TriveraTech
TECHNOLOGY TRAINING

Experience is Everything

Leveraging OpenAI for Enterprise Solutions: Business-Focused AI Boot Camp

TTAI2203

Jumping right In…

Welcome!

- **Leveraging OpenAI for Enterprise Solutions (TTAI2230)**
- Topics or activities may adjust during delivery based on your interests, roles and goals.

Hours & Breaks:

- One Hour for Lunch; A few breaks as needed

A Bit About Me:

- Dr. Ernesto Lee, Chief Innovation Officer, Trivera Technologies
- Partner Company to **Skillsoft/Global Knowledge**
- You can reach me at Dr.Lee@triveratech.com

A Bit About You:

- What's your role / day to day?
- Are you working with these skills already?
- What kinds of related things are you working on?
- What are you most excited to learn about in this class?

Agenda Review

1. OpenAI and LLMs for Business Overview
2. Enterprise Strength Speech Recognition (with OpenAI Whisper)
3. OpenAI API (Python Flavor)
4. OpenAI GPT-4 Prompt Engineering for Business Class NLP
5. OpenAI Embedding API (Searching Massive Collections of Business and Legal Documents)

Agenda Review (cont'd)

6. Build a Front End for OpenAI ChatGPT and Dall-E2 Apps
7. Voice Commands and Live Transcription
8. Build an Industry Specific Advisor / Consultant using GPT Embeddings
9. AI in Your World: Reserved for Additional Topics and Q&A
10. Voice Cloning with GPT Embeddings
11. Build a GPT model with Private Business Data
12. End to End GPT App with Private Business Data

There will be ample Hands-on Labs & Activities

Teaming for Success

Sharing Feedback – We're Here to Provide Value!

- Feedback is welcome & always encouraged
- Real time is best
- Other ways to connect
- Course Check In & End of Course feedback

TriveraTech
TECHNOLOGY TRAINING

Any Questions?

Let's Dive In!

Experience is Everything

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

DNA

CAVEMAN

What you know

Temperature (78 degrees)

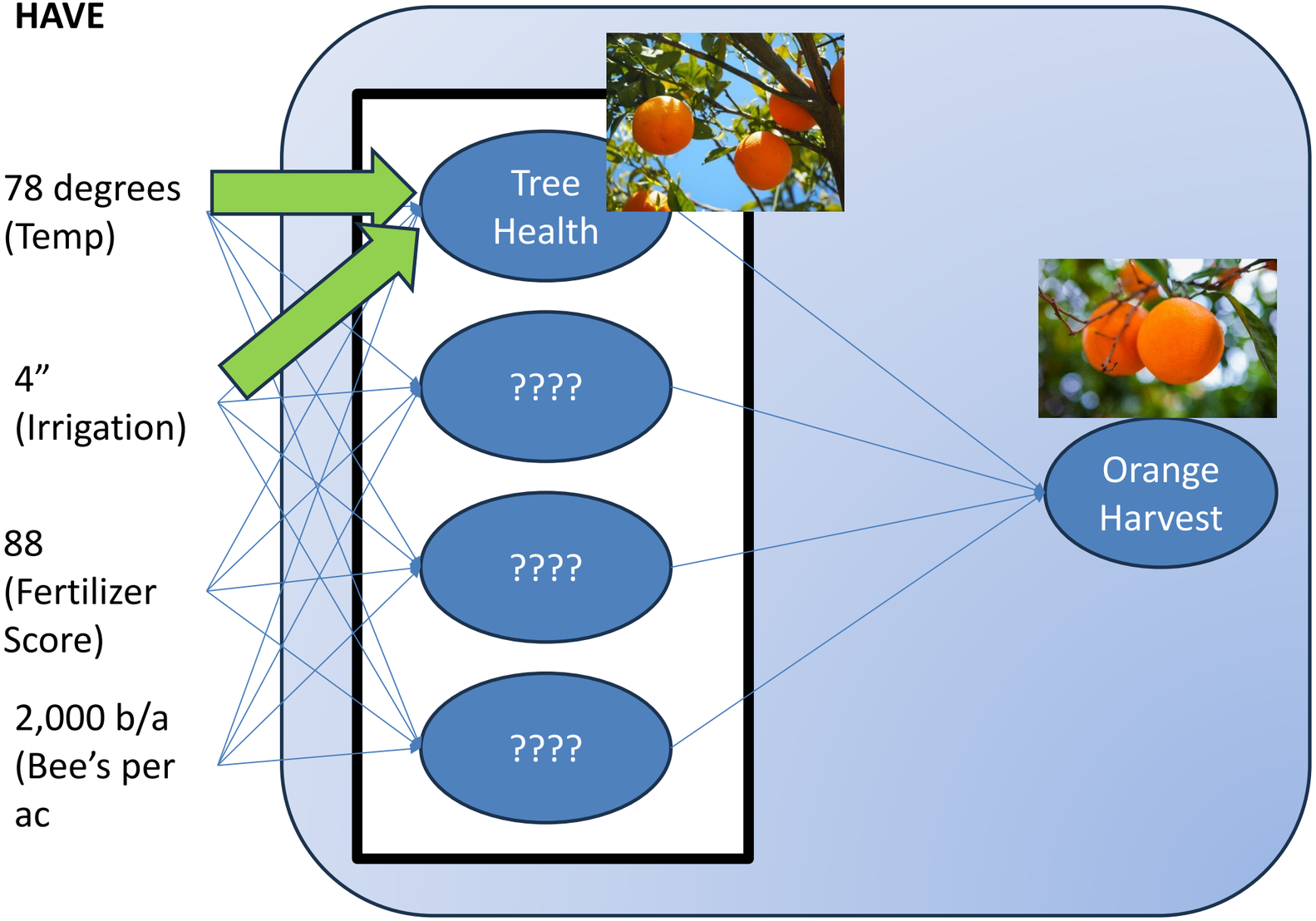
Irrigation Levels (4” per mo)

Fertilizer (88 tons/acre)

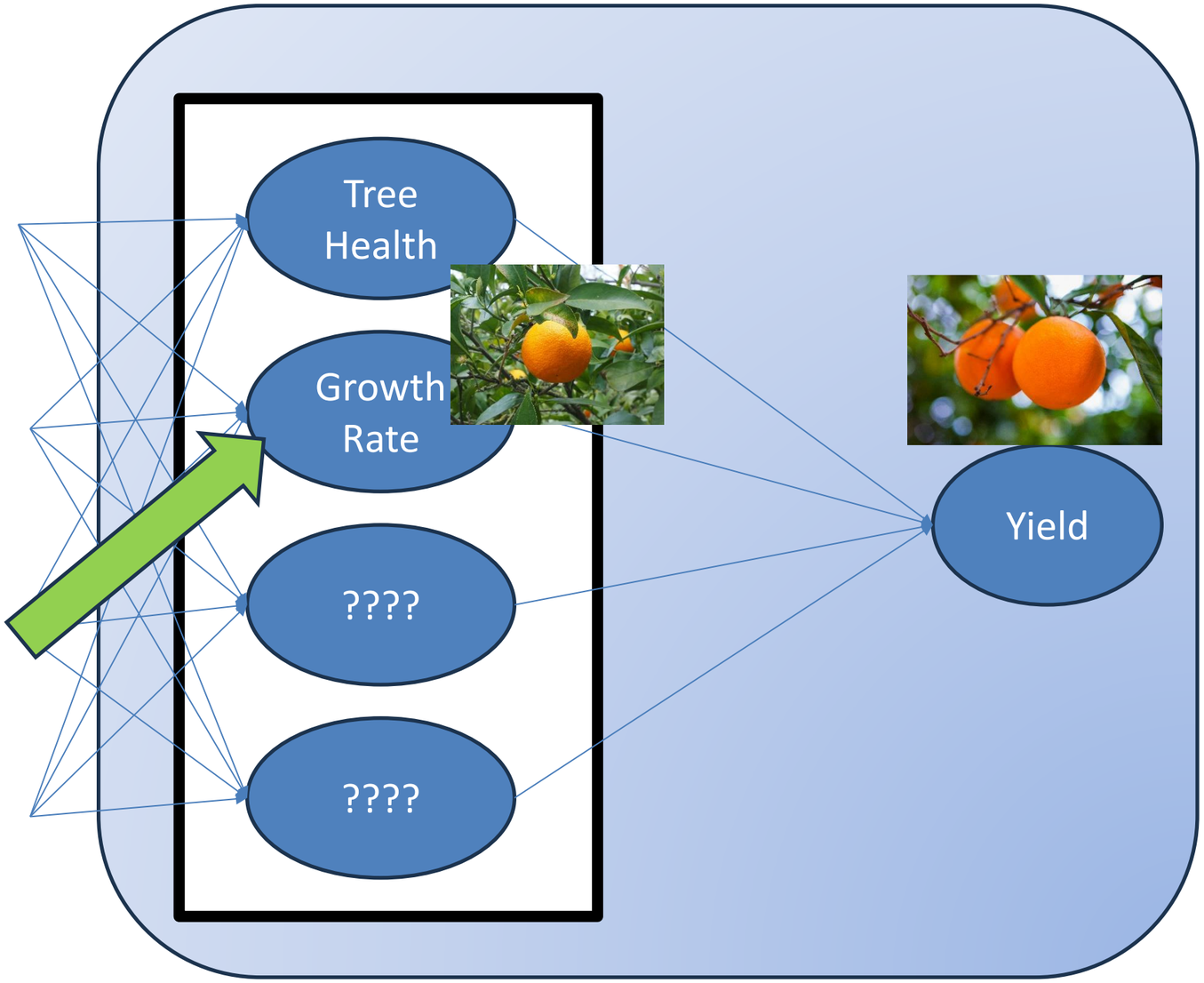
Bees (2000 bees per acre)

What you WANT to know

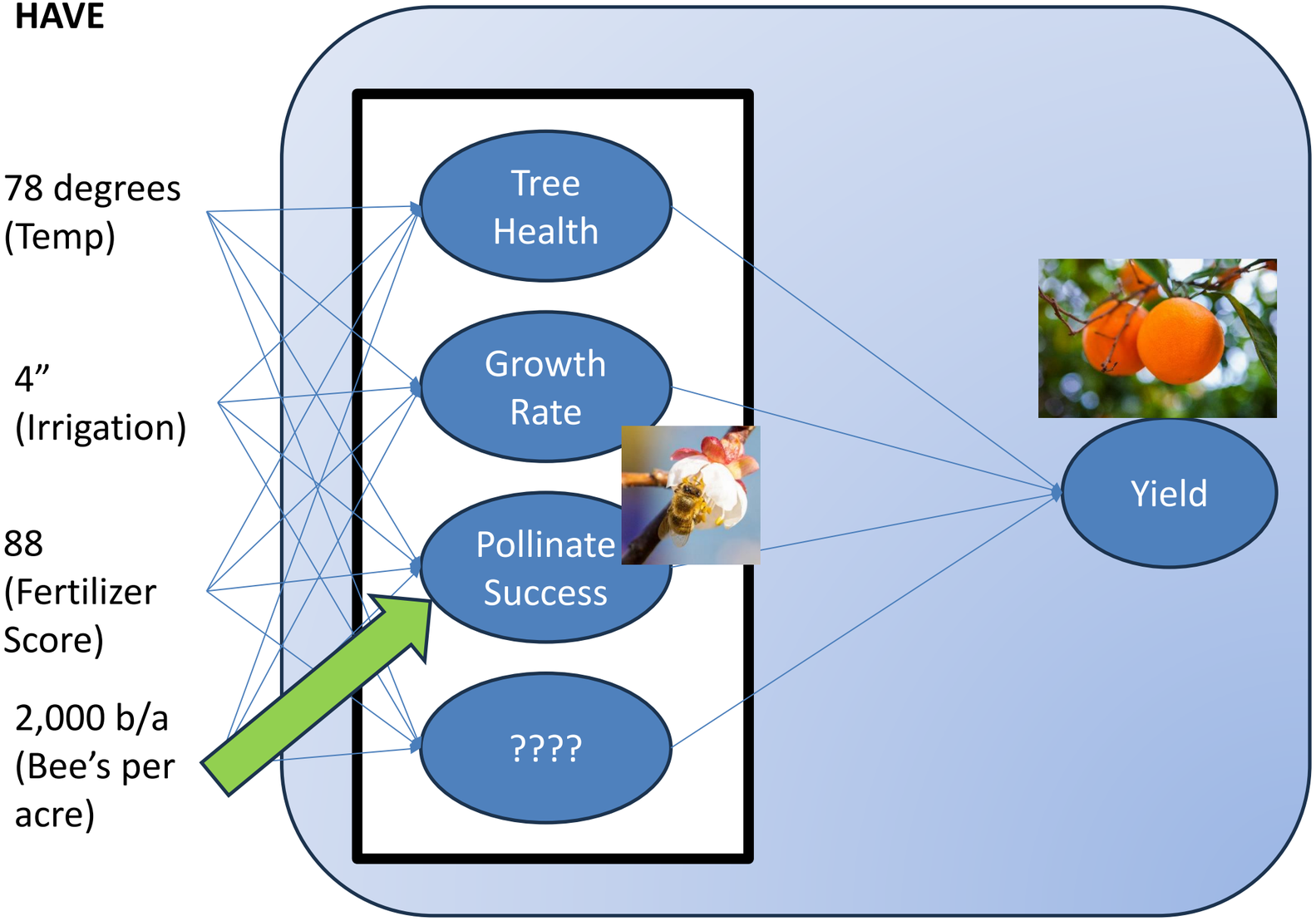
INPUTS / DATA YOU HAVE



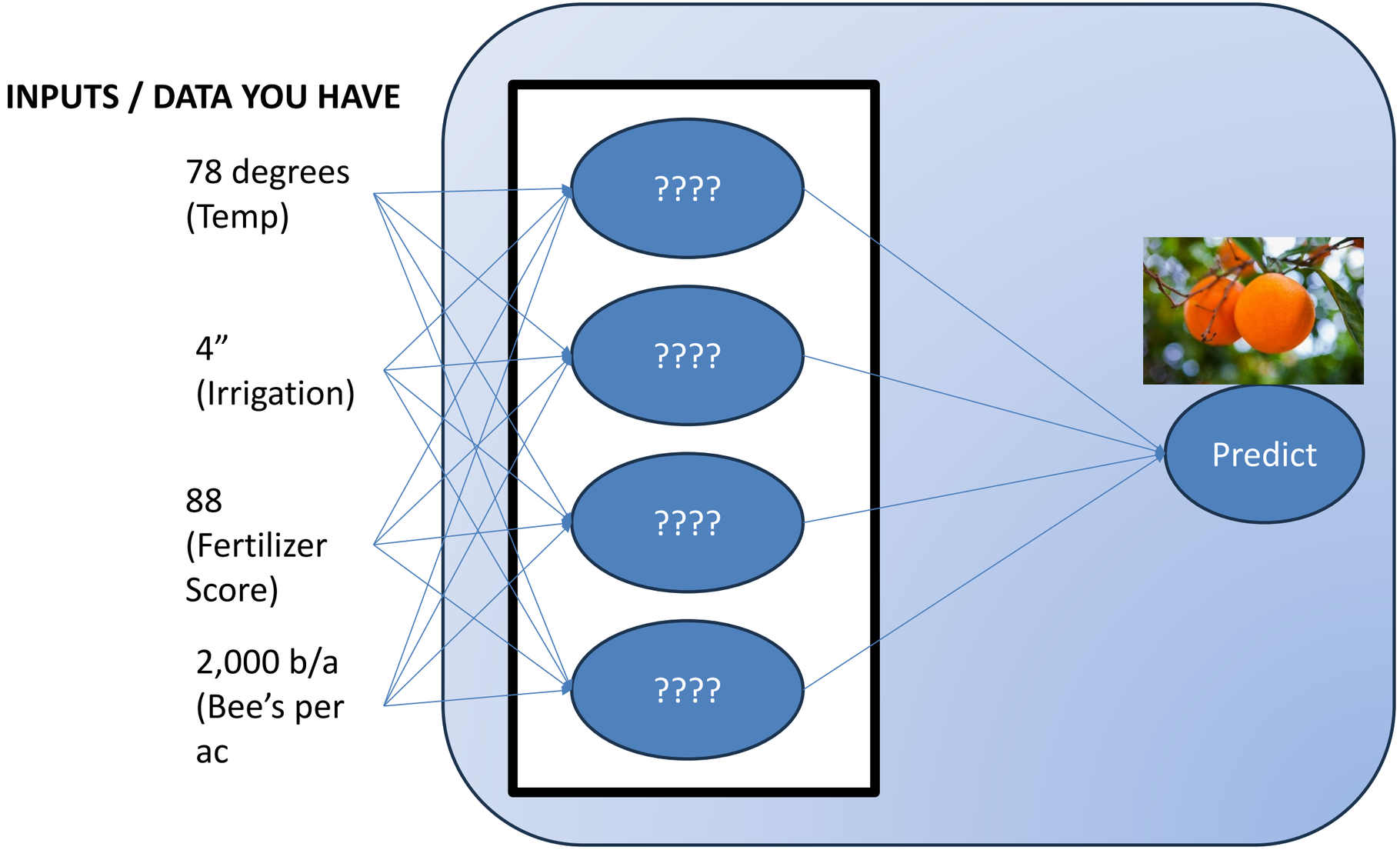
INPUTS / DATA YOU HAVE



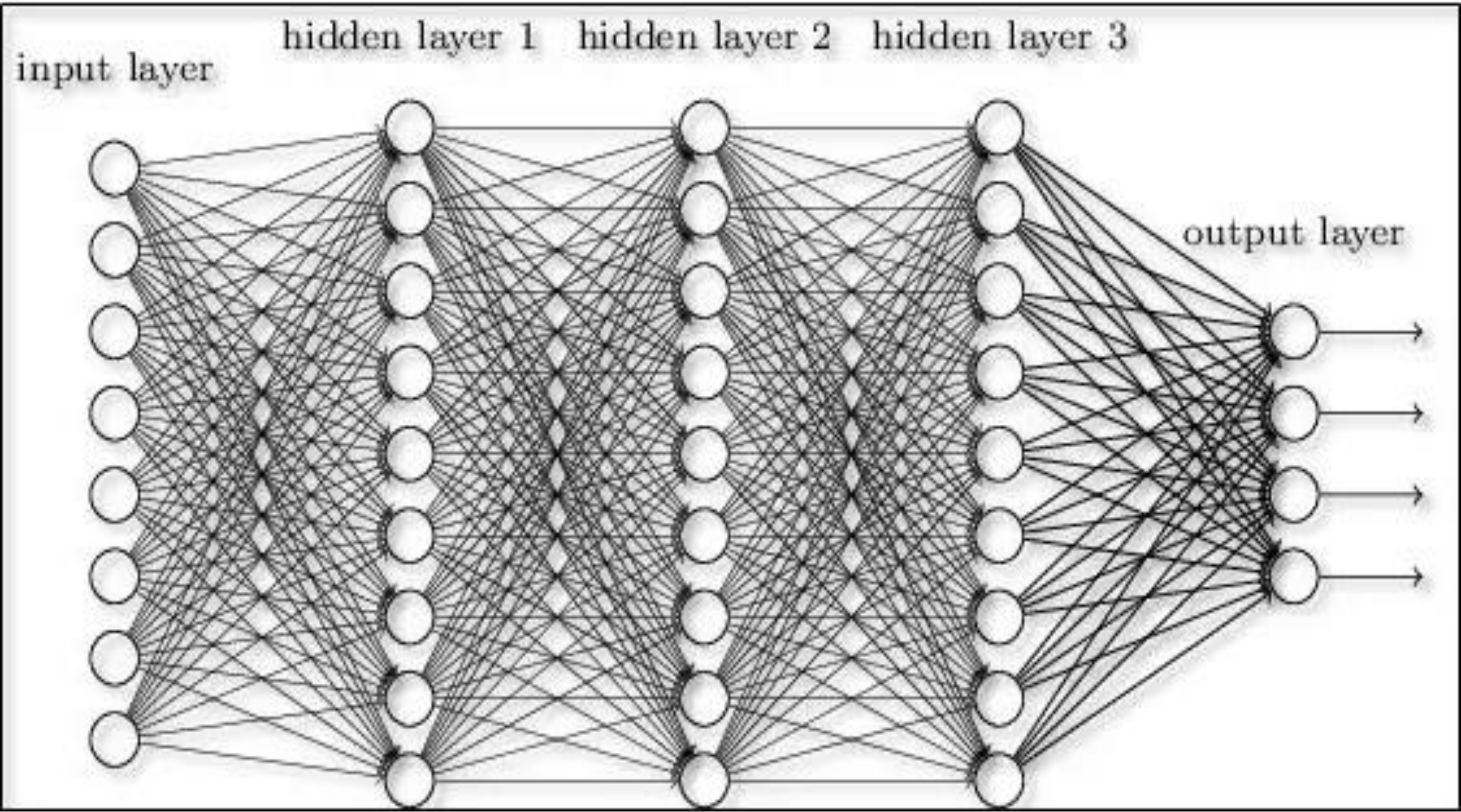
INPUTS / DATA YOU HAVE



Here is the crazy part…



This Photo by Unknown Author is licensed under CC BY-SA


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

https://arxiv.org/abs/1706.03762

Cornell University

the Simons Foundation, Stockt

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

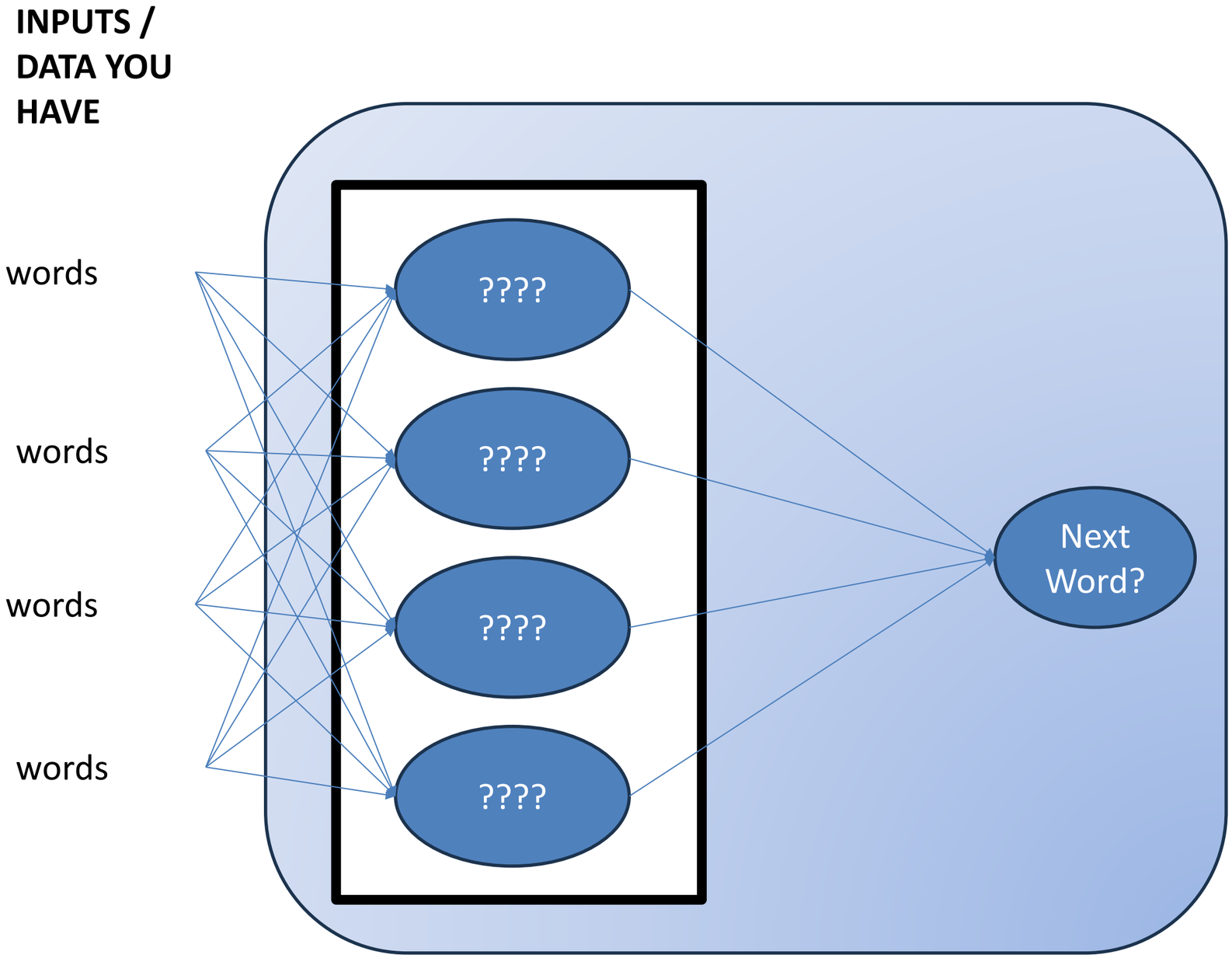
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

