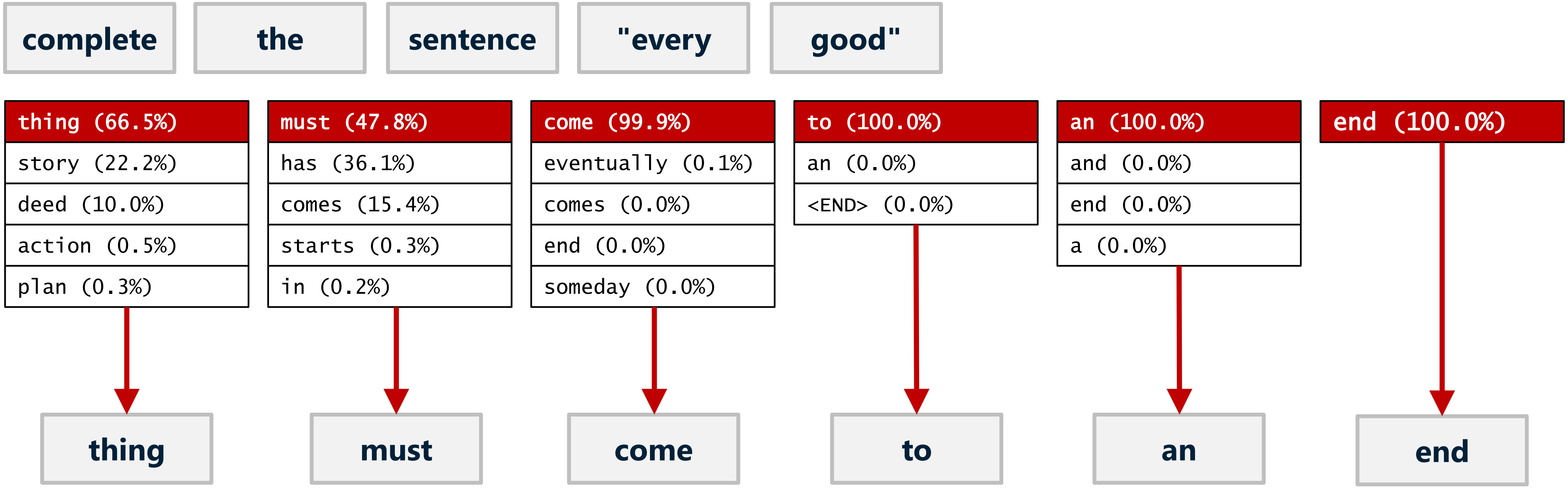
Latest GPT model capable of **complex problem solving**. Currently in preview.

GPT-4o1 mini

Distilled version of GPT-4o1. Currently in preview.

How GPT Models Work

```
temperature=0.0
```



complete | the | sentence | "every | good"

thing (66.5%)
story (22.2%)
deed (10.0%)
action (0.5%)
plan (0.3%)

must (47.8%)
has (36.1%)
comes (15.4%)
starts (0.3%)
in (0.2%)

come (99.9%)
eventually (0.1%)
comes (0.0%)
end (0.0%)
someday (0.0%)

to (100.0%)
an (0.0%)
<END> (0.0%)