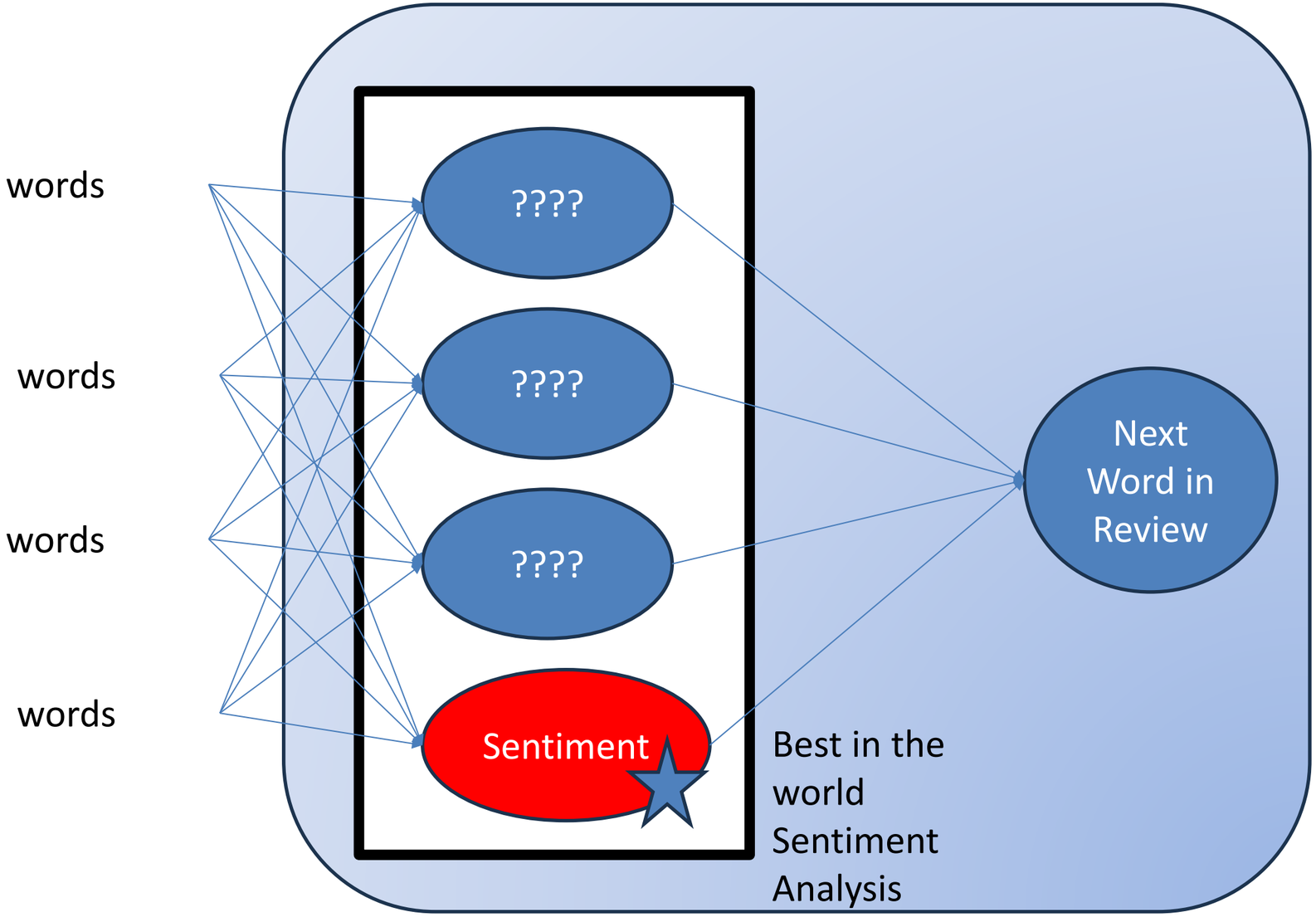
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, 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.

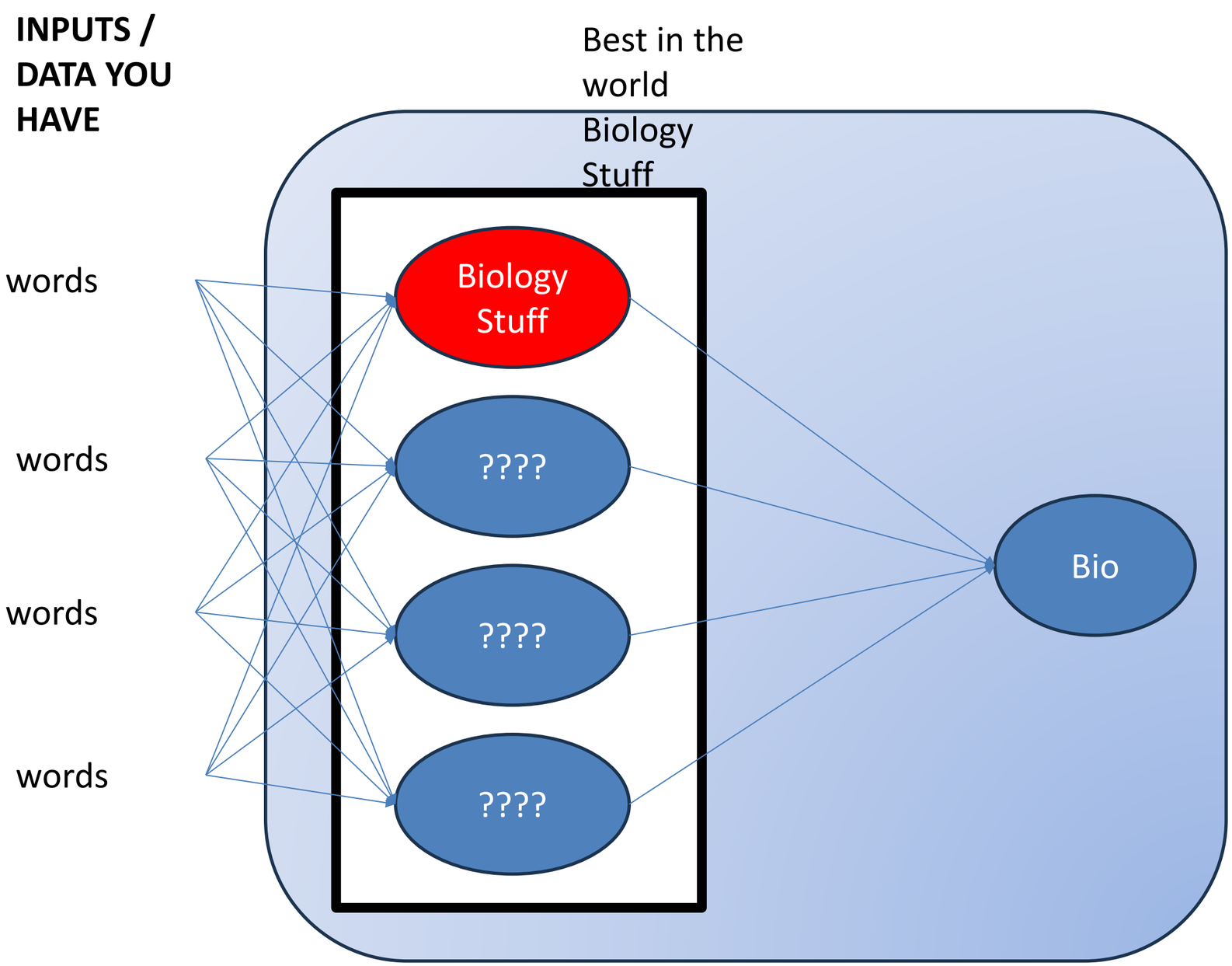
Comments: 15 pages, 5 figures
Subjects: **Computation and Language (cs.CL)**; Machine Learning (cs.LG)
Cite as: arXiv:1706.03762 **[cs.CL]**
(or arXiv:1706.03762v7 **[cs.CL]** for this version)
https://doi.org/10.48550/arXiv.1706.03762

Submission history

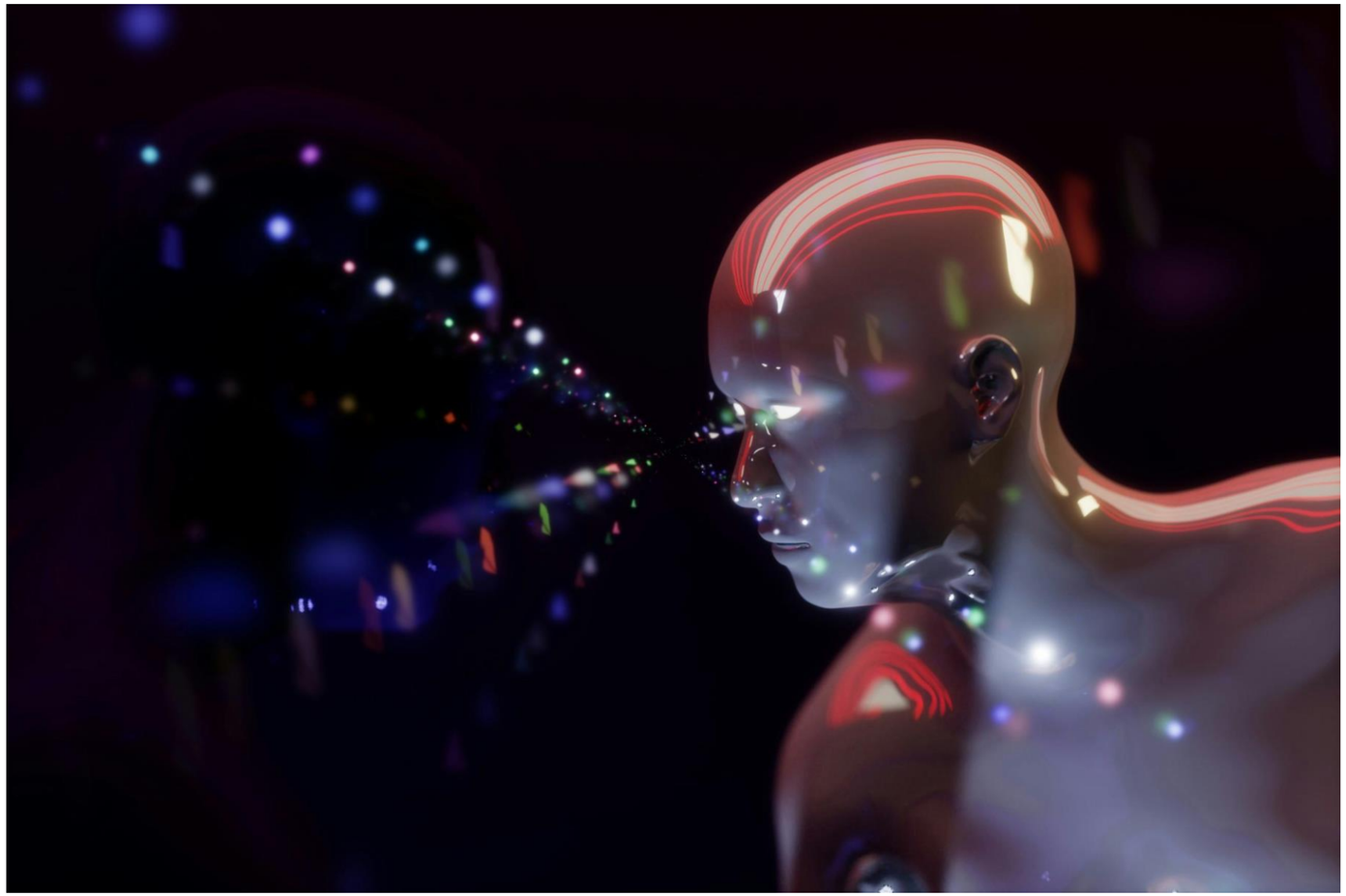
From: Llion Jones [view email]
[v1] Mon, 12 Jun 2017 17:57:34 UTC (1,102 KB)

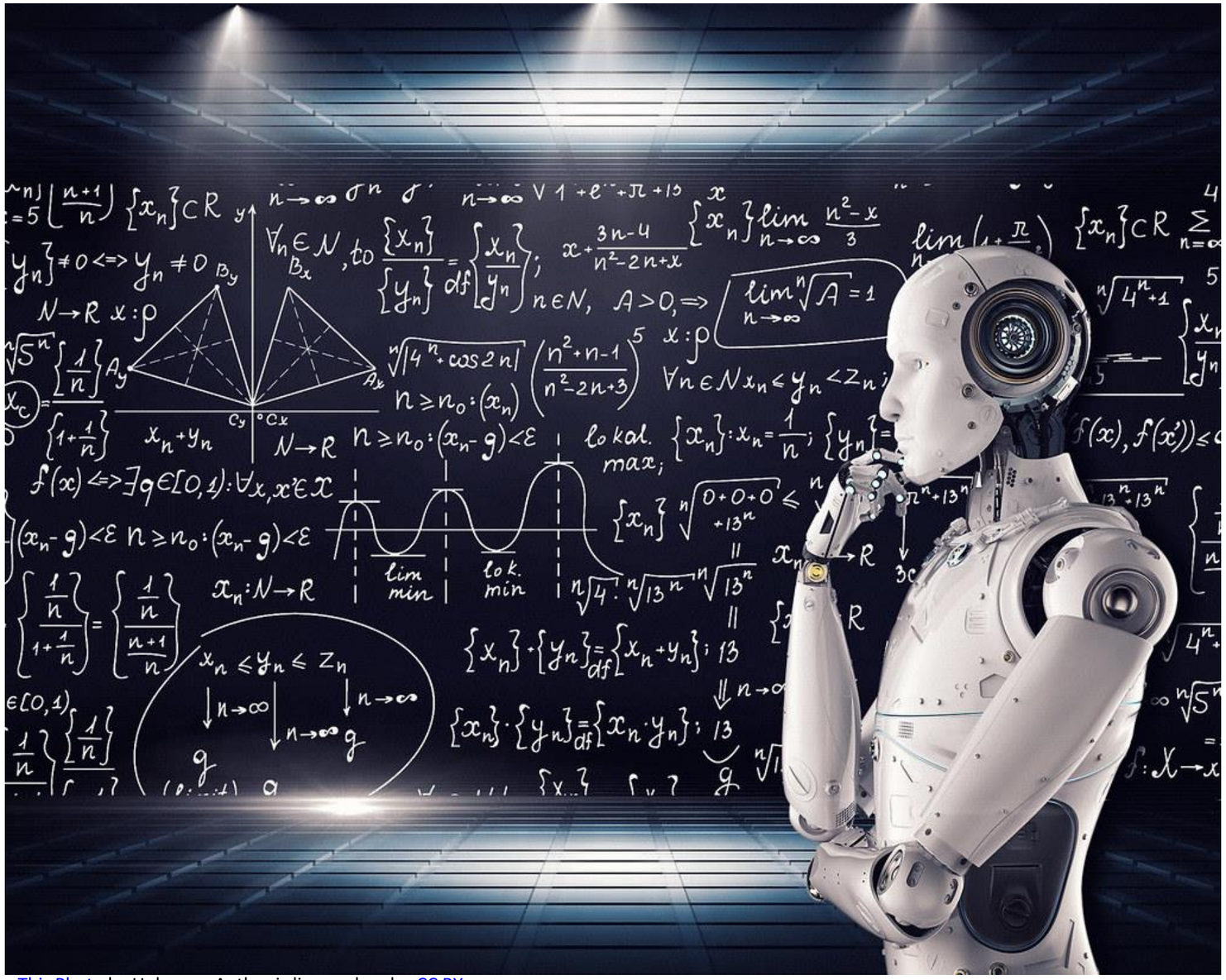

INPUTS / DATA YOU HAVE
words
words
words
words
????
????
????
????
Next Word?


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
dreamstime





This Photo by Unknown Author is licensed under CC BY

You can have confidence when using Paid OpenAI Service

When you use paid OpenAI Service, your prompts (inputs) and completions (outputs), your embeddings, and your training data

Are **NOT** available to other customers.

ARE **NOT** available to OpenAI.

Are **NOT** used to improve OpenAI models.

Are **NOT** used to improve any Microsoft or 3rd party products or services.

Are **NOT** used for automatically improving Azure OpenAI models for your use in your resource (The models are stateless, unless you explicitly fine-tune models with your training data).

Your fine-tuned Azure OpenAI models are available **exclusively** for your use.

The Azure OpenAI Service is fully controlled by Microsoft.

Microsoft hosts the OpenAI models in Microsoft's Azure environment and the Service does **NOT** interact with any services operated by OpenAI (e.g., ChatGPT, or the OpenAI API).

Tokens are the fundamental units of text that generative AI models, particularly transformers, analyze and utilize to comprehend and produce language.

Tokens help the model break down complex text into simpler, manageable pieces, making it easier to analyze and understand the underlying meaning.

By processing these individual tokens, AI models learn how words and phrases relate to each other in different contexts, which is crucial for generating coherent and contextually appropriate responses.

The use of tokens allows transformer models to focus on and predict the most relevant next pieces of text, enhancing the model's ability to generate accurate and contextually appropriate responses.

Tokens are the fundamental units of text that generative AI models, particularly transformers, analyze and utilize to comprehend and produce language.

Characters	Tokens
150	25

Use Case	Description
Document Generation	Automate the creation and updating of documents.
Transform Hardware Specifications	Convert document-based hardware specs into JSON format.
Automated Functional Testing	Streamline testing processes for both hardware and software.
Code Porting	Facilitate the migration of code across different platforms.
Code Modernization and Migration	Simplify the updating and transitioning of legacy code.
Democratize Expertise	Make Subject Matter Expert (SME) knowledge accessible to a broader audience.
Hardware Logging and Error Analysis	Enhance the analysis of hardware logs and error reports.
Predictive Maintenance	Implement AI to predict and prevent equipment failures by analyzing usage patterns and historical data.
Scientific Data Analysis	Accelerate interpretation of complex scientific data collected from space missions and experiments.

TriveraTech
TECHNOLOGY TRAINING

Demo

GPT 4.0 Documentation Generation

Experience is Everything

To all projects

Current project
retroburn-project-east

Project overview
Settings

Get started
Model catalog
Model benchmarks
Prompt catalog

Project playground
Chat
Assistants PREVIEW
Images
Completions

Tools
Code PREVIEW
Prompt flow
Tracing PREVIEW
Evaluation PREVIEW
Fine-tuning PREVIEW

Components
Data
Indexes
Deployments
Content filters PREVIEW

Assistants playground

Connected resource: Default_AzureOpenAI

New assistant | Load assistant | Delete | View Code

Deployment * Create new deployment

gpt-4 (version:1106-Preview)

Assistant name

SOW Generator

Assistant id: asst_Z9DR4DeHoYJQnDWzKAJoAVF7

Prompt | **Tools** | Parameters

File search — Add files

Code interpreter — Add files

Functions — Add function

Last saved: today 8:33 PM

Clear chat | Logs | Thread files 1 | JSON response | 0 tokens

New thread started

thread_A8eoNCH85ZpTPc1mrlpebi93

Type user query here. (Shift + Enter for new line)

Messages in the Assistants playground are visible to anyone with access to this resource and using the API.

TriveraTech
TECHNOLOGY TRAINING

Demo

GPT-4o Transform Hardware Specs

Experience is Everything

Draw Ask Copilot 1 of 52

NanoPZ™

Ultra-High Resolution Motion System

Newport®
Experience | Solutions

USER'S MANUAL

TriveraTech
TECHNOLOGY TRAINING

Demo

GPT-4o Code Porting

Experience is Everything

Contoso

Show chat history

Share

Start chatting

This chatbot is configured to answer your questions

I have a simple todo app written in python 2.x, please help me convert it into python 3, at the explain the differences

def display_tasks(tasks):

TriveraTech
TECHNOLOGY TRAINING

Demo

GPT-4o Code Monetization

Experience is Everything

Contoso

Show chat history

Share

Start chatting

This chatbot is configured to answer your questions

I have a todo app written in FORTRAN, can you help me update the application to C# with .net?

PROGRAM TODO_APP

TriveraTech
TECHNOLOGY TRAINING

Demo

GPT-4o Hardware Logging and Error Analysis

Experience is Everything

To all projects

Current project
retroburn-project-east

Project overview
Settings

Get started
Model catalog
Model benchmarks
Prompt catalog

Project playground
Chat
Assistants PREVIEW
Images
Completions

Tools
Code PREVIEW
Prompt flow
Tracing PREVIEW
Evaluation PREVIEW
Fine-tuning PREVIEW

Components
Data
Indexes
Deployments
Content filters PREVIEW

Assistants playground

Connected resource: Default_AzureOpenAI

New assistant | Load assistant | Delete | View Code

Deployment * Create new deployment

gpt-4 (version:1106-Preview)

Assistant name

Hardware Logging and Error Analysis

Assistant id: asst_w1WAXkfQ9TWofbi1XxgFjW7P

Prompt Tools Parameters

Instructions

You are an automated system monitoring logs and providing error analysis and troubleshooting guidance to the users. When you see logs, process them and provide notification of errors and guidance. If there are no issues, just confirm receipt of the logs.

Last saved: today 9:04 PM

Clear chat | Logs | Thread files 1 | JSON response | 0 tokens

New thread started

thread_wT7skHNIWZtUQDamrdvH6bfr

[2024-07-25 00:59:00] [Celestial Tracker] Data transmission initiated
[2024-07-25 01:00:00] [Celestial Tracker] Data transmission in progress
[2024-07-25 01:01:00] [Celestial Tracker] Data transmission successful
[2024-07-25 01:02:00] [Celestial Tracker] Data received by server

Messages in the Assistants playground are visible to anyone with access to this resource and using the API.

TriveraTech
TECHNOLOGY TRAINING

Demo

GPT-4o Predictive Maintenance

Experience is Everything

← To all projects

Current project
retroburn-project-east

Project overview
Settings

Get started
Model catalog
Model benchmarks
Prompt catalog

Project playground
Chat
Assistants PREVIEW
Images
Completions

Tools
Code PREVIEW
Prompt flow
Tracing PREVIEW
Evaluation PREVIEW
Fine-tuning PREVIEW

Components
Data
Indexes
Deployments
Content filters PREVIEW

Assistants playground

Success: Successfully updated thread

Connected resource: Default_AzureOpenAI | New assistant | Load assistant | Delete | View Code

Deployment * Create new deployment

gpt-4o (version:2024-05-13)

Assistant name

Preventative Maintenance

Assistant id: asst_tX2SesVqrIEqgBU9WkhfKJyc

Prompt | Tools | Parameters

File search — Add files

Code interpreter — Add files

Functions — Add function

Last saved: today 10:01 PM

Clear chat | Logs | Thread files 1 | JSON response | 0 tokens

New thread started
thread_HAP0nUy0zniEJHcf0dqne1yn

System Pressure: 3.5 bar
Voltage: 230V
Current (Amp Usage): 13A
Notes: Internal temperature higher, pressure significantly increased. Voltage stable but current usage is elevated. Check for compressor issues or blocked vents.

Messages in the Assistants playground are visible to anyone with access to this resource and using the API.

TriveraTech
TECHNOLOGY TRAINING

Demo

Using OpenAI with Jupyter Notebook (Colab)

Experience is Everything

Azure AI | Machine Learning Studio

TeamOne-AppAlpha-Dev
retroburn-ml

All workspaces

Home

Model catalog

Authoring

Notebooks

Automated ML

Designer

Prompt flow

Tracing PREVIEW

Assets

Data

Jobs

Components

Pipelines

Environments

Models

Endpoints

Manage

Compute

Monitoring

Retroburn > retroburn-ml > Notebooks

*example.ipynb

Edit in VS Code | Compute: ml-studio - Running | Python 3.10 - SDK v2

ml-studio · Kernel idle CPU100% RAM 2% | Last saved a few seconds ago | Python 3.10 - SDK V2

Files

```
pip install openai
```

```
import os
from openai import AzureOpenAI

client = AzureOpenAI(
  azure_endpoint = "https://retroburn-oai-west.openai.azure.com/",
  api_key="2686407d21864d0a99aac127e8b3f3d4",
  api_version="2024-02-01"
)

response = client.chat.completions.create(
    model="gpt-4o", # model = "deployment_name"
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "What are the differences between Azure Machine Learning and Azure AI services?"}
    ]
)

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

TriveraTech
TECHNOLOGY TRAINING

Demo

Using Azure OpenAI with Postman

Experience is Everything

Home Workspaces Explore Search Postman Sign In Create Account

POST https://aoai-virginia-priv | GET https://govsolution-ai-se | POST https://aoai-virginia-priv | POST https://docintel-virginia | GET https://docintel-virginia- | GET https://docintel-virginia-

https://aoai-virginia-private.openai.azure.us/openai/deployments/gpt-4/chat/completions?api-version=2024-02-15-preview

Save

POST https://aoai-virginia-private.openai.azure.us/openai/deployments/gpt-4/chat/completions?api-version=2024-02-15-preview Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary JSON Beautify

```
{
    "messages": [
        {
            "role": "system",
            "content": "You are an AI assistant that helps people find information."
        },
        {
            "role": "user",
            "content": "What is NIST 800-53?"
        }
    ],
    "data_sources": [
    {
      "type": "azure_search",
      "parameters": {
        "endpoint": "https://govsolution-ai-search.search.azure.us",
        "index_name": "vector-1718718349325",
        "semantic_configuration": "vector-1718718349325-semantic-configuration",
        "query_type": "semantic",
        "fields_mapping": {
          "content_fields_separator": "\n",
          "content_fields": [
            "chunk"
          ],
```

Body Cookies Headers (11) Test Results Status: 200 OK Time: 8.42 s Size: 11.87 KB Save Response

Pretty Raw Preview Visualize JSON

Console Not connected to a Postman account

Type here to search 12:23 AM 7/26/2024

Grounding

Capture the semantics of the input data such that similar data points are close to each other in this space, and dissimilar data points are far apart.

Embeddings are a specific type of vector representation of data (like words, sentences, or even images)

Each dimension of the vector captures some aspect of the data's content, making embeddings rich and informative representations.

Embedding models are trained on large datasets to learn how best to represent data as vectors, capturing complex patterns and relationships.

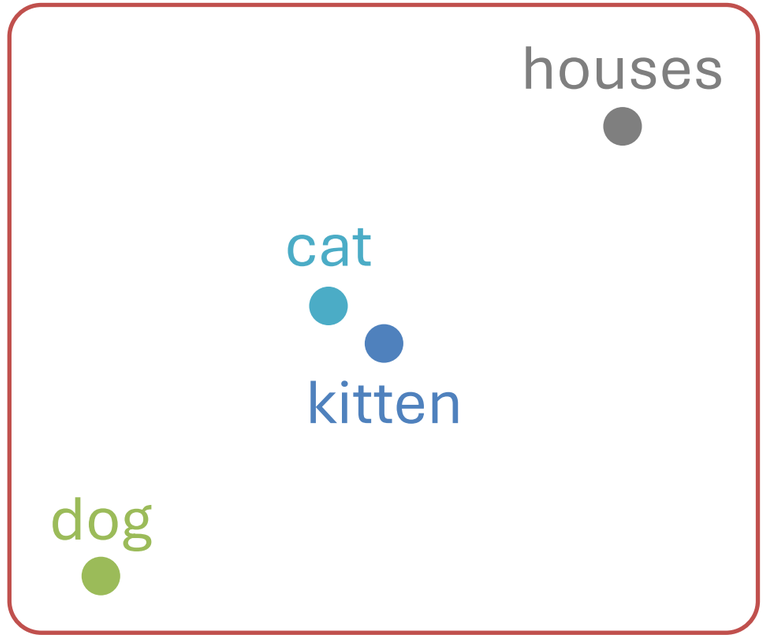
Embedding Single Example

"cat" ➡ Embedding model ➡ { 0.8 , 0.9 , -0.3 , -0.2 }

Embedding Example Set

	mammal	walks	verb	plural
cat	0.8	0.9	-0.3	-0.2
kitten	0.6	0.8	-0.5	-0.1
dog	0.7	0.9	-0.1	-0.3
houses	-0.8	-0.9	0.3	0.8

Embedding Semantic Relationship Example



Chunking is a data segmentation technique that enhances the embedding and semantic search process by dividing content into smaller segments.

Enables the RAG model to efficiently search and retrieve relevant chunks of information from large databases, focusing the retrieval process on more specific text segments.

RAG models can generate responses that are contextually richer and more accurate, as each chunk provides a focused context for generation.

Chunking scales the integration of external knowledge into the generation process, allowing RAG models to handle expansive knowledge bases more effectively.

Document to Chunk

Microsoft Azure OpenAI Service offers a suite of powerful tools that significantly benefit enterprise customers by integrating advanced AI capabilities into their operations. Firstly, the service allows businesses to leverage state-of-the-art AI models like GPT (Generative Pre-trained Transformer) and DALL-E, enhancing their applications with features such as natural language understanding, text generation, and image creation. This integration enables businesses to automate complex tasks, improve customer interactions through sophisticated chatbots, and generate dynamic content, leading to increased efficiency and innovation.

Another significant advantage of Azure OpenAI for enterprises is its seamless integration with the broader Azure ecosystem. This compatibility ensures that businesses can easily deploy AI solutions within their existing infrastructure, benefiting from Azure’s robust security features, compliance coverage, and global scalability. The ability to maintain data within Azure's secure environment addresses critical concerns around data privacy and regulatory compliance, crucial for sectors like finance and healthcare.

Furthermore, Azure OpenAI Service provides enterprises with extensive support and scalability options, adapting to varying business needs and demands. Companies can start with smaller deployments to test and iterate on their AI implementations and then scale effortlessly as their requirements grow. The service's built-in monitoring and management tools also allow enterprises to maintain control over their AI models, ensuring they deliver consistent performance and align with business goals. This flexibility and control empower enterprises to stay ahead in a rapidly evolving technological landscape, maximizing their return on investment in AI technologies.

Azure OpenAI Service.pdf

Chunking Output

Microsoft Azure OpenAI Service offers a suite of powerful tools that significantly benefit enterprise customers by integrating advanced AI capabilities into their operations. Firstly, the service allows businesses to leverage state-of-the-art AI models like GPT (Generative Pre-trained Transformer) and DALL-E, enhancing their applications with features such as natural language understanding, text generation, and image creation. This integration enables businesses to automate complex tasks, improve customer interactions through sophisticated chatbots, and generate dynamic content, leading to increased efficiency and innovation.

Another significant advantage of Azure OpenAI for enterprises is its seamless integration with the broader Azure ecosystem. This compatibility ensures that businesses can easily deploy AI solutions within their existing infrastructure, benefiting from Azure's robust security features, compliance coverage, and global scalability. The ability to maintain data within Azure's secure environment addresses critical concerns around data privacy and regulatory compliance, crucial for sectors like finance and healthcare.

Furthermore, Azure OpenAI Service provides enterprises with extensive support and scalability options, adapting to varying business needs and demands. Companies can start with smaller deployments to test and iterate on their AI implementations and then scale effortlessly as their requirements grow. The service's built-in monitoring and management tools also allow enterprises to maintain control over their AI models, ensuring they deliver consistent performance and align with business goals. This flexibility and control empower enterprises to stay ahead in a rapidly evolving technological landscape, maximizing their return on investment in AI technologies.

X


```
{
  “chunk": {
    "origin": "Azure OpenAI Service.pdf",
    "chunkNumber": 1,
    "chunkTotal": 3,
    "content": "Microsoft Azure OpenAI Service offers a
          suite of powerful tools that significantly
          benefit enterprise customers by integrating
          advanced AI capabilities into their operations.
          Firstly, the service allows businesses to
          leverage state-of-the-art AI models like GPT
          (Generative Pre-trained Transformer) and DALL-E,
          enhancing their applications with features such
          as natural language understanding, text
          generation, and image creation. This integration
          enables businesses to automate complex tasks,
          improve customer interactions through
          sophisticated chatbots, and generate dynamic
          content, leading to increased efficiency and
          innovation."
  }
}
```


Embedding model

Chunk Embedding


```
{
  “chunk": {
    "origin": "Azure OpenAI
Service.pdf",
    "chunkNumber": 1,
    "chunkTotal": 3,
    "content": "Microsoft Azure OpenAI
Service offers a     suite of powerful
tools that significantly       benefit
enterprise customers by integrating
          advanced AI capabilities into
their operations.    Firstly, the
service allows businesses to   leverage
state-of-the-art AI models like GPT
          (Generative Pre-trained
Transformer) and DALL-E,       enhancing
their applications with features such
          as natural language
understanding, text  generation, and
```


Embedding model