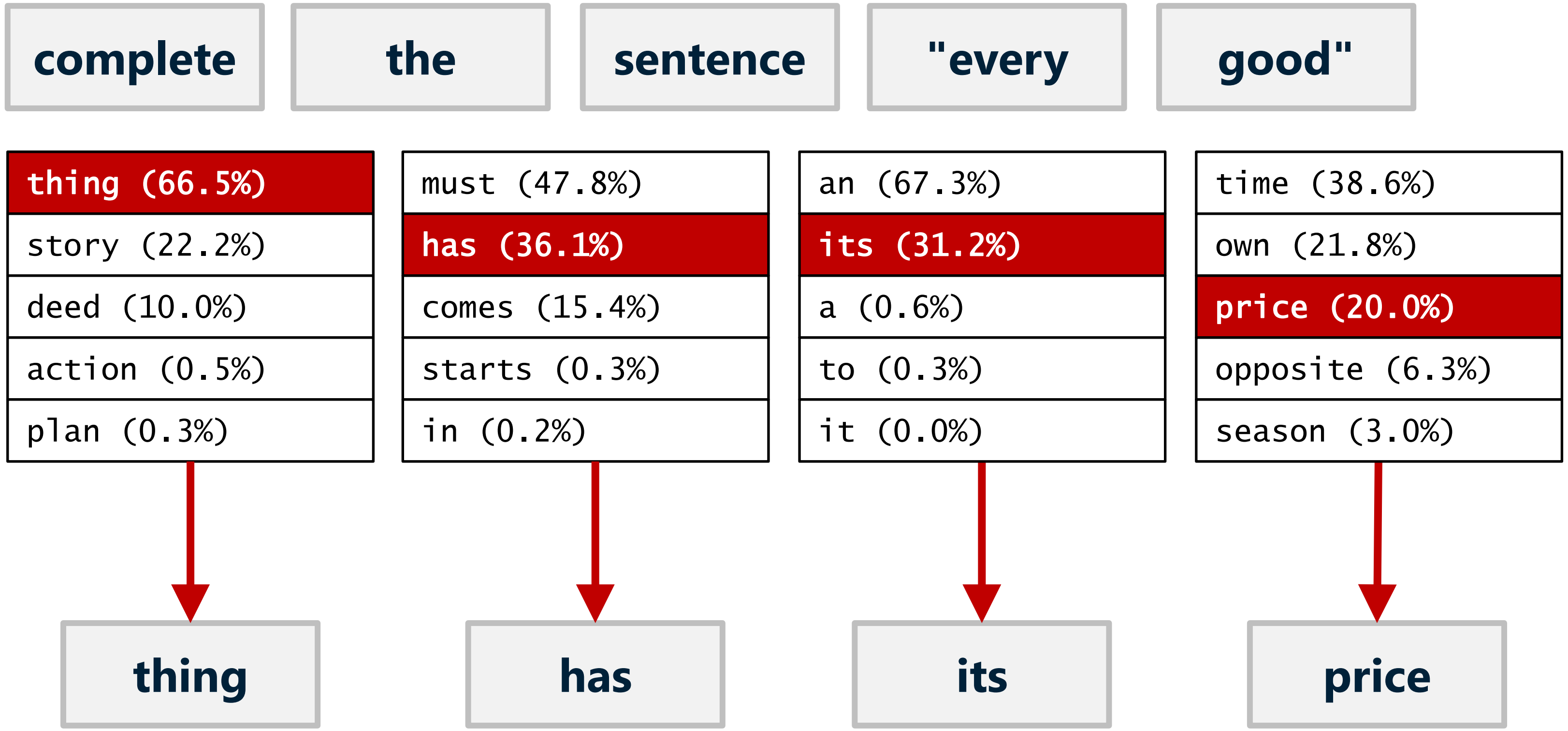
an (100.0%)
and (0.0%)
end (0.0%)
a (0.0%)

end (100.0%)

thing | must | come | to | an | end

How GPT Models Work, Cont.

```
temperature=0.7
```



complete the sentence "every good"

thing (66.5%)	must (47.8%)	an (67.3%)	time (38.6%)
story (22.2%)	**has (36.1%)**	**its (31.2%)**	own (21.8%)
deed (10.0%)	comes (15.4%)	a (0.6%)	**price (20.0%)**
action (0.5%)	starts (0.3%)	to (0.3%)	opposite (6.3%)
plan (0.3%)	in (0.2%)	it (0.0%)	season (3.0%)

thing has its price

Generating Text with GPT-4o

```python
from openai import OpenAI

client = OpenAI(api_key='OPENAI_API_KEY') # Or use OPENAI_API_KEY environment variable
messages = [{ 'role': 'user', 'content': 'Why is the sky blue?' }]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages
)

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

Generating Text Asynchronously

```
from openai import AsyncOpenAI

client = AsyncOpenAI(api_key='OPENAI_API_KEY')
messages = [{ 'role': 'user', 'content': 'Why is the sky blue?' }]

response = await client.chat.completions.create(
    model='gpt-4o',
    messages=messages
)