```
{
  “embedding": [0.52955132, 0.1604042,
0.29707274,          0.84226021,
0.95790925, 0.0208008, 0.29175048,
          0.67672266, 0.77173089,
0.19946357, 0.13638158,
          0.96587453, 0.20663748,
0.8226592, 0.72857113,
          0.13109249, 0.72750379,
0.19441223, 0.65595485,
          0.93552191, 0.63599398,
0.39110781, 0.02016978,
          0.63665149, 0.57542323,
0.8908598, 0.25021257,
          0.52606665, 0.20760358,
```

Secure, scalable solution that revolutionizes information retrieval of user-owned content.

Comprehensive search engine that supports vector, full-text, and hybrid searches across a richly indexed database.

Works closely with other Azure services, allowing for automated data ingestion from Azure data sources and incorporation of consumable AI for advanced processing tasks

Advanced features like semantic ranking, relevance tuning, and comprehensive query syntax support.

Information Assistant Accelerator

Custom Configuration Code

App Services, Application Insights, Azure Cosmos DB, Azure Form Recognizer, Azure OpenAI Service, Azure Cognitive Search, Azure AD Application Registration, Azure Active Directory, Log Analytics workspace, Storage account

Deployment Kit

Parameterized Scripts and Installation Guide to rapidly spin up your own instance on your own tenant

Cost Calculator

Enables customers/ partners to examine cost components impacting users/capacity/ performance

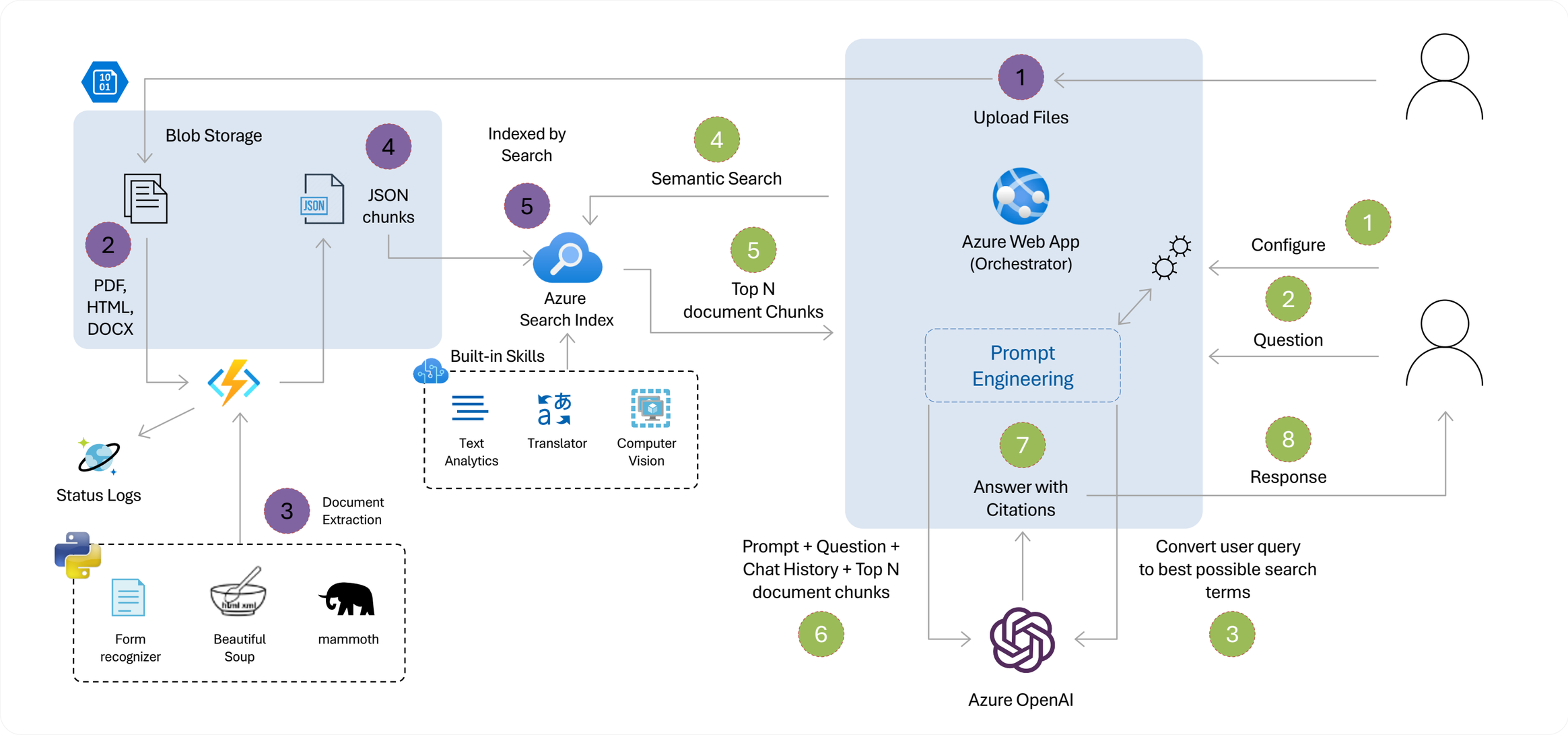
Documentation

Architectural Diagrams, Service Configurations, Documentation to assist with Accreditation on Classified Networks

Adoption Workshop Content (2 sessions, 4 hours each)

A combination of virtual assets and direct customer assistance to stand up the accelerator in a customer or Partner's Azure instance

Conversation with Private/Domain Specific Data



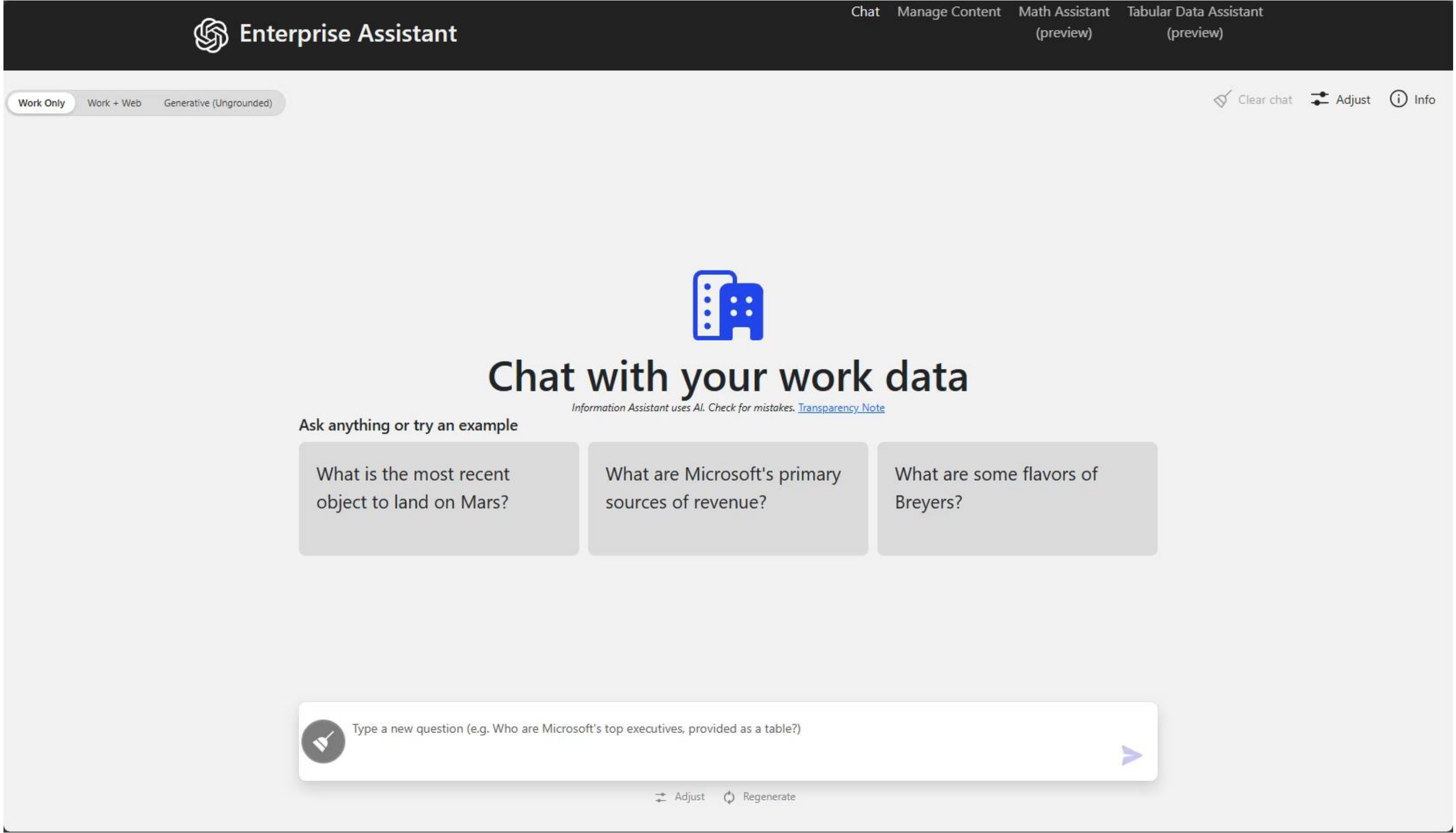
TriveraTech
TECHNOLOGY TRAINING

Demo

Information Assistant

Private/Domain Data Use-case

Experience is Everything


Enterprise Assistant
Chat
Manage Content
Math Assistant (preview)
Tabular Data Assistant (preview)
Work Only
Work + Web
Generative (Ungrounded)
Clear chat
Adjust
Info
Chat with your work data
Information Assistant uses AI. Check for mistakes. Transparency Note
Ask anything or try an example
What is the most recent object to land on Mars?
What are Microsoft's primary sources of revenue?
What are some flavors of Breyers?
Type a new question (e.g. Who are Microsoft's top executives, provided as a table?)
Adjust
Regenerate

OpenAI

What is the OpenAI?

Experience is Everything

Coming up...

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

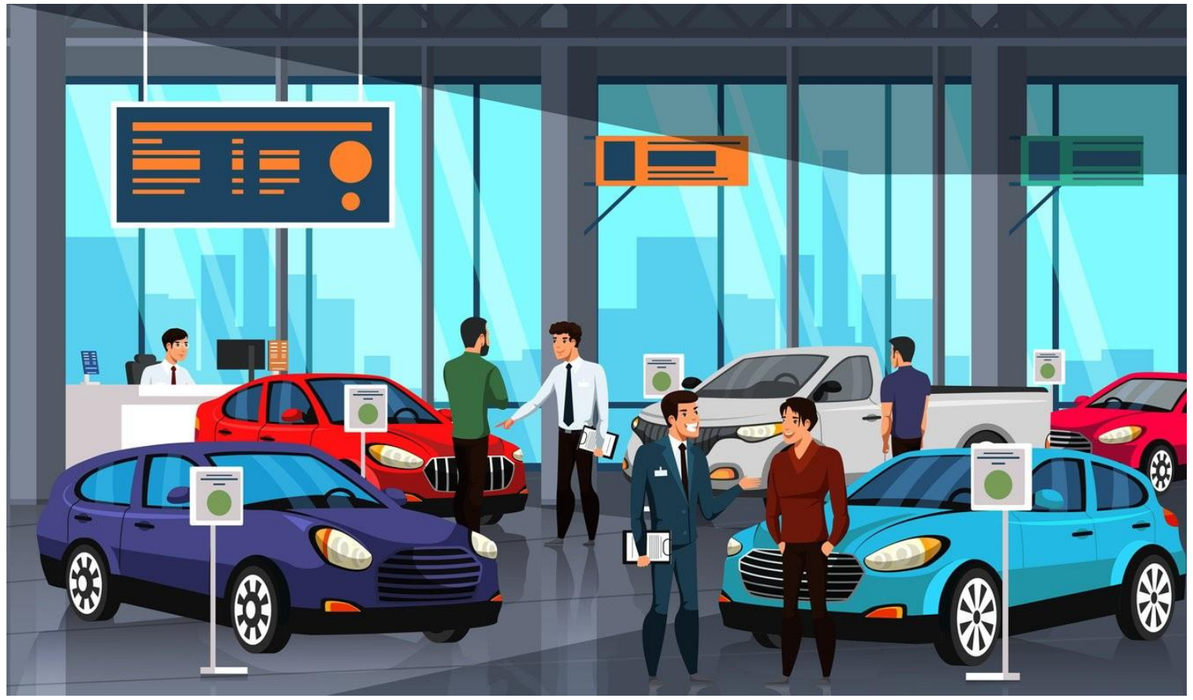
OpenAI, ChatGPT, and the OpenAI API

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

OpenAI

OpenAI, ChatGPT, and the OpenAI API

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

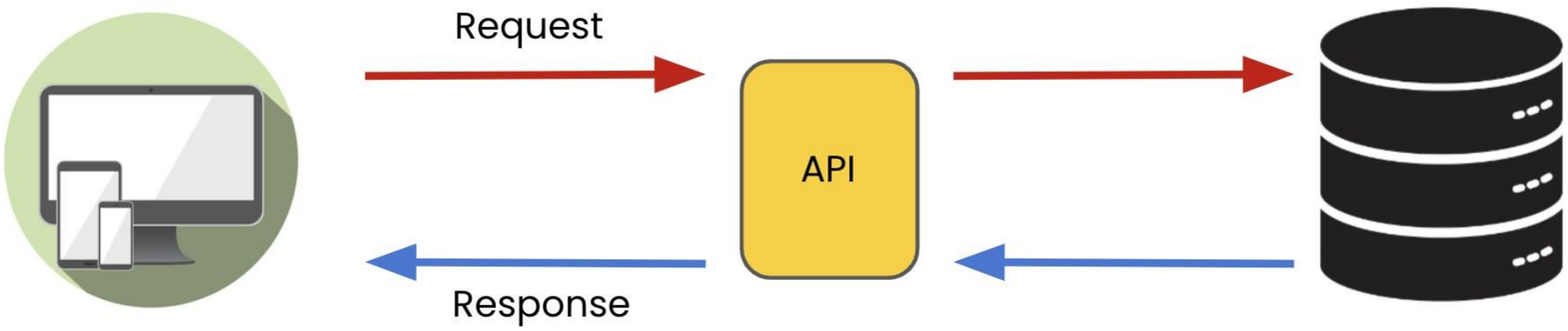


OpenAI, ChatGPT, and the OpenAI API

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

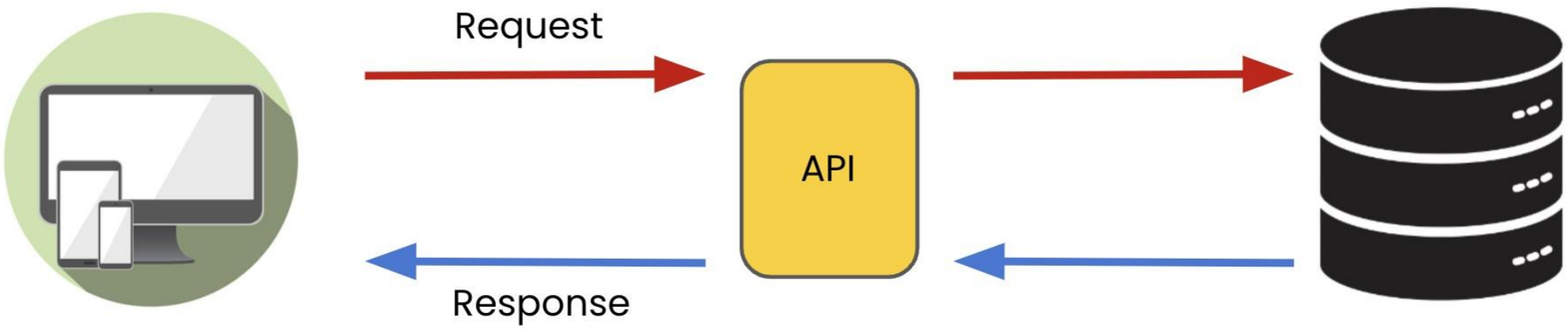
What is an API?

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

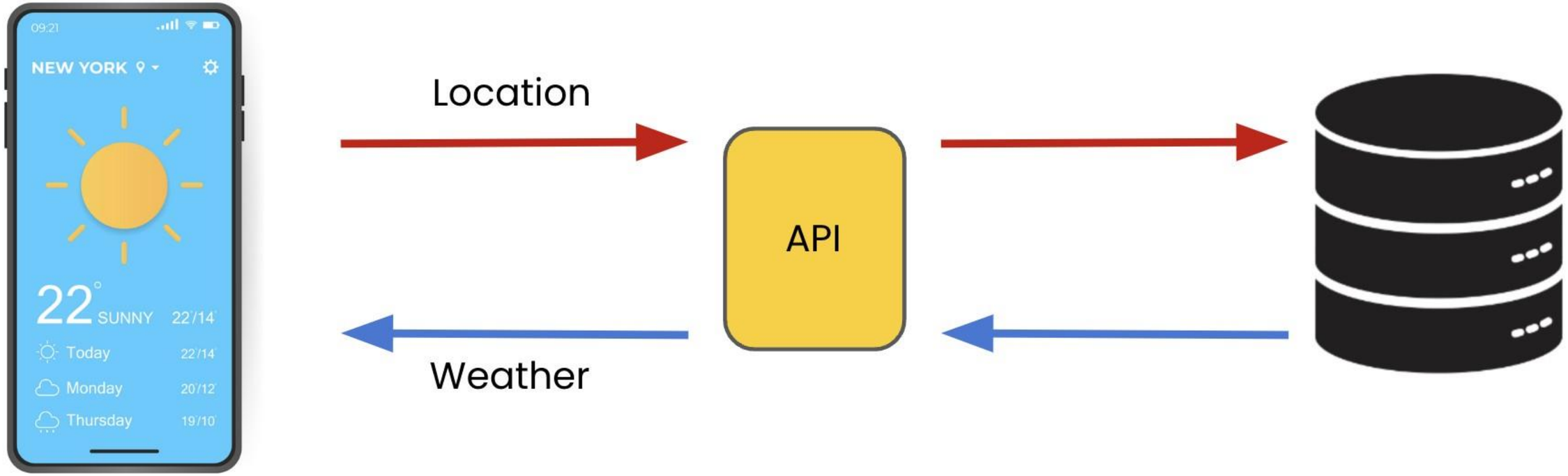


What is an API?

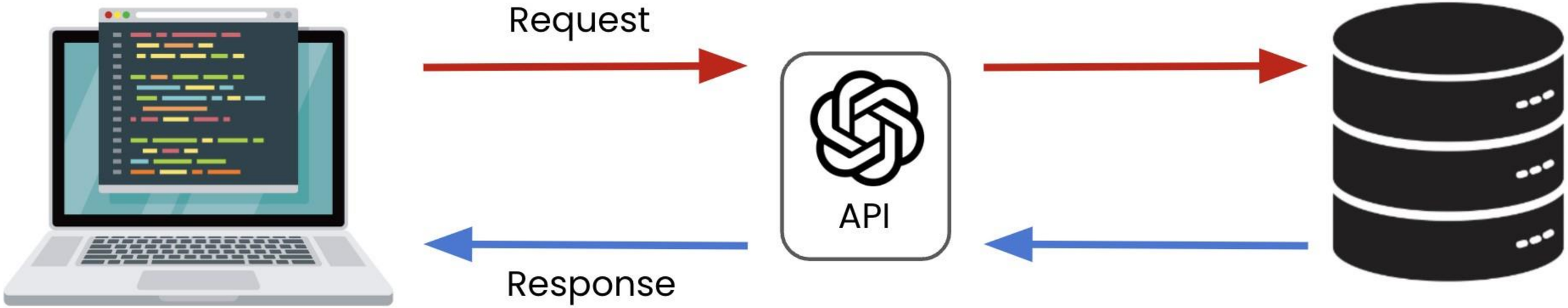
- **A**pplication **P**rogramming **I**nterface



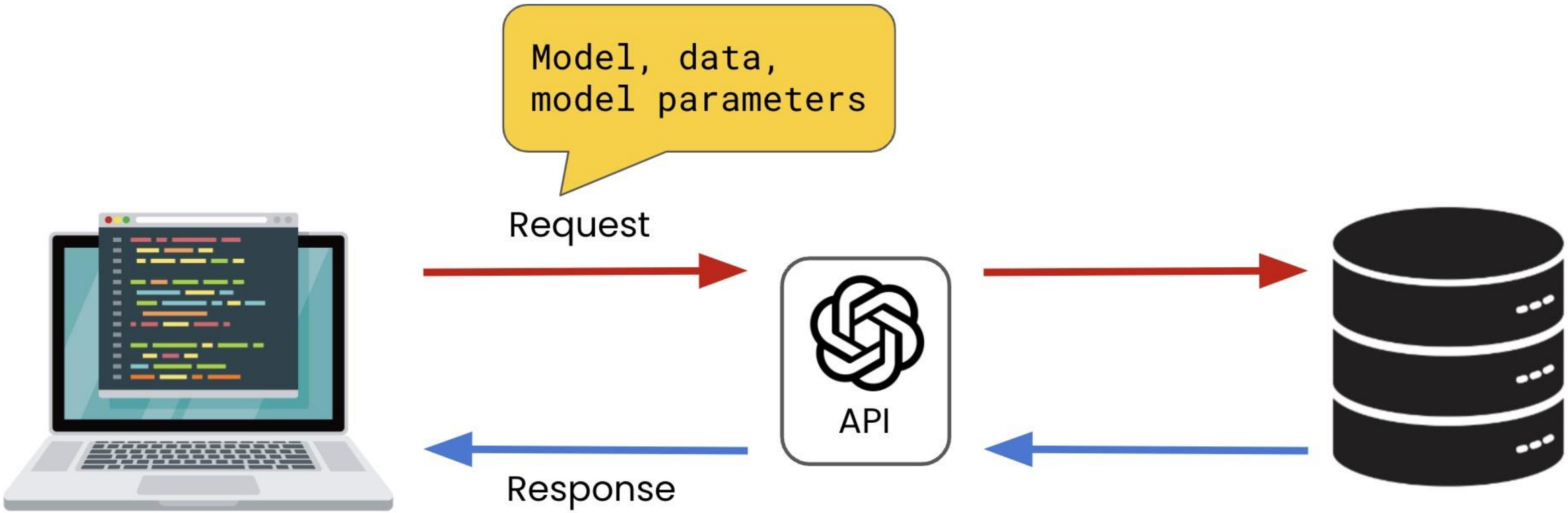
What is an API?



The OpenAI API

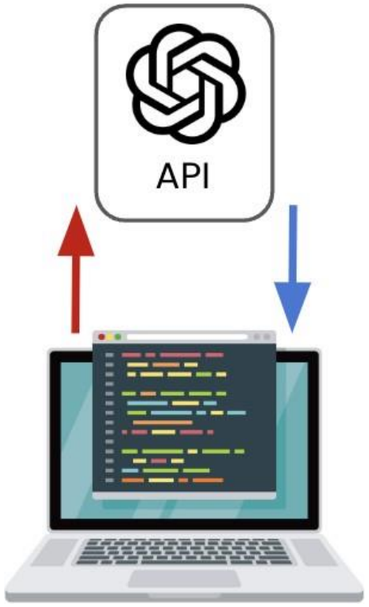


The OpenAI API

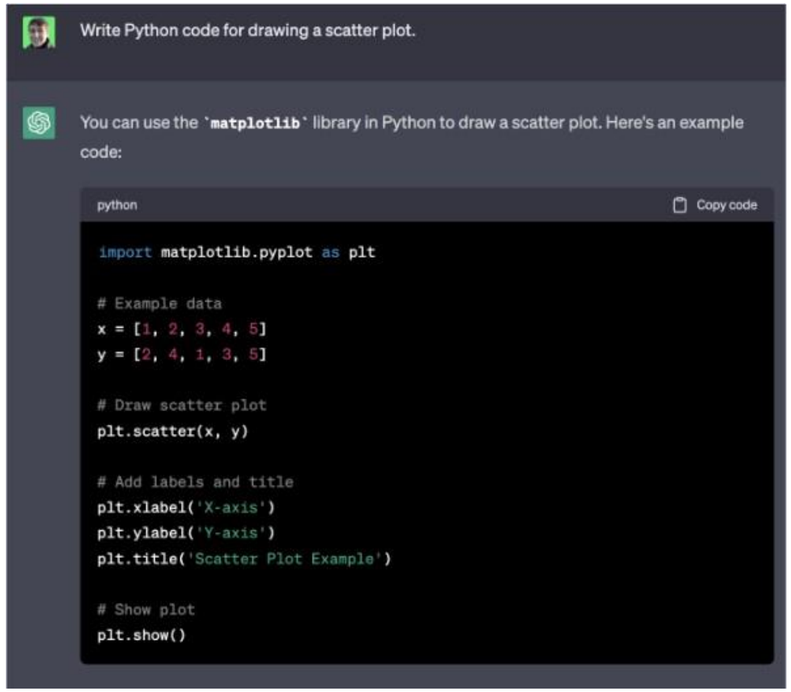


API vs. web interface

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

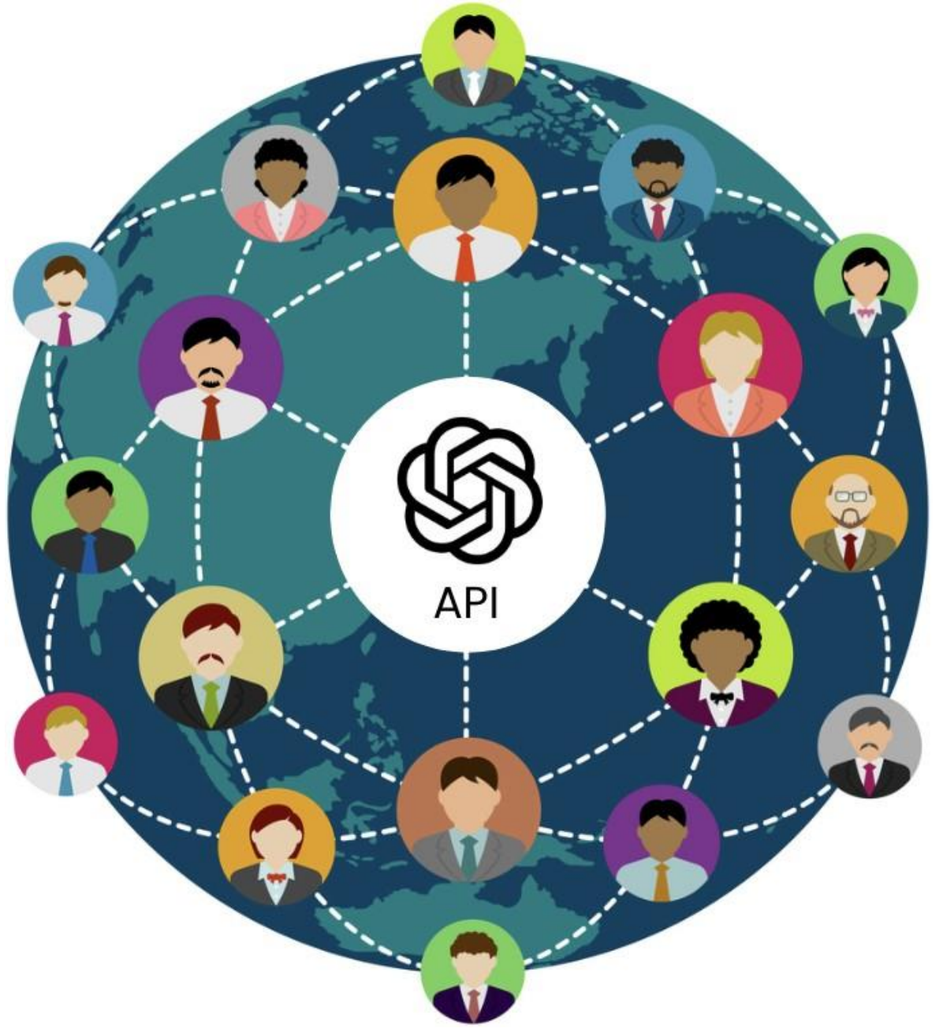


- Very little setup
- Sufficient for *streamlining* individual's workflows



Why the OpenAI API?

- Accessibility to AI
- Don't require data or ML experience

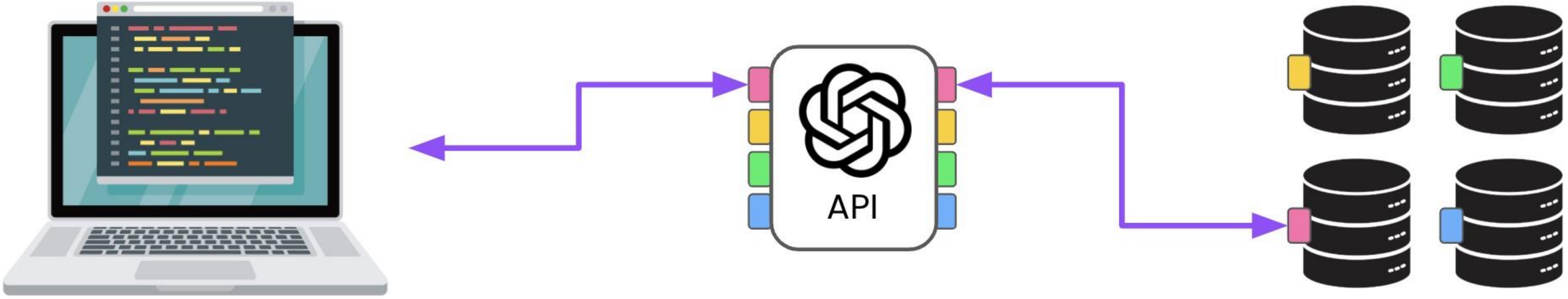


TriveraTech
TECHNOLOGY TRAINING

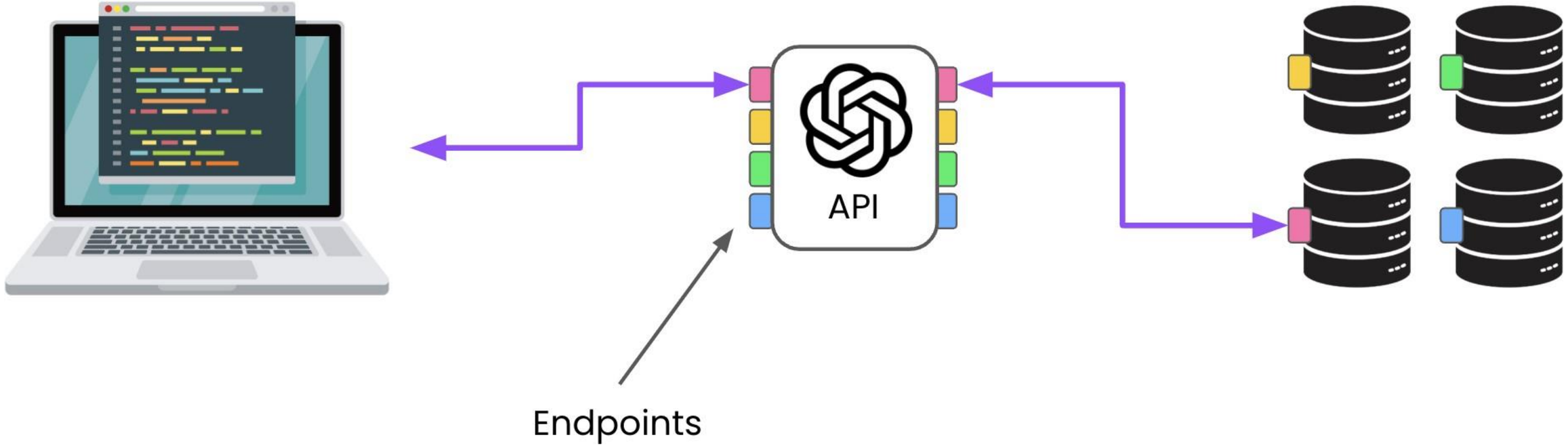
Making Requests to the OpenAPI

Experience is Everything

API endpoints

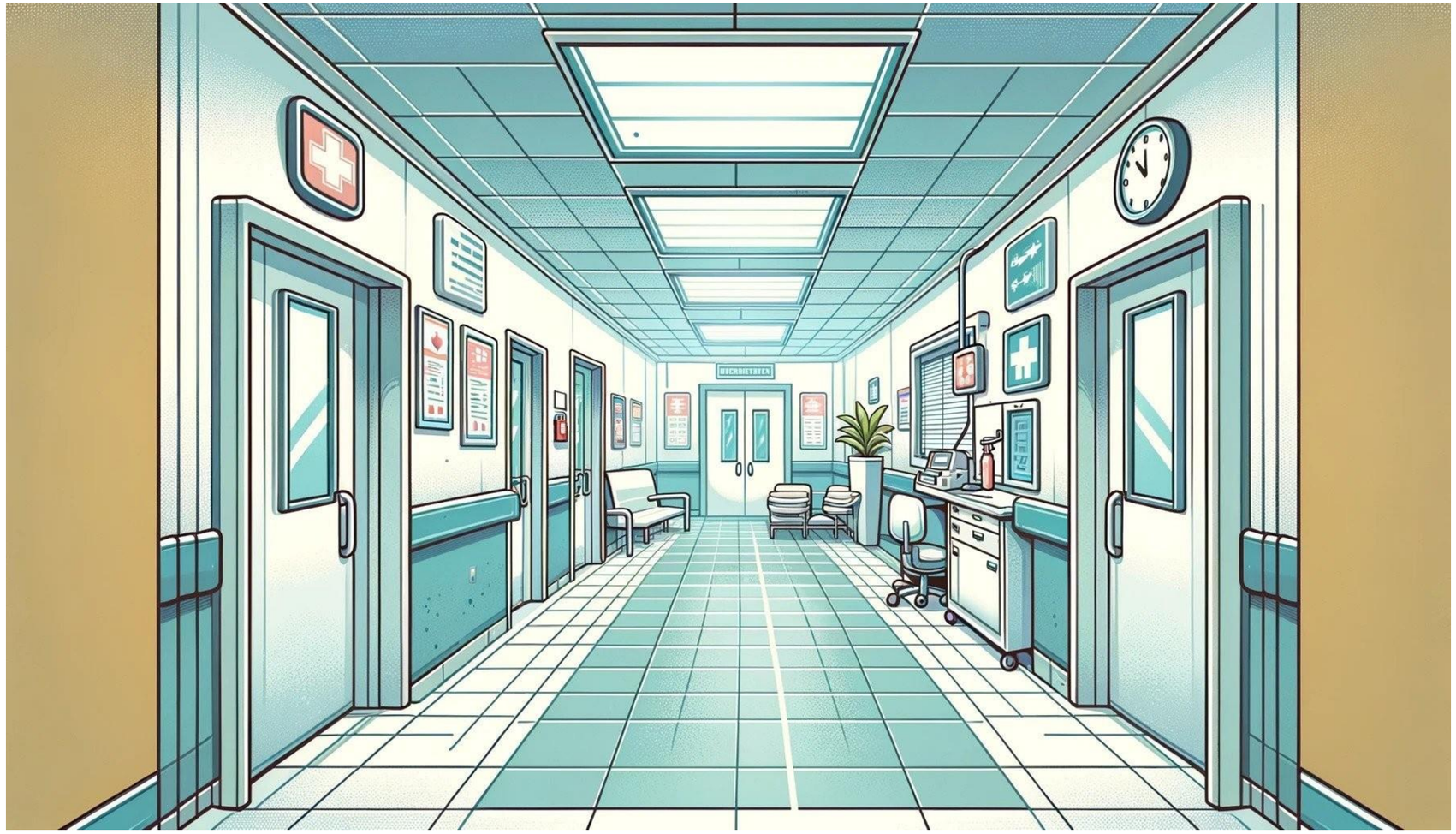


API endpoints



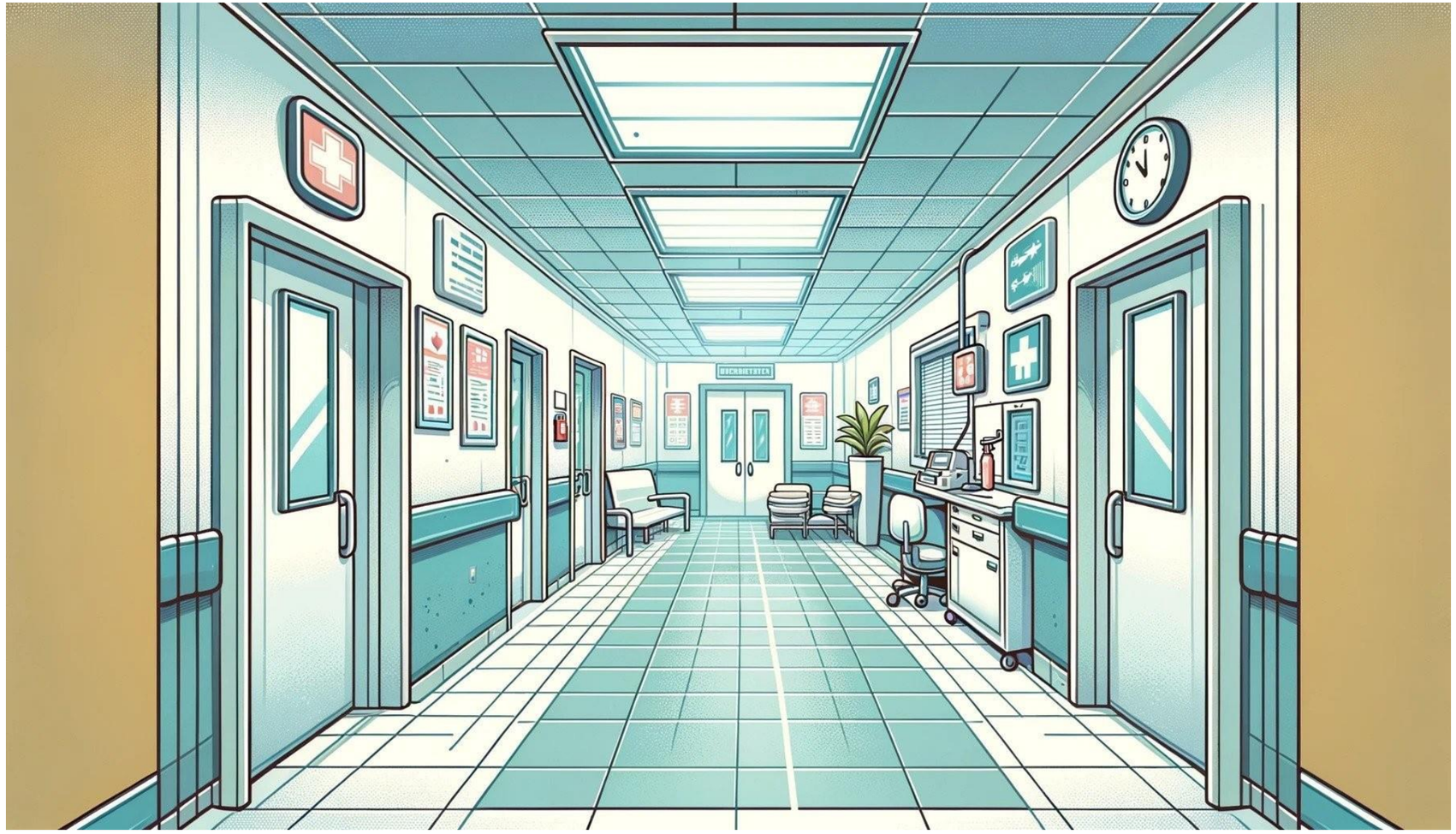
API endpoints

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



API authentication

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



API usage costs

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

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

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**

NAME	KEY	CREATED	LAST USED
Secret key	sk-...Vpp5	5 Feb 2023	25 Apr 2023

+ Create new secret key

Making a request