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

Streaming the Response

```
client = OpenAI(api_key='OPENAI_API_KEY')
messages = [{ 'role': 'user', 'content': 'Why is the sky blue?' }]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages,
    stream=True
)

for chunk in response:
    content = chunk.choices[0].delta.content
    if content is not None:
        print(content, end='')
```

Specifying the Temperature

```
client = OpenAI(api_key='OPENAI_API_KEY')
messages = [{ 'role': 'user', 'content': 'Why is the sky blue?' }]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages,
    temperature=0.2
)

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

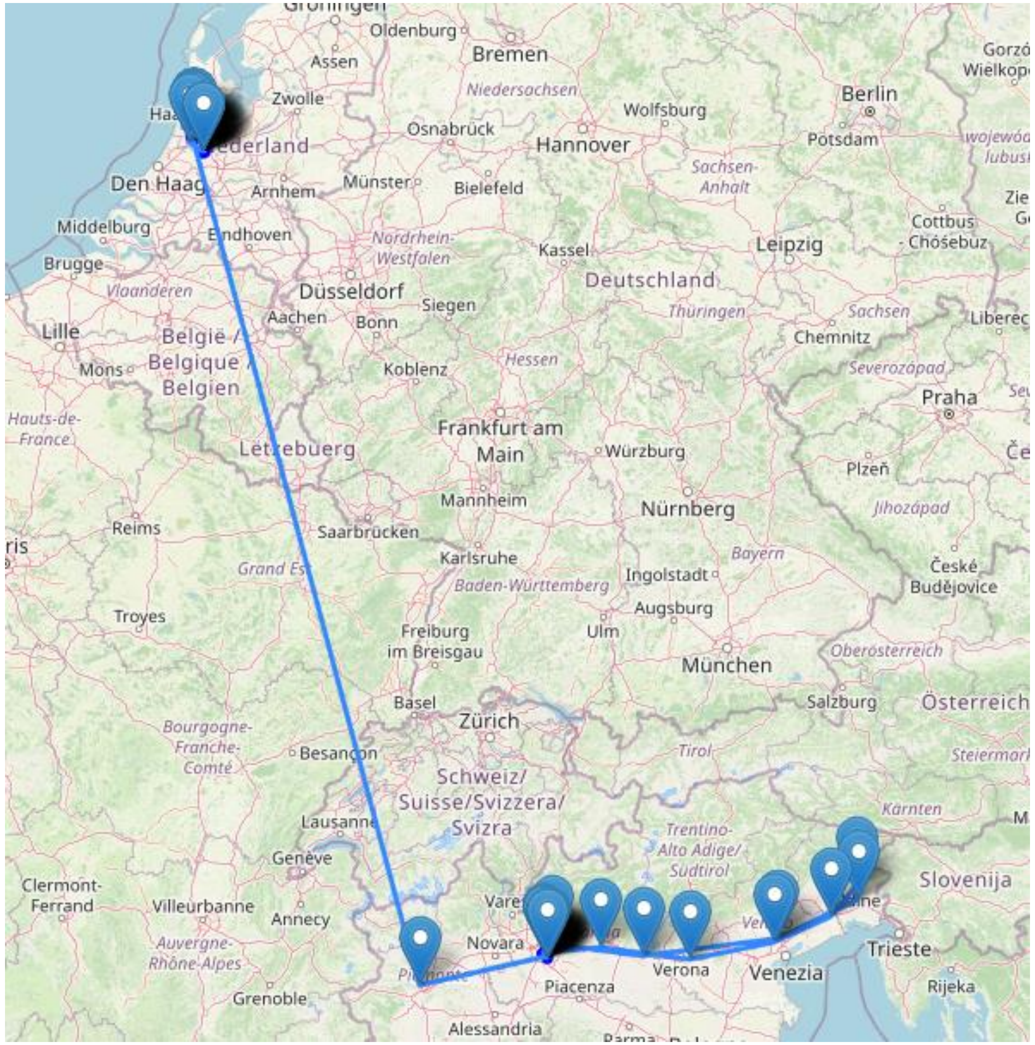
Generating JSON Output

```
prompt = '''
    Which books comprise the Harry Potter series? Please respond
    in JSON using the following format for each book:
    {
        "title": "Harry Potter and the Prisoner of Azkaban",
        "year": 1999
    }
    '''

client = OpenAI(api_key='OPENAI_API_KEY')
messages = [{ 'role': 'user', 'content': prompt }]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages,
    response_format={ 'type': 'json_object' }
)

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

Demo

Text Generation

Multimodal LLMs (MLLMs)



Describe the shipment plotted on this map.