```
from openai import OpenAI

client = OpenAI(api_key="ENTER YOUR KEY HERE")


response = client.chat.completions.create(

    model="gpt-4o-mini",

    messages=[{"role": "user", "content": "What is the OpenAI API?"}]

)


print(response)
```

The response

```
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}))
```

The response

```
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}))
```

Interpreting the response

```
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))]
```

Interpreting the response

```
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))
```

Interpreting the response

```
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.
```

TriveraTech
TECHNOLOGY TRAINING

Generating and transforming text

Experience is Everything

Recap...

- **Q&A**

```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[{"role": "user", "content": "How many days are in October?"}]
)

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

```
October has 31 days.
```

- Text *most likely* to complete the prompt

```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[{"role": "user", "content": "Life is like a box of chocolates."}]
)

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

```
You never know what you're going to get. This famous quote from the movie
"Forrest Gump"...
```

- Response is **non-deterministic** (inherently random)

Controlling response randomness

- `temperature` : control on *determinism*
- Ranges from `0` (highly deterministic) to `2` (very random)

```
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[{"role": "user", "content": "Life is like a box of chocolates."}],
    temperature=2
)
print(response.choices[0].message.content)
```

```
"...you never know what you're gonna get." That quote reminds us of the
unpredictability of life and the diverse set of experiences we might encounter.
Whether sweet, nutty, bitter, or flashy, life holds a treasure trove of surprises.
```

Content transformation

- Changing text based on an instruction
  - Find and replace
  - Summarization
  - Copyediting

```
prompt = """
Update name to Dr. Lee, pronouns to he/him, and job title to CIO in the following
text:

Joanne is an executive at Trivera. Her favorite programming language is Python,
which she uses for her statistical analyses.
"""
```

Content transformation

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

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

```
Dr. Lee is the CIO at Trivera. His favorite programming language is Python, which he
uses for his statistical analyses.
```

```python
prompt = "Create a tagline for a new hot dog stand."

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

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

```
"Frankly, we've got the BEST dogs in town!"
```

Controlling response length

Default max_tokens

```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  prompt="Write a haiku about AI."
)

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

```
AI so powerful
Computers that think and learn
Superseding
```

Controlling response length

Default max_tokens

```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  prompt="Write a haiku about AI."
)

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

```
AI so powerful
Computers that think and learn
Superseding
```

max_tokens = 30

```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  prompt="Write a haiku about AI.",
  max_tokens=30
)

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

```
A machine mind thinks
Logic dictates its choices
Mankind ponders anew
```

TriveraTech
TECHNOLOGY TRAINING

How can the OpenAI API deliver business value?

Understanding tokens

Experience is Everything

Returning to cost

- Usage costs dependent on amount of input and output text
  - Models are priced by *cost/tokens*
  - Input and output tokens can be priced differently
- Increasing `max_tokens` increases cost
- Scoping feature cost often starts with a rough calculation:

$$\frac{\text{Cost}}{\text{Time}} = \text{Avg. Tokens Generated} \times \text{Model Cost} \times 1000 \times \frac{\text{Expected no. of requests}}{\text{Time}}$$

TriveraTech
TECHNOLOGY TRAINING

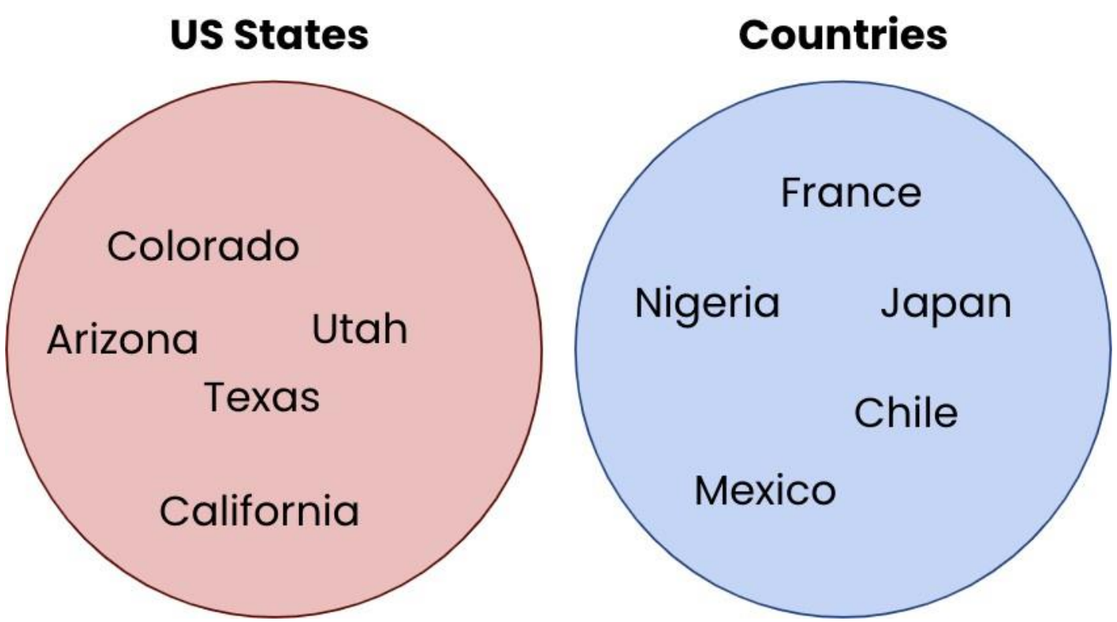
Sentiment analysis and classification

Experience is Everything

Classification tasks

- Task that involves assigning a label to information
  - Identifying the language from text
  - Categorization
  - Classifying sentiment

- OpenAI models can perform these tasks, providing:
  - Model has sufficient knowledge
  - Prompt contains sufficient context



Categorizing animals

```
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[{"role": "user", "content": "Classify the following animals into
    categories: zebra, crocodile, blue whale, polar bear, salmon, dog."}],
    max_tokens=50
)
print(response.choices[0].message.content)
```

```
Here are the animals classified into categories based on their general classifications:


Mammals: Zebra, Polar Bear, Dog

Fish: Salmon

Reptiles: Crocodile
```

Specifying groups

```
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[{"role": "user", "content": "Classify the following animals into animals
    with fur and without: zebra, crocodile, dolphin, polar bear, salmon, dog."}],
    max_tokens=50
)

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

```
Sure! Here's the classification of the animals you provided:


Animals with fur: Dog, Polar Bear, Zebra

Animals without fur: Crocodile, Dolphin, Salmon
```

Classifying sentiment

```
prompt = """Classify sentiment in the following statements:
1. The service was very slow
2. The steak was awfully tasty!
3. Meal was decent, but I've had better.
4. My food was delayed, but drinks were good.
"""
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[{"role": "user", "content": prompt}],
    max_tokens=50
)
print(response.choices[0].message.content)
```

Classifying sentiment

1. The service was very slow
2. The steak was awfully tasty!
3. Meal was decent, but I've had better.
4. My food was delayed, but drinks were good.

```
1. Negative
2. Positive
3. Neutral
4. Mixed
```

Classifying sentiment

```
prompt = """Classify sentiment as 1-5 (bad-good) in the following statements:
1. The service was very slow
2. The steak was awfully tasty!
3. Meal was decent, but I've had better.
4. My food was delayed, but drinks were good.
"""
```

```
1. 1
2. 5
3. 3
4. 2
```

Zero-shot vs. one-shot vs. few-shot prompting

- **Zero-shot** prompting: *no examples* provided

In-context learning:

- **One-shot** prompting: *one example* provided
- **Few-shot** prompting: *multiple examples* provided

```python
prompt = """Classify sentiment in the following statements:
The service was very slow // Disgruntled
Meal was decent, but I've had better. //
"""

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

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

```
Neutral to Discontented
```

```python
prompt = """Classify sentiment in the following statements:
The service was very slow // Disgruntled
The steak was awfully tasty! // Delighted
Good experience overall. // Satisfied
Meal was decent, but I've had better. //
"""

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

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

```
Neutral/Discontented
```

TriveraTech
TECHNOLOGY TRAINING

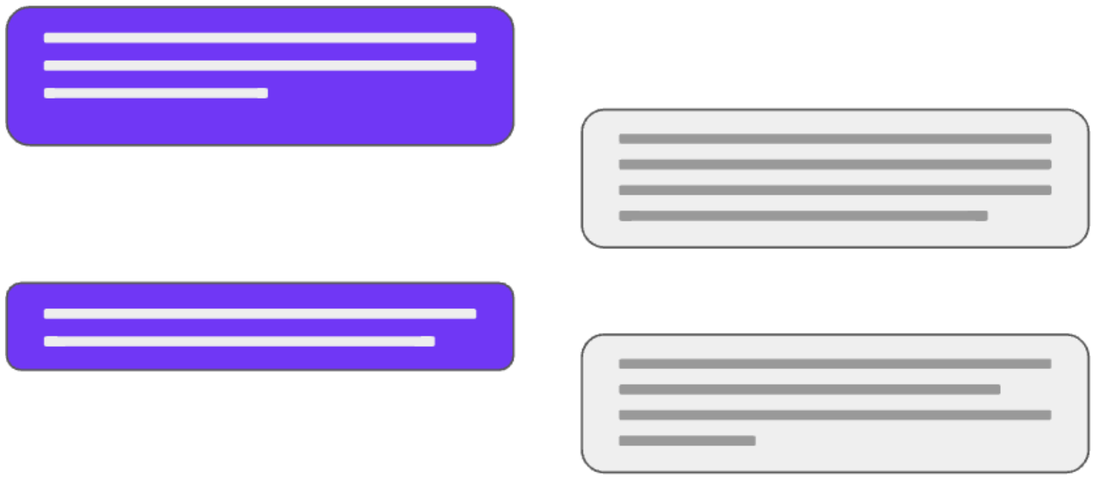
Chat completions with GPT

Experience is Everything

The Chat Completions endpoint

Single-turn tasks

- Text generation
- Text transformation
- Classification

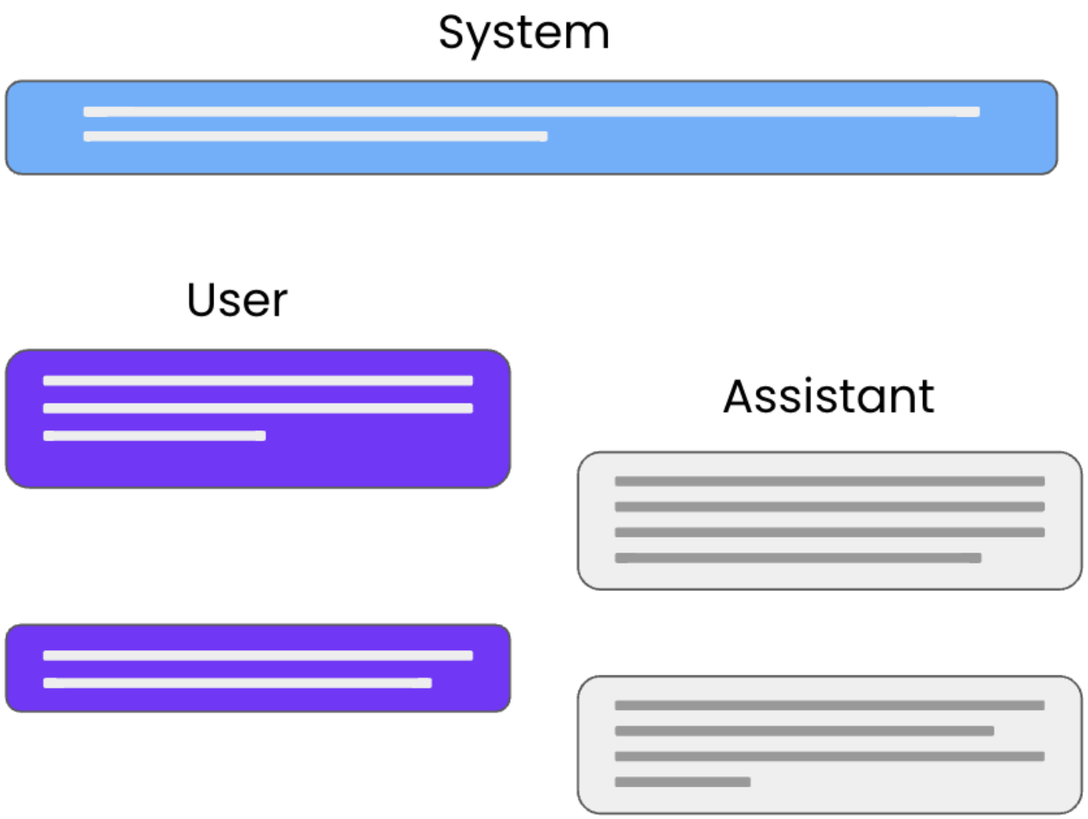


Multi-turn conversations

* Build on previous prompts and responses

Roles

- **System**: controls assistant's *behavior*
- **User**: *instruct* the assistant
- **Assistant**: response to user instruction
  - Can also be written by the user to provide examples