The shipment began in the Netherlands and made its way to northeastern Italy using a combination of air and ground transportation.

The initial segment went by air, originating from a location near the coast of the Netherlands, possibly near Rotterdam, and ending in northeastern Italy, likely near Milan, as the shipment appears to fly across the Netherlands, over Germany, and into Italy.

The subsequent legs of the journey were traveled by road. The shipment passed through several cities in northern Italy, eventually reaching its destination near Venice.

The map does not specify exact highways, but the route within Italy likely involved major roads such as the A4 motorway, which connects Milan to Venice.

Submitting an Image to GPT-4o

```
messages = [{
    'role': 'user',
    'content': [
        { 'type': 'text', 'text': 'Describe what you see in this image' },
        { 'type': 'image_url', 'image_url': { 'url': 'IMAGE_URL' }} # Can be a data URL
    ]
}]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages
)
```

Submitting Multiple Images to GPT-4o

```
messages = [{
    'role': 'user',
    'content': [
        { 'type': 'text', 'text': 'Describe similarities between these images' },
        { 'type': 'image_url', 'image_url': { 'url': 'IMAGE_URL_1' }},
        { 'type': 'image_url', 'image_url': { 'url': 'IMAGE_URL_2' }}
    ]
}]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages
)
```

Demo

Multimodal LLMs

Code Generation

- Flagship LLMs such as GPT-4o, Gemini 1.5, and Llama 3 are adept at generating code in 40+ languages
  - Write code (convert natural language into code) and unit tests
  - Comment and explain code (convert code into natural language)
  - Convert code from one programming language to another
  - Complete code, edit/refactor code, and find bugs
- SLMs optimized for code generation are available, too
  - Code Llama, available in 7B, 13B, 34B, and 70B parameter versions
  - CodeGemma, available in 2B and 7B parameter versions

Generating Code

```
content = '''
    Create a Python function that accepts an array of numbers as
    input, bubble sorts the numbers, and returns a sorted array
    '''

messages = [{ 'role': 'user', 'content': content }]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages,
    temperature=0 # Use a low temperature setting for code generation
)
```

Converting Code to Natural Language

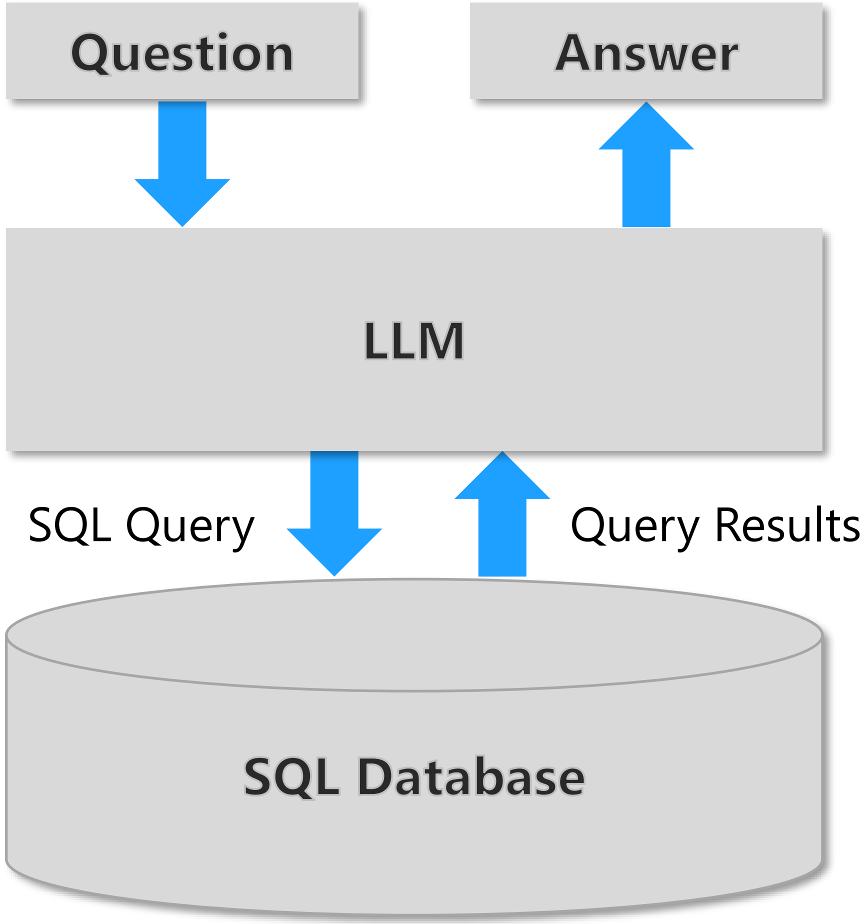
```
content = '''
    Explain what the following code does:

    def bubble_sort(arr):
        n = len(arr)
        for i in range(n):
            for j in range(0, n-i-1):
                if arr[j] > arr[j+1]:
                    arr[j], arr[j+1] = arr[j+1], arr[j]
        return arr
    '''

messages = [{ 'role': 'user', 'content' : content }]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages
)
```

Translating Code

```
content = '''
    Convert the following Python code into FORTRAN:

    def bubble_sort(arr):
        n = len(arr)
        for i in range(n):
            for j in range(0, n-i-1):
                if arr[j] > arr[j+1]:
                    arr[j], arr[j+1] = arr[j+1], arr[j]
        return arr
    '''

messages = [{ 'role': 'user', 'content' : content }]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages,
    temperature=0
)
```

SQL-Augmented Generation (SAG)

- Use LLM to implement a natural-language interface to databases
  - Convert natural-language questions into SQL queries
  - Execute the queries against the database
  - Use LLM to phrase query results
- Use CREATE TABLE statements in prompts to define schemas
- Optionally provide sample questions and resultant SQL queries in prompts



Generating a SQL Query

```python
prompt = f'''
    Assume the database has a table that is defined as follows:
    {table_definition}
    Generate a well-formed SQL query from the following prompt:
    {question}
    '''

response = client.chat.completions.create(
    model='gpt-4o',
    messages=[{ 'role': 'user', 'content': prompt }],
    temperature=0
)
```

Generating a Natural-Language Response

```
prompt = f'''
   Given the following question and query result, phrase the answer
   in terms that a human can understand.
   Question: {question}
   Result: {result}
   '''

response = client.chat.completions.create(
    model='gpt-4o',
    messages=[{ 'role': 'user', 'content': prompt }]
)
```