Request

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

Prompt setup

```
messages=[{"role": "system",
           "content": "You are a data science tutor who speaks concisely."},
          {"role": "user",
           "content": "What is the difference between mutable and immutable objects?"}]
```

Making a request

```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[{"role": "system",
             "content": "You are a data science tutor who speaks concisely."},
            {"role": "user",
             "content": "What is the difference between mutable and immutable objects?"}]
)

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

The response

```
Mutable objects can be changed after creation, while immutable objects cannot be
modified once they are created.
```

TriveraTech
TECHNOLOGY TRAINING

Multi-turn chat completions with GPT

Experience is Everything

Chat completions for single-turn tasks

```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[{"role": "system",
             "content": "You are a data science tutor."},
            {"role": "user",
             "content": "What is the difference between mutable and immutable objects?"}]
)
```

- **System**: controls assistant's *behavior*
- **User**: *instruct* the assistant
- **Assistant**: response to user instruction

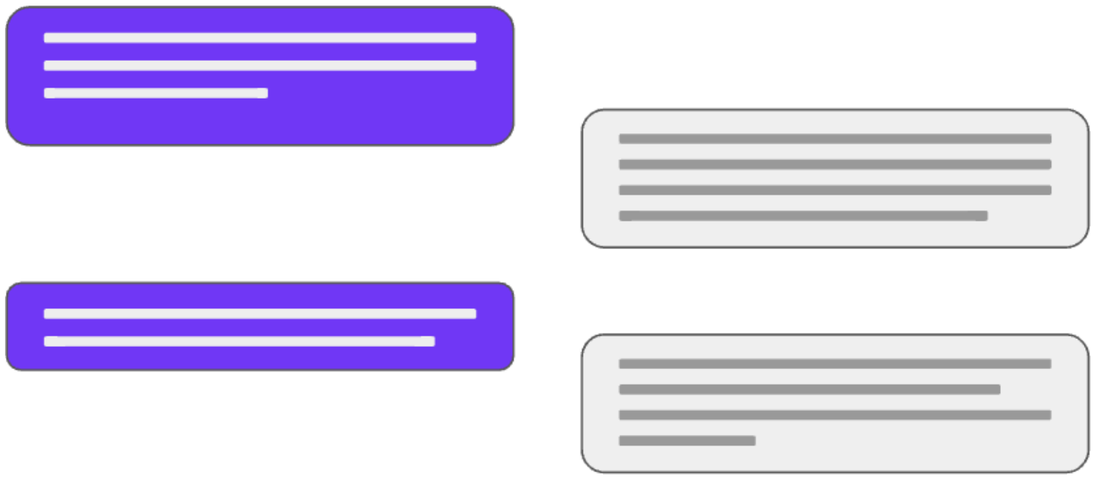
```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[{"role": "system",
             "content": "You are a data science tutor who speaks concisely."},
            {"role": "user",
             "content": "How do you define a Python list?"},
            {"role": "assistant",
             "content": "Lists are defined by enclosing a comma-separated sequence of
                         objects inside square brackets [ ]."},
            {"role": "user",
             "content": "What is the difference between mutable and immutable objects?"}]
)
```

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

```
Mutable objects are objects whose values can change after they are created. Examples of
mutable objects in Python include lists, sets and dictionaries. Immutable objects are
objects whose values cannot change after they are created. Examples of immutable objects
in Python include strings, numbers and tuples.
```

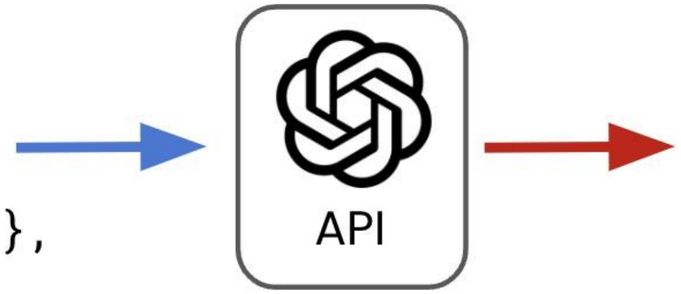
Storing responses

- Create conversation history
- Create back-and-forth conversations



Building a conversation

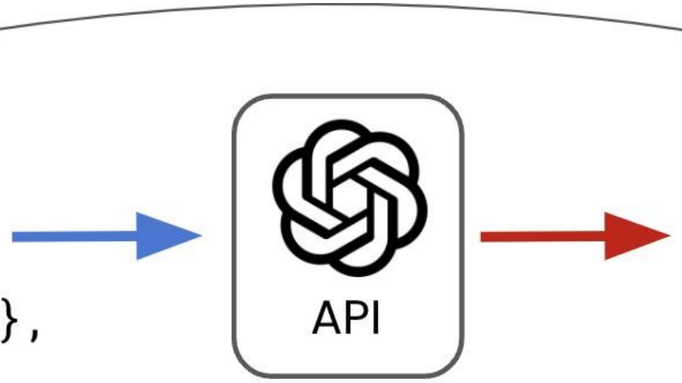
```
messages = [{"role": "system",
             "content": "Behavior…"},
            {"role": "user",
             "content": "My name is James."}]
```




```
{"role": "assistant",
 "content": "Hi James!"}
```

Building a conversation

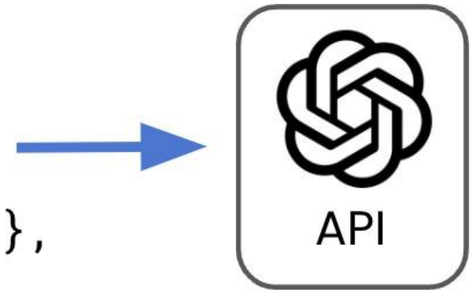
```
messages = [{"role": "system",
             "content": "Behavior…"},
            {"role": "user",
             "content": "My name is James."}]
```




```
{"role": "assistant",
 "content": "Hi James!"}
```

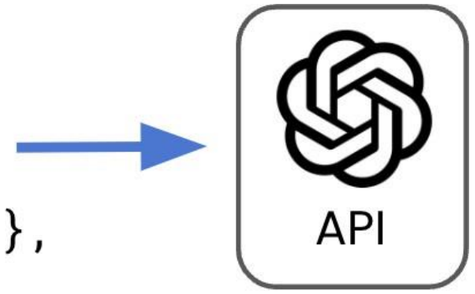
Building a conversation

```
messages = [{“role”: “system”,
             “content”: “Behavior…”},
            {“role”: “user”,
             “content”: “My name is James.”},
            {“role”: “assistant”,
             “content”: “Hi James!”}]
```



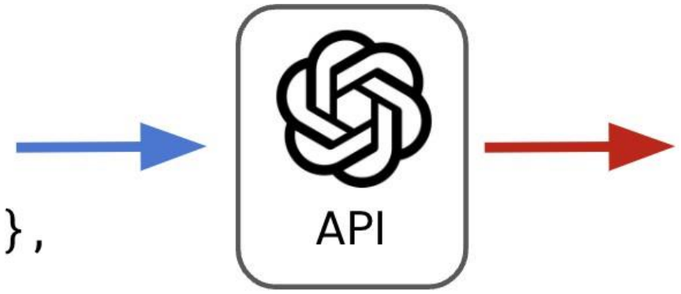
Building a conversation

```
messages = [{“role”: “system”,
             “content”: “Behavior…”},
            {“role”: “user”,
             “content”: “My name is James.”},
            {“role”: “assistant”,
             “content”: “Hi James!”},
            {“role”: “user”,
             “content”: “What is my name?”}]
```



Building a conversation

```
messages = [{"role": "system",
             "content": "Behavior…"},
            {"role": "user",
             "content": "My name is James."},
            {"role": "assistant",
             "content": "Hi James!"},
            {"role": "user",
             "content": "What is my name?"}]
```




```
{"role": "assistant",
 "content": "Your name
is James."}
```

Coding a conversation

```
messages = [{"role": "system",
             "content": "You are a data science tutor who provides short, simple explanations."}]

user_qs = ["Why is Python so popular?", "Summarize this in one sentence."]

for q in user_qs:
    print("User: ", q)
    user_dict = {"role": "user", "content": q}
    messages.append(user_dict)

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

    assistant_dict = {"role": "assistant", "content": response.choices[0].message.content}
    messages.append(assistant_dict)
    print("Assistant: ", response.choices[0].message.content, "\n")
```

Conversation with an AI

```
User:  Why is Python so popular?
Assistant:  Python is popular for many reasons, including its simplicity,
versatility, and wide range of available libraries. It has a relatively
easy-to-learn syntax that makes it accessible to beginners and experts alike. It
can be used for a variety of tasks, such as data analysis, web development,
scientific computing, and machine learning. Additionally, Python has an active
community of developers who contribute to its development and share their
knowledge through online resources and forums.

User:  Summarize this in one sentence.
Assistant:  Python is popular due to its simplicity, versatility, wide range of
libraries, and active community of developers.
```

TriveraTech
TECHNOLOGY TRAINING

Text moderation

Experience is Everything

Going beyond text completions...

Completions →generate *new text output* using text prompt

Beyond completions:

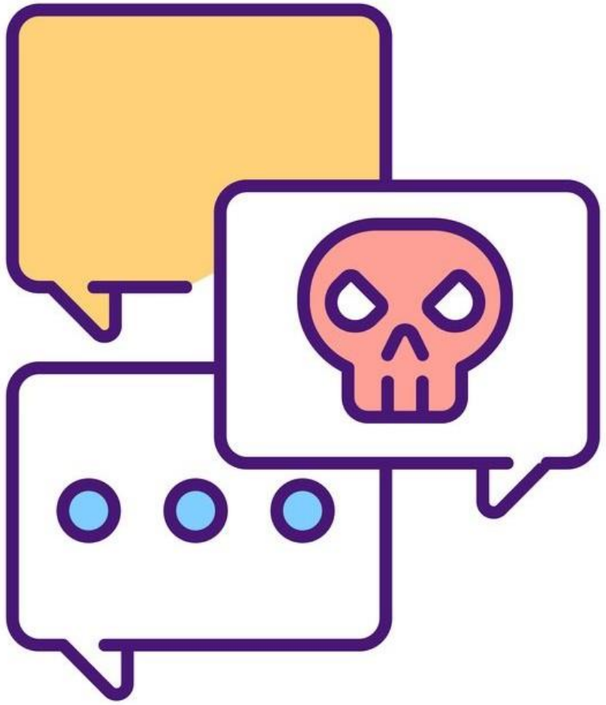
- Text moderation
- Audio transcription and translation
- Combining models together

Text moderation

- Identifying inappropriate content

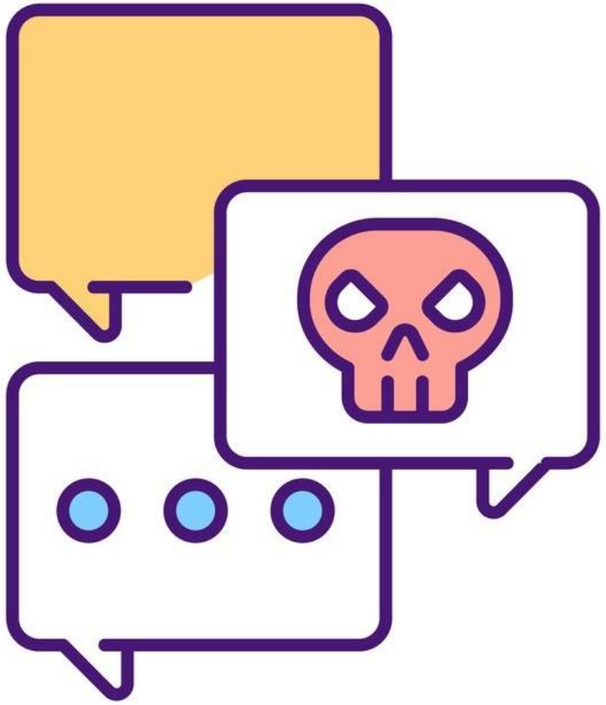
Traditionally,

- Moderators flag content *by-hand*
  - Time-consuming
- **Keyword pattern matching**
  - Lacks *nuance* and understanding of *context*



Violation categories

- Identify violations of terms or use
- Differentiate violation type by category
  - Violence
  - Hate Speech



Creating a moderations request

```
from openai import OpenAI

client = OpenAI(api_key="ENTER API KEY")


response = client.moderations.create(

  model="text-moderation-latest",

  input="I could kill for a hamburger."

)
```

Interpreting the results

- `categories`
  - `true` / `false` indicator of category violation
- `category_scores`
  - Confidence of a violation
- `flagged`
  - `true` / `false` indicator of a violation

```python
print(response.model_dump())
```

```
{'id': 'modr-8S80XeaVqvs4mmbufDBP9gTAmEGXP',
 'model': 'text-moderation-006',
 'results':
   [{'categories': {'harassment': False,
                    'harassment_threatening': False,
                    'hate': False,
                    'hate_threatening': False,
                    ...},
     'category_scores': {'harassment': 2.775940447463654e-05,
                         'harassment_threatening': 1.3526056363843963e-06,
                         'hate': 2.733528674525587e-07,
                         'hate_threatening': 4.930571506633896e-08,
                         ...},
     'flagged': False}]
}
```

Interpreting the category scores

```
print(response.results[0].category_scores)
```

```
CategoryScores(harassment=2.775940447463654e-05,
               harassment_threatening=1.3526056363843963e-06,
               hate=2.733528674525587e-07,
               hate_threatening=4.930571506633896e-08,
               ...,
               violence=0.0500854030251503,
               ...)
```

- Larger numbers →greater certainty of violation
- Numbers ≠ probabilities

Considerations for implementing moderation

```
CategoryScores(harassment=2.775940447463654e-05,
               harassment_threatening=1.3526056363843963e-06,
               hate=2.733528674525587e-07,
               hate_threatening=4.930571506633896e-08,
               ...,
               violence=0.05008540302515 03,
               ...)
```

- Determine appropriate thresholds for each use case
- Stricter thresholds may result in fewer *false negatives*
- More lenient thresholds may result in fewer *false positives*

Whisper

TriveraTech
TECHNOLOGY TRAINING

Speech-to-Text Transcription with Whisper

Experience is Everything

OpenAIs Whisper

Speech-to-text capabilities:

- Transcribe audio
- Translate and transcribe audio into *English*
- Supports `mp3`, `mp4`, `mpeg`, `mpga`, `m4a`, `wav`, and `webm` (25 MB limit)

- Meeting transcripts
- Video captions

Loading audio files

Example: transcribe `meeting_recording_20230602.mp3`

```
audio_file = open("meeting_recording_20230602.mp3", "rb")
```

If the file is located in a *different directory*

```
audio_file = open("path/to/file/meeting_recording_20230602.mp3", "rb")
```

Making a request

- Audio endpoint

```
audio_file= open("meeting_recording_20230602.mp3", "rb")

response = client.audio.transcriptions.create(model="whisper-1", file=audio_file)


print(response)
```

```
Transcription(text="Welcome everyone to the June product monthly. We'll get started in...)
```

The transcript

```
print(response.text)
```

```
Welcome everyone to the June product monthly. We'll get started in just a minute.
Alright, let's get started. Today's agenda will start with a spotlight from Chris
on the new mobile user onboarding flow, then we'll review how we're tracking on
our quarterly targets, and finally, we'll finish with another spotlight from Katie
who will discuss the upcoming branding updates...
```

- **Don't use sensitive or confidential recordings**

Transcribing non-English languages

Afrikaans, Arabic, Armenian, Azerbaijani, Belarusian, Bosnian, Bulgarian, Catalan, Chinese, Croatian, Czech, Danish, Dutch, English, Estonian, Finnish, French, Galician, German, Greek, Hebrew, Hindi, Hungarian, Icelandic, Indonesian, Italian, Japanese, Kannada, Kazakh, Korean, Latvian, Lithuanian, Macedonian, Malay, Marathi, Maori, Nepali, Norwegian, Persian, Polish, Portuguese, Romanian, Russian, Serbian, Slovak, Slovenian, Spanish, Swahili, Swedish, Tagalog, Tamil, Thai, Turkish, Ukrainian, Urdu, Vietnamese, and Welsh.

1. `open()` audio file
2. Make a transcriptions request to the Audio endpoint
3. Extract text from the response

TriveraTech
TECHNOLOGY TRAINING

Speech Translation with Whisper

Experience is Everything

Whisper's translation capabilities

- Translate and transcribe audio
- Currently limited to *English* transcripts
- Supports `mp3`, `mp4`, `mpeg`, `mpga`, `m4a`, `wav`, and `webm` (25 MB limit)

Translating audio

```
audio_file = open("non_english_audio.m4a", "rb")

response = client.audio.translations.create(model="whisper-1", file=audio_file)

print(response.text)
```

```
The search volume for keywords like A I has increased rapidly since the launch of
Cha GTP.
```

- Performance can vary wildly, depending on:
  - Audio quality
  - Audio language
  - Model's knowledge of the subject matter

Bringing prompts into the mix

- Can provide `prompt` to the model (optional)
- Improve response quality by:
  - Providing an example of desired style
  - Provide additional context about transcript

Example: Retaining filler words

```
prompt="Ok, ummm... this is what we should do, like, to uhhh... increase revenue."
```

Example: Provide context

```
prompt="A discussion on how to increase revenue."
```

Adding in a prompt

```
audio_file = open("non_english_audio.m4a", "rb")
prompt = "The transcript is about AI trends and ChatGPT."

response = client.audio.translations.create(model="whisper-1",
                                            file=audio_file,
                                            prompt=prompt)

print(response.text)
```

```
The search volume for keywords like AI has increased rapidly since the launch of
ChatGPT.
```

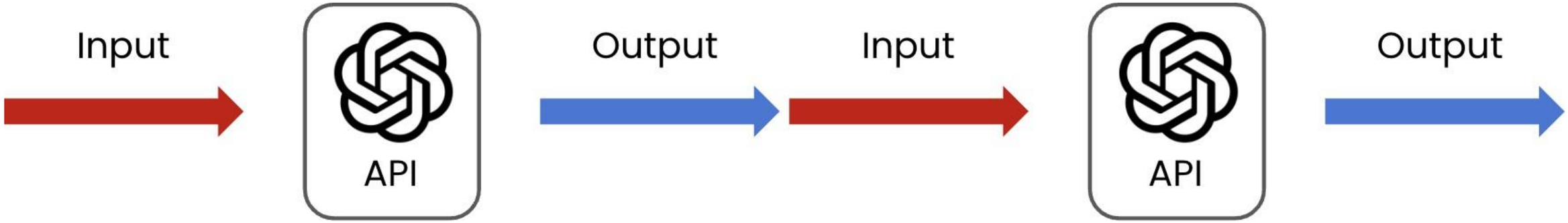
TriveraTech
TECHNOLOGY TRAINING

Combining models

Experience is Everything

Combining models

Combining models



- **Chaining**: Feeding output from one model into another
- Can use the *same model* multiple times
  - **Example**: validating original response
- Or *different models*
  - **Example**: summarizing meeting recordings

Example: Extracting meeting at endees

```
audio_file = open("meeting_recording_20230608.mp4", "rb")

audio_response = client.audio.transcriptions.create(model="whisper-1", file=audio_file)


transcript = audio_response.text

prompt = "Extract the attendee names from the start of this meeting transcript: " + transcript


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


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

Example: Extracting meeting at endees

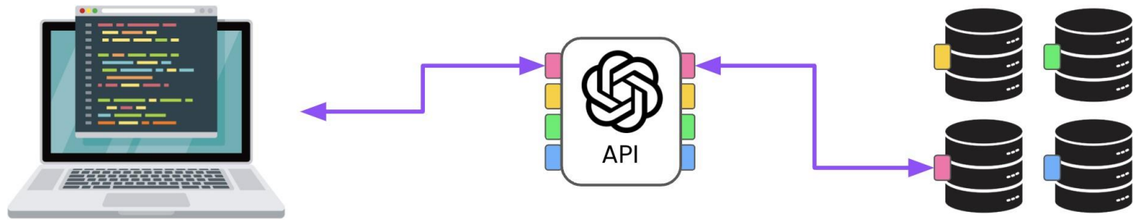
```
The meeting attendees were Otis, Paul, Elaine, Nicola, Alan, and Imran.
```

Note:

- **No guarantees** on model performance
- Ensure that applications are **well-tested**
- Usage should be restricted to **non-sensitive data**

Review 1

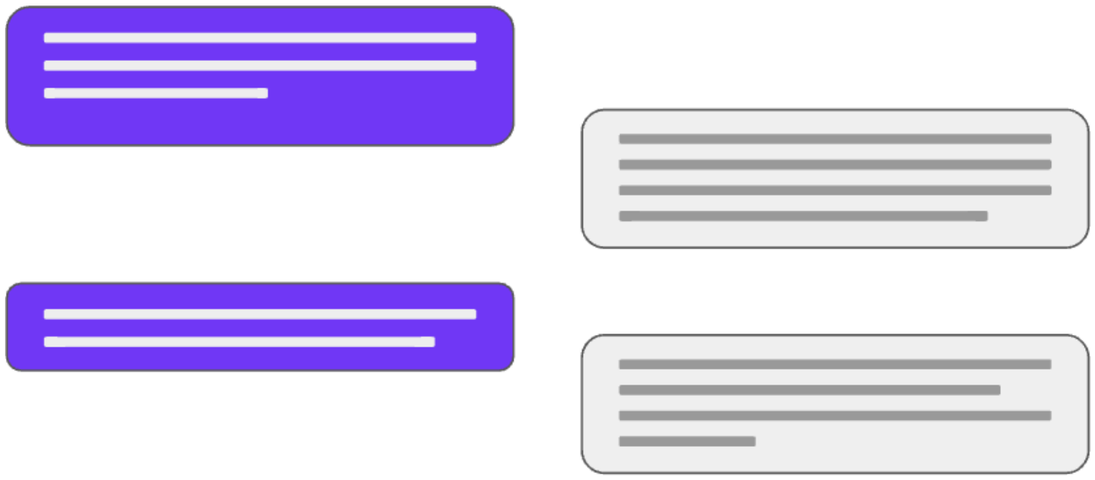
- What the OpenAI API is used for
- How to create requests to the Chat Completions endpoint




```
client = OpenAI(api_key="<OPENAI_API_TOKEN>")

response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[{"role": "user", "content": "..."}]
)
print(response.choices[0].message.content)
```

Review 2

```
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[{"role": "user", "content": "..."}],
  max_tokens=20,
  temperature=0.5
)
```



- Q&A
- Text transformation
- Content generation
- Sentiment analysis
- Categorization
- `max_tokens` and `temperature`
- Multi-turn conversation with chat *roles*

Review 3

Moderation

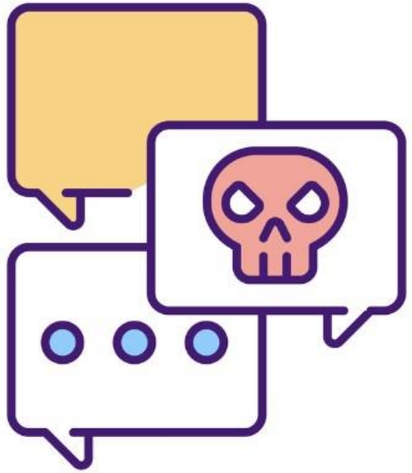
- Detect inappropriate content

Audio

- Translate
- Transcribe

Chaining

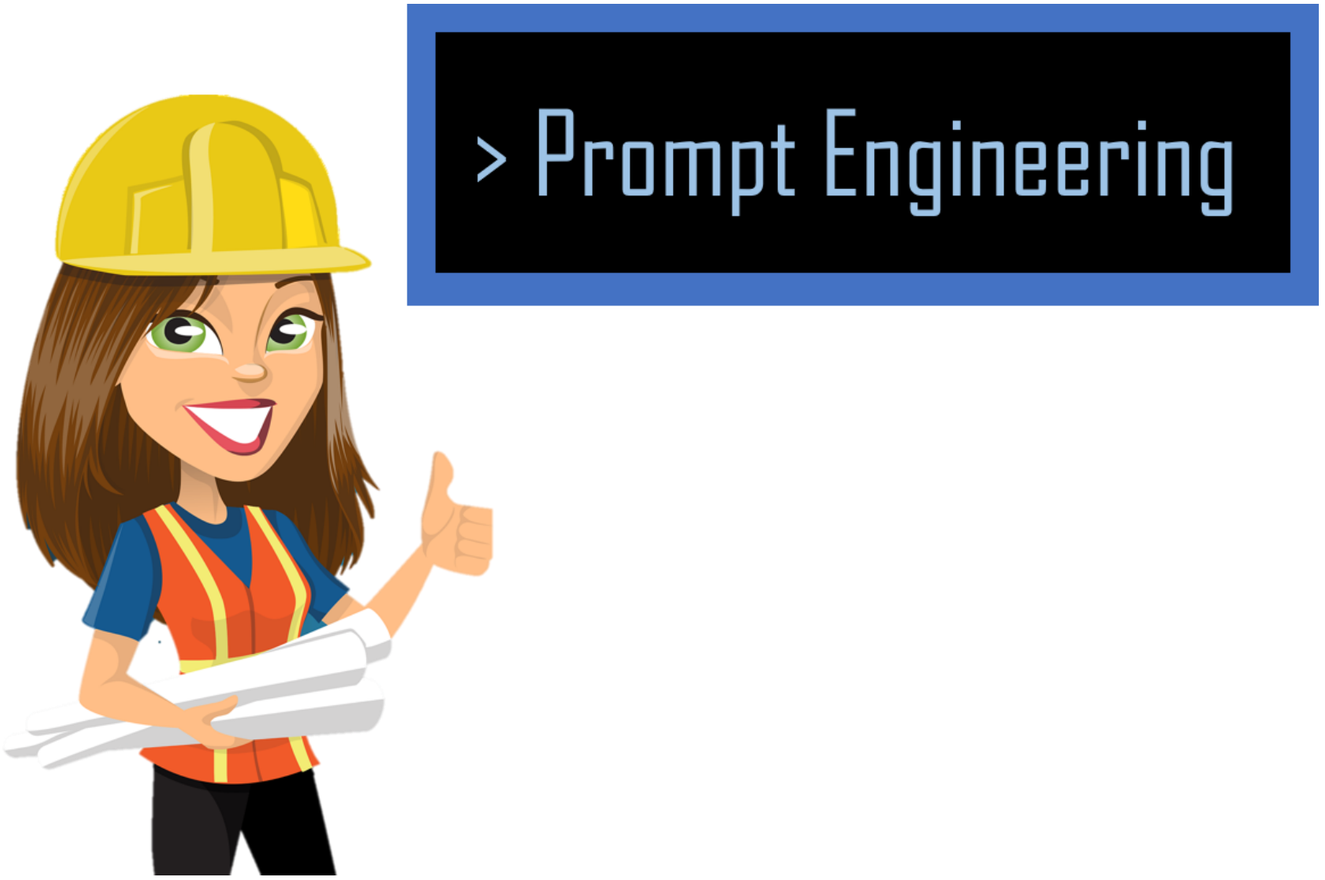
- Automating complex tasks



Prompt Engineering

What is prompt engineering?

Crafting prompts or instructions given to LLMs to get desired responses



Why prompt engineering?

Why prompt engineering?

Recap: message roles

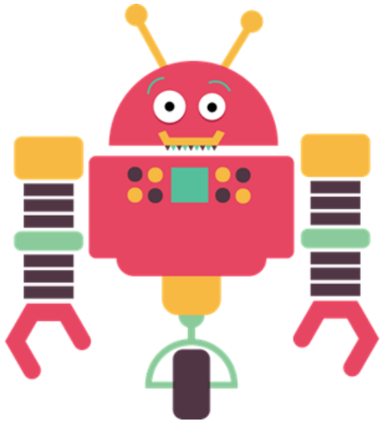
Every message has one of **three** roles

User

Recap: message roles

Every message has one of **three** roles

User

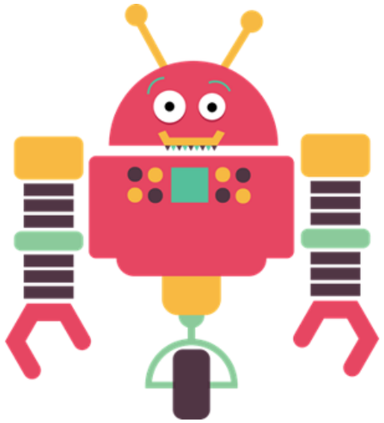


Assistant

Recap: message roles

Every message has one of **three** roles

User

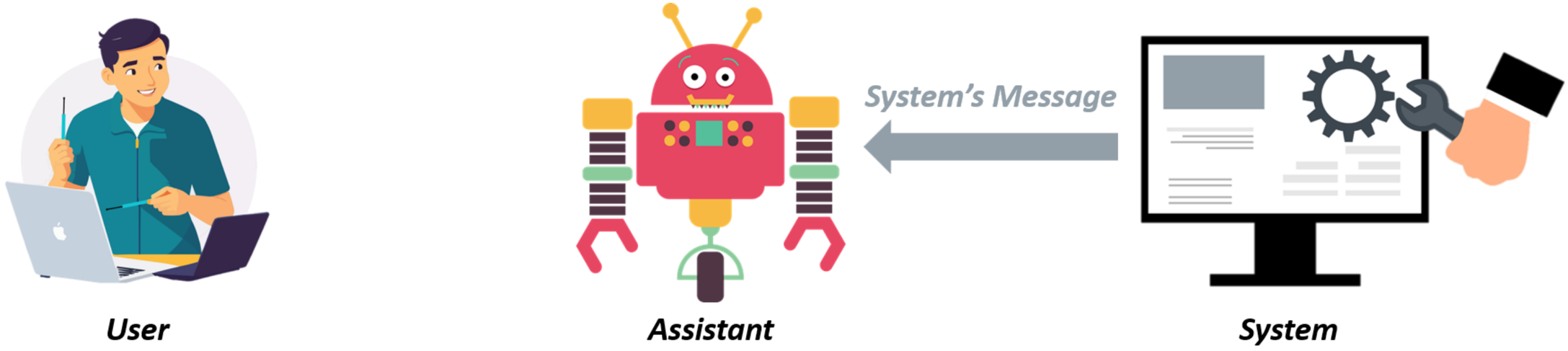


Assistant

System

Recap: message roles

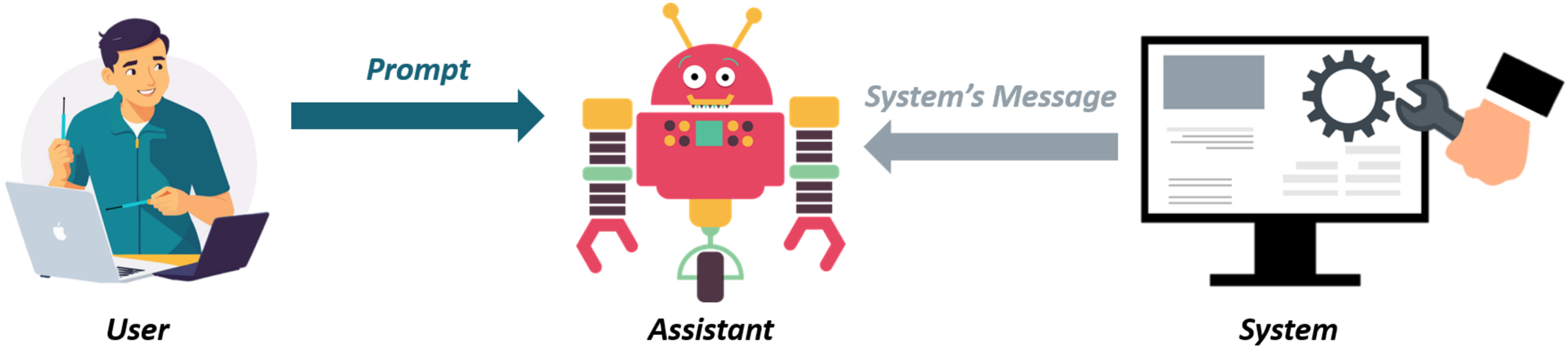
Every message has one of **three** roles



- **System message:** guides model behavior

Recap: message roles

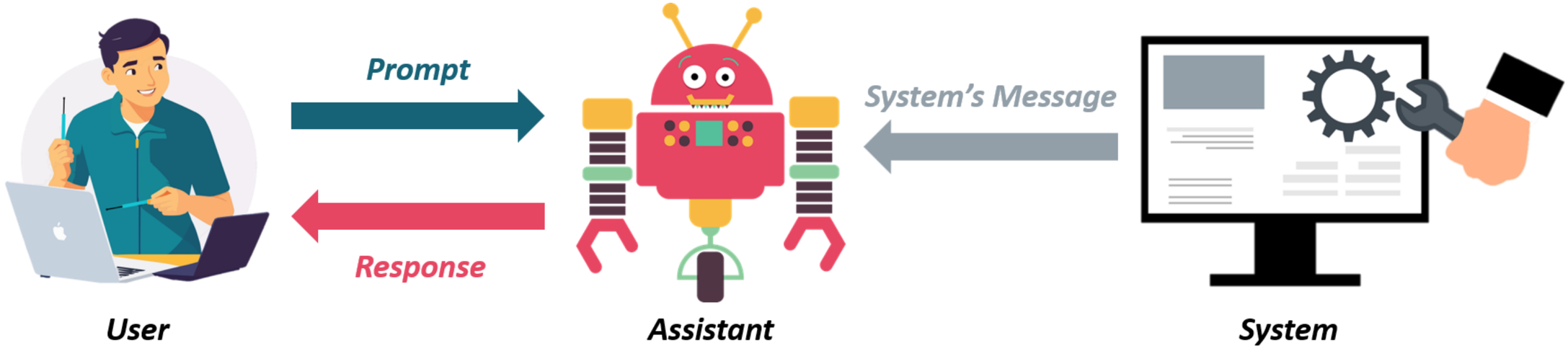
Every message has one of **three** roles



- **System message:** guides model behavior
- **User message:** prompt from the user

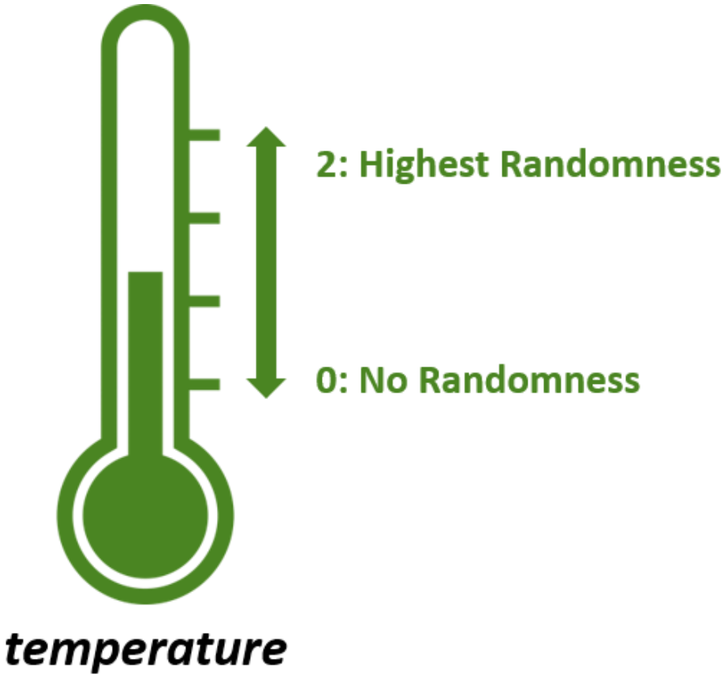
Recap: message roles

Every message has one of **three** roles



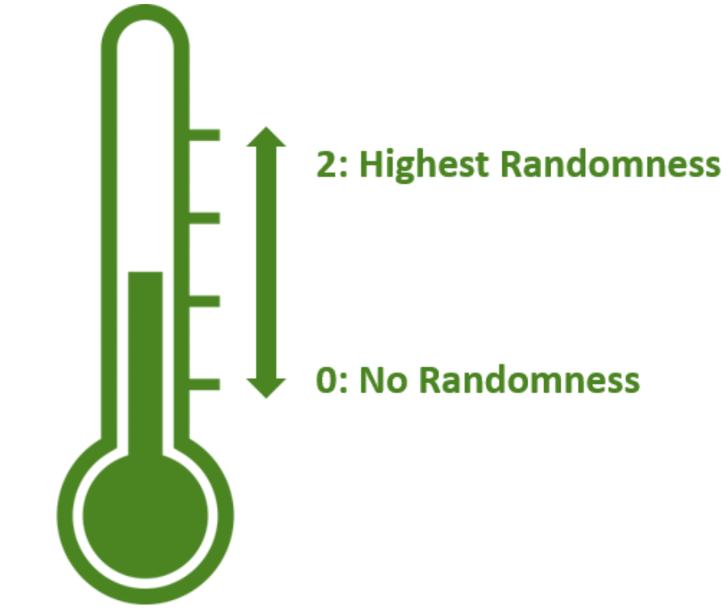
- **System message:** guides model behavior
- **User message:** prompt from the user
- **Assistant message:** response to user prompt

Recap: control parameters

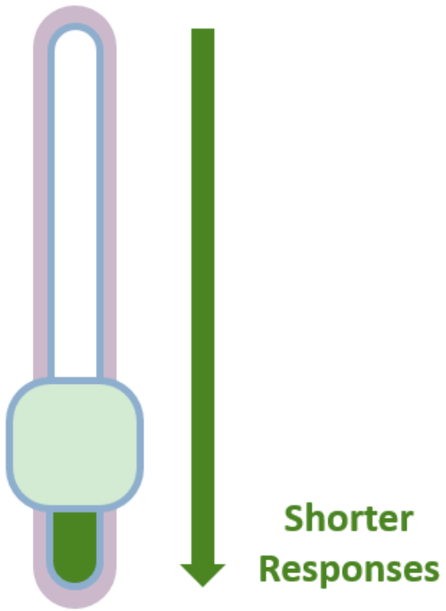


- `temperature` : controls answer's randomness

Recap: control parameters



temperature



max_tokens

- `temperature` : controls answer's randomness
- `max_tokens` : controls response length

Recap: communicating with the OpenAI API

```
prompt = "What is prompt engineering?"
client = OpenAI(api_key="api_key")
response = client.chat.completions.create(
  model = "gpt-3.5-turbo",
  messages = [{"role": "user",
               "content": prompt}],
  temperature = 0
)
print(response.choices[0].message.content)
```

```
Prompt engineering refers to the process of designing and refining prompts or
instructions given to a language model like ChatGPT in order to elicit desired
responses or behaviors. It involves formulating specific guidelines or hints to
guide the model's output towards a desired outcome.
```

Creating get_response() function

```
def get_response(prompt):
  response = client.chat.completions.create(
    model = "gpt-3.5-turbo",
    messages = [{"role": "user",
                 "content": prompt}],
    temperature = 0
  )
  return response.choices[0].message.content
```

Usage

```
response = get_response("What is prompt engineering?")
print(response)
```

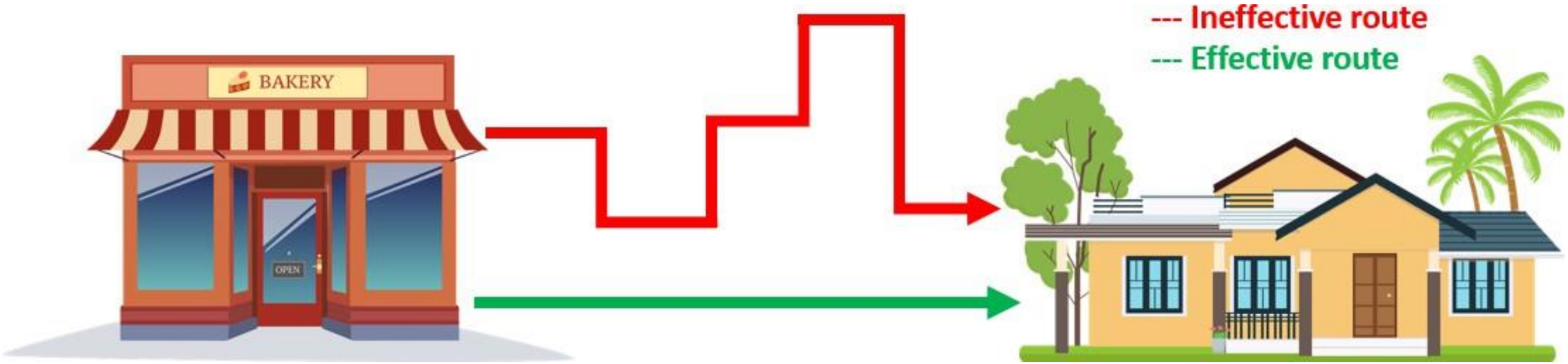
```
Prompt engineering refers to the process of designing and refining prompts or instructions given to a language model
like ChatGPT in order to elicit desired responses or behaviors. It involves formulating specific guidelines or hints
to guide the model's output towards a desired outcome.
```

Prompt improvement

```
prompt = "What is prompt engineering? Explain it in terms that can be understood
by a 5-year-old"
response = get_response(prompt)
print(response)
```

```
Imagine you have a very smart friend who can understand and answer lots of
questions. But sometimes, they might not understand exactly what you want or give
the wrong answer. So, prompt engineering is like giving your friend really clear
instructions or hints to help them give you the best answer possible.
```

Clear and precise prompts



Key principles

- □ Appropriate action verbs
- □ Detailed and precise instructions
- Well-structured delimited prompts

Using action verbs

- Guide model what to do

Verbs to use
Write
Complete
Explain
Describe
Evaluate
...