Demo

LLMs Over Databases

Answering Questions

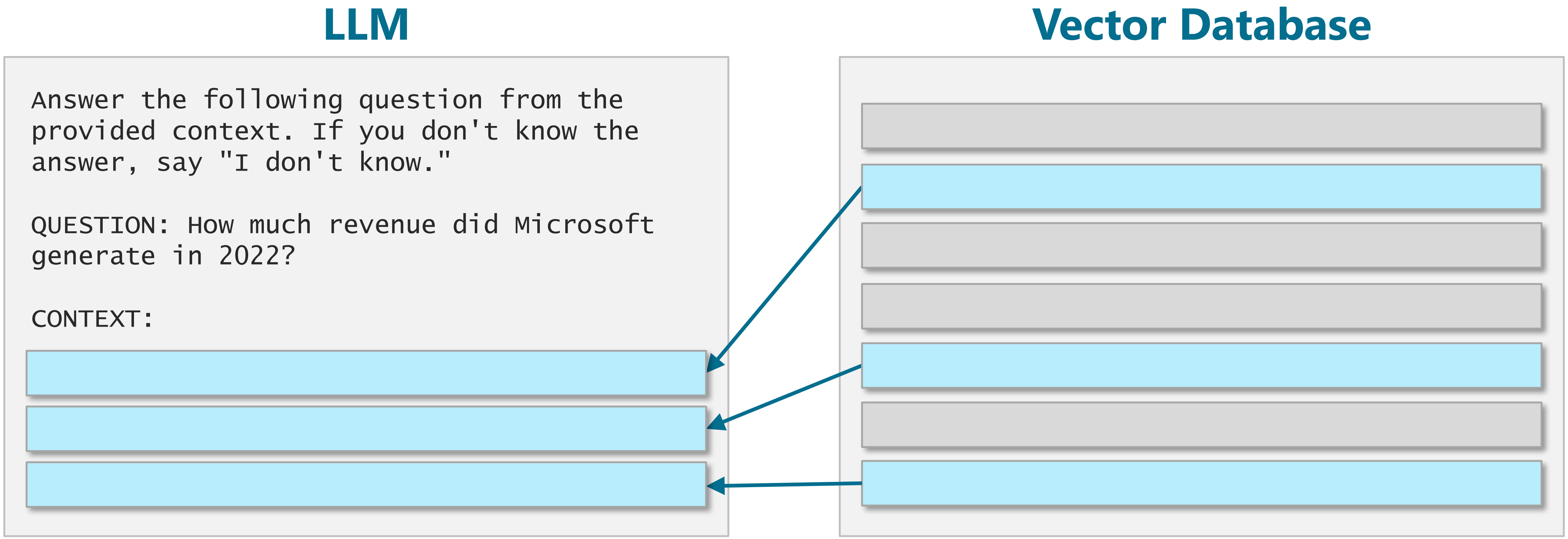
```
content = f'''
    Answer the following question, and if you don't know the answer, say "I don't know:"
    Question: Who was the first president of the United States?
    Answer:
    '''

messages = [{ 'role': 'user', 'content': content }]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages
)
```

Answering Contextual Questions

```
content = f'''
    Answer the following question using the provided context, and if the answer isn't
    contained in the context, say "I don't know:"
    Context: {context} # Insert text to be searched for an answer
    Question: Who was the first president of the United States?
    Answer:
    '''

messages = [{ 'role': 'user', 'content': content }]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages
)
```

Retrieval-Augmented Generation (RAG)

- Use LLM to answer questions using documents as context
- "Chunk" documents and use embeddings to identify relevant chunks



```
Answer the following question from the
provided context. If you don't know the
answer, say "I don't know."

QUESTION: How much revenue did Microsoft
generate in 2022?

CONTEXT:
```

OpenAI Embeddings API

- Generates high-quality embedding vectors from text
  - `text-embedding-3-small` model is best and most cost-effective
  - Generates vectors of 1,536 floating-point numbers and is applicable to long and short text samples, replacing earlier models that were sensitive to text length
- Compute similarity between two text samples by generating embeddings for each and computing the dot product
  - Useful for semantic-search systems, recommender systems, deduplication systems, and other similarity-based systems
- Combine with vector databases such as **Pinecone** or **ChromaDB** to create systems that scale to millions of embedding vectors

Generating an Embedding Vector

```
from openai import OpenAI

client = OpenAI(api_key='OPENAI_API_KEY')

response = client.embeddings.create(
    model='text-embedding-3-small',
    input='Four score and seven years ago our fathers brought forth, ' \
          'upon this continent, a new nation, conceived in liberty, and ' \
          'dedicated to the proposition that all men are created equal'
)

embedding = response.data[0].embedding

# [0.018447233363986015, 0.041993703693151474, 0.026396041736006737, ...]
```

Comparing Embedding Vectors

```
x = client.embeddings.create(
    model='text-embedding-3-small',
    input='What was the average MKT for shipments that went through LHR?'
).data[0].embedding

y = client.embeddings.create(
    model='text-embedding-3-small',
    input='Average Mean Kinetic Temperature for shipments via Heathrow'
).data[0].embedding

similarity = np.dot(np.array(x), np.array(y))
# 0.8646524079065321
```

Demo

LLMs Over Documents