Avoid ambiguous verbs

- Confuse model's understanding

Verbs to use	***Verbs to avoid using***
Write	*Understand*
Complete	*Think*
Explain	*Feel*
Describe	*Try*
Evaluate	*Know*
…	…

Effective prompt

```
prompt = "Think about the issue of deforestation."
response = get_response(prompt)
print(response)
```

```
Deforestation is a significant environmental issue involving permanently removing
or destroying forests and woodlands.
It has far-reaching impacts on the environment, ecosystems, wildlife,
and human communities. [...]
```

```
prompt = "Propose strategies to reduce deforestation."
response = get_response(prompt)
print(response)
```

```
Reducing deforestation requires
a comprehensive and multi-dimensional approach involving various stakeholders.
Here are several strategies
that can help address the issue:
Strengthen Forest Governance
Promote Sustainable Land Use [...]
```

Formulating detailed instructions

Provide **specific, descriptive, and detailed instructions** regarding:

- □ Context
- □ Output length
- □ Format and style
- □□ Audience

Effective prompt with instructions

Ineffective prompt: ~~"Tell me about dogs."~~

Effective prompt

```
prompt = "Write a descriptive paragraph about the behavior and characteristics of
Golden Retrievers, highlighting their friendly nature, intelligence,
and suitability as family pets."
print(get_response(prompt))
```

```
Golden Retrievers are beloved worldwide for their exceptional behavior, remarkable
characteristics, and friendly nature. They are highly intelligent, trainable, and
adaptable, making them great companions for families, including those with children.
```

Limiting output length

max_tokens:

- Limit on number of tokens
- Output cannot bypass it
- Might lead to incomplete or cut responses

Prompt:

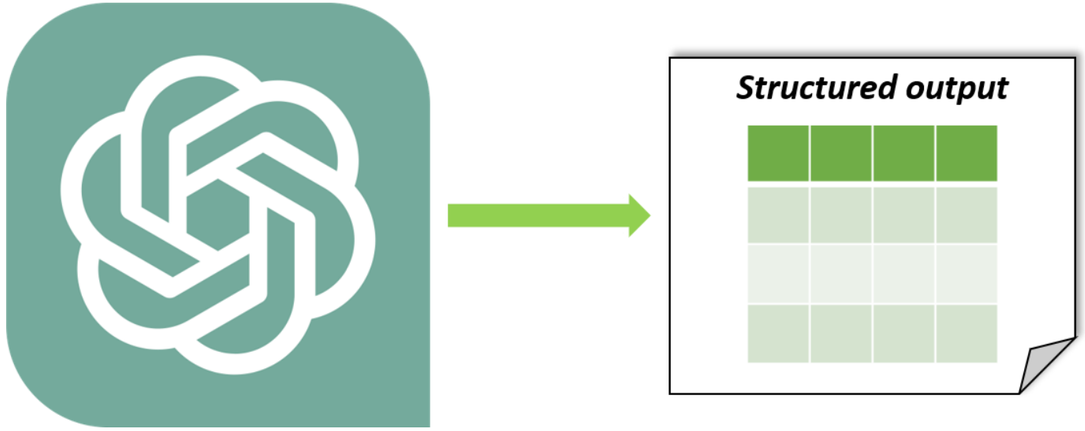
- Limit on words, sentences, or paragraphs
- Output may bypass it
- Complete responses

Prompt components

- **Instructions** and **input data** to operate on
- Example: text summarization
  - **Instruction:** summarize the given text
  - **Input data:** text to summarize

Crafting a well-structured prompt with delimiters

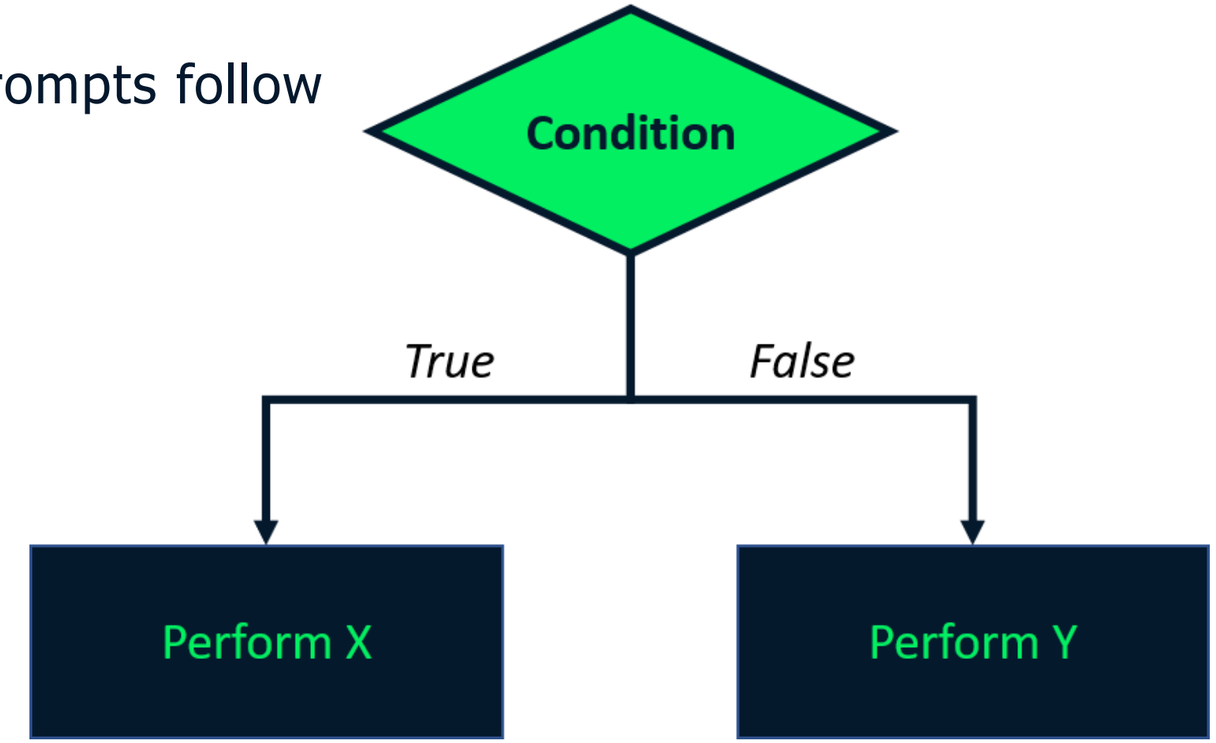
- Start prompt with instructions
- Use **delimiters** (parentheses, brackets, backticks, etc.) to specify input parts
- Mention which delimiters are used

```
prompt = """Summarize the text delimited by triple backticks into bullet points.
            ```TEXT GOES HERE```"""
response = get_response(prompt)
```
```

Using formatted strings (f-strings)

- Include defined string into another string

```
text = "This is a sample text to summarize"
prompt = f"""Summarize the text delimited by triple backticks into bullet points.
            ```{text}```"""
print(prompt)
```

```
Summarize the text delimited by triple backticks into bullet points.
```This is a sample text to summarize```
```

Structured outputs

- LLMs generate structured outputs only when given explicit instructions
- Output structures:
  - Table
  - List
  - Structured paragraph
  - Custom format



Tables

- Clearly mention expected columns

```
prompt = "Generate a table containing 5 movies I should watch if I am an action lover, with columns for Title and Rating."
print(get_response(prompt))
```

```
| Title                                  | Rating |
|----------------------------------------|--------|
| Mad Max: Fury Road                     | 8.1    |
| John Wick                              | 7.4    |
| The Dark Knight                        | 9.0    |
| Die Hard                               | 8.2    |
| Gladiator                              | 8.5    |
```

Lists

- Helpful for enumerations

```
prompt = "Generate a list containing the names of the top 5 cities to visit."
print(get_response(prompt))
```

```
1. Tokyo, Japan
2. Paris, France
3. Rome, Italy
4. New York City, United States
5. Sydney, Australia
```

Lists

- Requirements for numbering should be mentioned in prompt

```
prompt = "Generate an unordered list containing the names of the top 5 cities to visit."
print(get_response(prompt))
```

```
- Paris, France
- Tokyo, Japan
- Rome, Italy
- New York City, United States
- Sydney, Australia
```

Structured paragraphs

- Mention structure requirements in prompt

```
prompt = "Provide a structured paragraph with clear headings and subheadings
about the benefits of regular exercise on overall health and well-being."
print(get_response(prompt))
```

```
I. Introduction
Regular exercise is essential for maintaining good health and overall well-being. [...]

II. Physical Health Benefits
a. Weight Management: Regular exercise is [...]
b. Reduced Risk of Chronic Diseases: Exercise has been linked to [...]
c. Increased Strength and Flexibility: Regular exercise improves muscle strength and [...]

III. Mental Health Benefits
a. Reduced Stress and Anxiety: Exercise has a profound impact on [...]
b. Improved Sleep Quality: Regular exercise is associated with improved sleep patterns [...]
c. Boosted Brain Function: Studies show that exercise enhances cognitive function [...]
```

```
text = "Once upon a time in a quaint little village, there lived a curious young boy named David. David was [...]"
instructions = "You will be provided with a text delimited by triple backticks. Generate a suitable title for it. "

output_format = """Use the following format for the output:
         - Text: <text we want to title>
         - Title: <the generated title>"""

prompt = instructions + output_format + f"```{text}```"
print(get_response(prompt))
```

```
- Text: Once upon a time in a quaint little village, there lived a curious young boy [...]
- Title: The Extraordinary Adventure of David and the Mysterious Book
```

Conditional prompts

- Incorporate logic or conditions Conditional prompts follow
- an if-else style



```
text = "Le printemps est ma saison préférée. Quand les premières fleurs commencent
à éclore, et que les arbres se parent de feuilles vertes et tendres, je me sens
revivre [...] "
```

```
prompt = f"""You will be provided with a text delimited by triple backticks.
         If the text is written in English, suggest a suitable title for it.
         Otherwise, write 'I only understand English'.
         ```{text}```"""

print(get_response(prompt))
```

```
I only understand English
```
```

- Can incorporate multiple conditions

```
text = "In the heart of the forest, sunlight filters through the lush green canopy,
creating a tranquil atmosphere [...] "
```

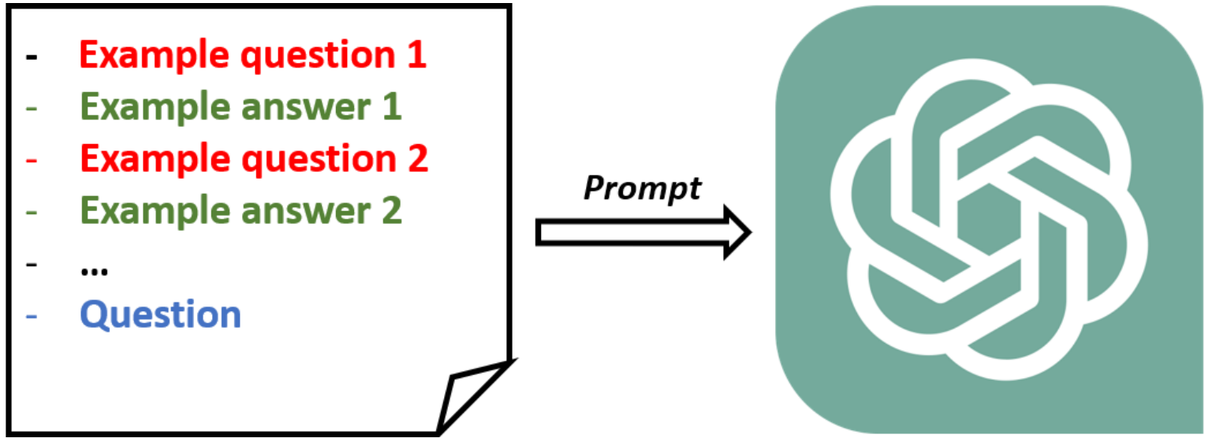
```
prompt = f"""You will be provided with a text delimited by triple backticks.
         If the text is written in English, check if it contains the keyword 'technology'.


         ```{text}```"""
print(get_response(prompt))
```
```

- Can incorporate multiple conditions

```
text = "In the heart of the forest, sunlight filters through the lush green canopy,
creating a tranquil atmosphere [...] "
```

```
prompt = f"""You will be provided with a text delimited by triple backticks.
         If the text is written in English, check if it contains the keyword 'technology'.
         If it does, suggest a suitable title for it, otherwise, write 'Keyword not found'.

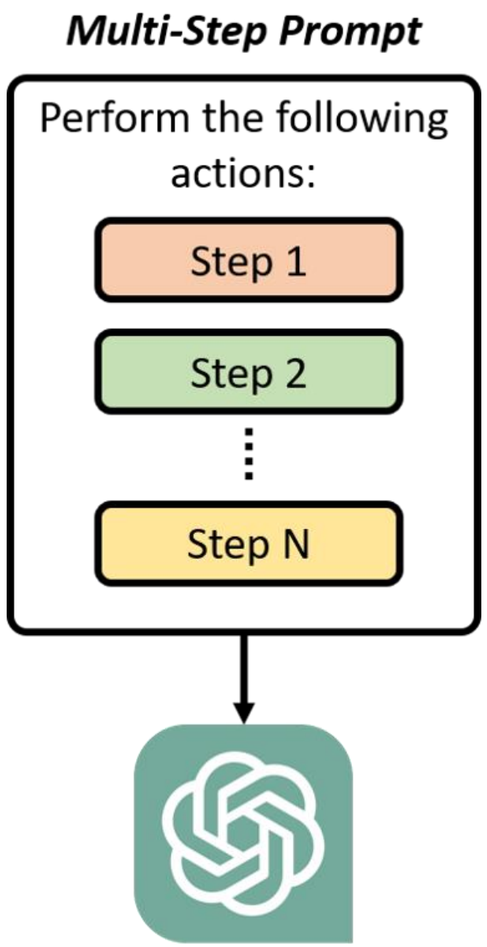
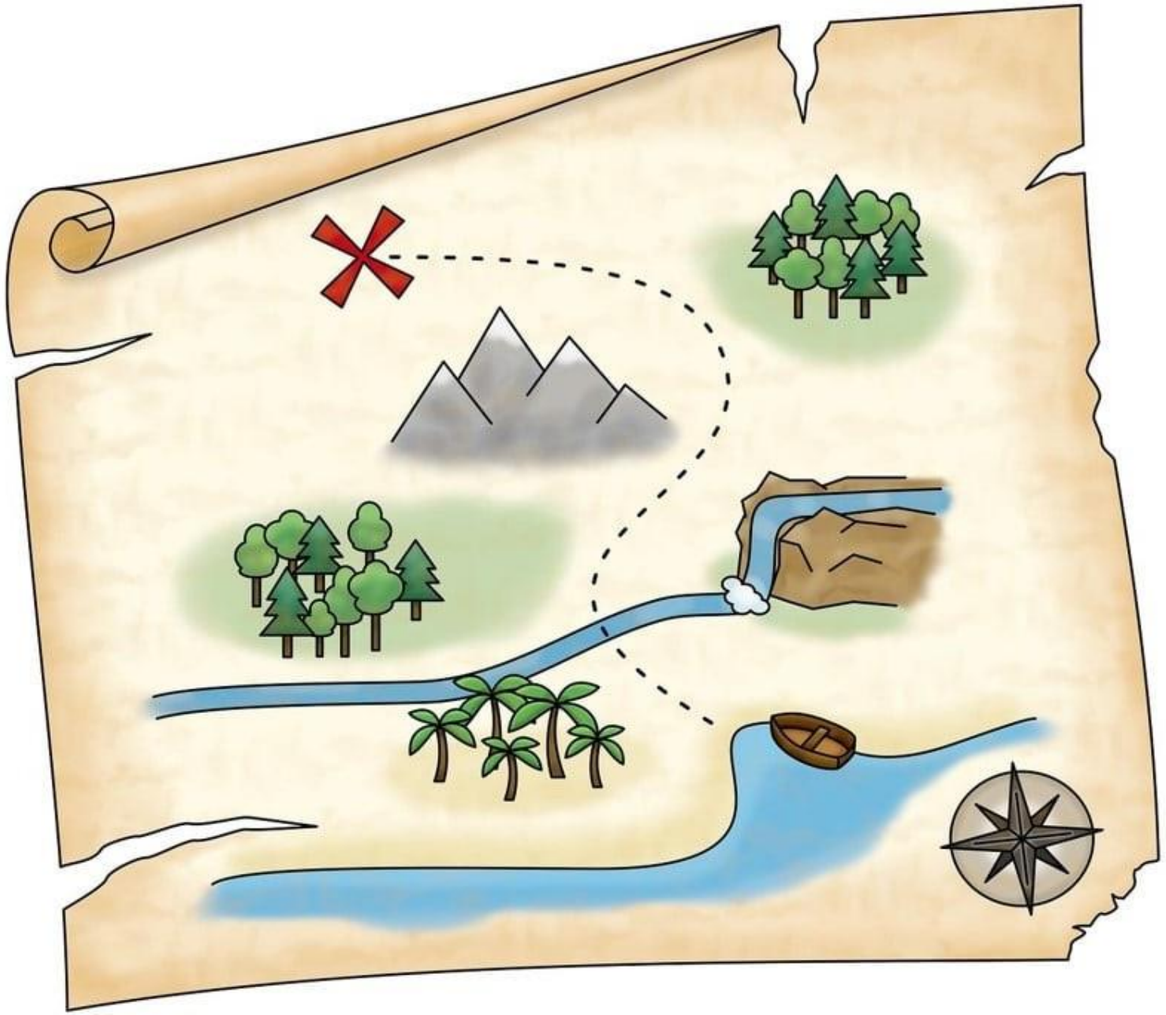
         ```{text}```"""
print(get_response(prompt))
```
```

- Can incorporate multiple conditions

```
text = "In the heart of the forest, sunlight filters through the lush green canopy,
creating a tranquil atmosphere [...] "
```

```
prompt = f"""You will be provided with a text delimited by triple backticks.
         If the text is written in English, check if it contains the keyword 'technology'.
         If it does, suggest a suitable title for it, otherwise, write 'Keyword not found'.
         If the text is not written in English, reply with 'I only understand English'.
         ```{text}```"""
print(get_response(prompt))
```

```
Keyword not found
```
```

TriveraTech
TECHNOLOGY TRAINING

Few-shot prompting

Experience is Everything

Few-shot prompting

- Model provided with examples (question-answer pairs)

- Example question 1
- Example answer 1
- Example question 2
- Example answer 2
- …
- Question

Few-shot prompting

- Model provided with examples (question-answer pairs)



Few-shot prompting

- Model provided with examples (question-answer pairs)

Few-shot prompting

- Model provided with examples (question-answer pairs)

- Number of examples:
  - Zero ->zero-shot prompting
  - One ->one-shot prompting
  - More than one ->few-shot prompting

Zero-shot prompting

- Providing a prompt without examples
- Model generates responses based on its knowledge
- Ideal for quick and uncomplicated tasks

```
prompt = "What is prompt engineering?"
print(get_response(prompt))
```

```
Prompt engineering refers to designing and refining prompts or instructions given
to a language model like ChatGPT to elicit desired responses or behaviors. It
involves formulating specific guidelines or hints to guide the model's output
towards a desired outcome.
```

One-shot prompting

- Provide the model a single example
- Useful for consistent formatting or style

```
prompt =  """
Q: Sum the numbers 3, 5, and 6. A: 3+5+6=14
Q: Sum the numbers 2, 4, and 7. A:
"""
print(get_response(prompt))
```

```
2+4+7=13
```

One-shot prompting

```
prompt = """
Q: Sum the numbers 3, 5, and 6. A: The sum of 3, 5, and 6 is 14
Q: Sum the numbers 2, 4, and 7. A:
"""
print(get_response(prompt))
```

```
The sum of 2, 4, and 7 is 13
```

Few-shot prompting

- Provide more than one example
- Powerful for contextual tasks

```
prompt = """
Text: Today the weather is fantastic -> Classification: positive
Text: The furniture is small -> Classification: neutral
Text: I don't like your attitude -> Classification: negative

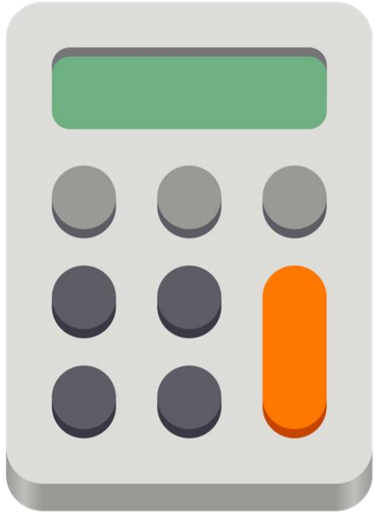
"""
```

Few-shot prompting

- Provide more than one example
- Powerful for contextual tasks

```
prompt = """
Text: Today the weather is fantastic -> Classification: positive
Text: The furniture is small -> Classification: neutral
Text: I don't like your attitude -> Classification: negative
Text: That shot selection was awful -> Classification:
"""
print(get_response(prompt))
```

```
negative
```

Few-shot prompting with a chat model

```
response = client.chat.completions.create(
  model = "gpt-3.5-turbo",
  messages = [{"role": "user",
              "content": "Today the weather is fantastic"},
              {"role": "assistant",
              "content": "positive"},
               {"role": "user",
              "content": "I don't like your attitude"},
               {"role": "assistant",
              "content": "negative"},
               {"role": "user",
              "content": "That shot selection was awful"}
              ],
  temperature = 0
)
print(response.choices[0].message.content)
```

```
negative
```

Considerations

- Choose number of shots according to **task complexity**
  - Fewer shots ->basic tasks
  - Diverse shots ->complex tasks

TriveraTech
TECHNOLOGY TRAINING

Multi-step prompting

Experience is Everything

Multi-step prompting

- Break down an end goal into series of steps
- Model goes through each step to give final output
- Multi-step prompts are used for:
  - Sequential tasks
  - Cognitive tasks



Multi-step prompts as treasure maps



Single-step prompt: writing a blog

```
prompt = "Compose a travel blog"
print(get_response(prompt))
```

```
Title: Exploring the Enchanting Landscapes of Iceland

Introduction: Welcome to my travel blog! Today, I am thrilled to share my unforgettable journey through
the mesmerizing landscapes of Iceland.

Day 1: Reykjavik - The Charming Capital [...]
Day 2: Golden Circle - Nature's Wonders [...]
Day 3: South Coast - A Journey of Ice and Fire [...]
Day 4: Glacier Lagoon - A Frozen Wonderland [...]
Day 5: Blue Lagoon - A Relaxing Finale [...]
```

Multi-step prompt: writing a blog post

```
prompt = """Compose a travel blog as follows:
Step 1: Introduce the destination.
Step 2: Share personal adventures during the trip.
Step 3: Summarize the journey.
"""
print(get_response(prompt))
```

Writing a travel blog post

```
Title: Exploring the Enchanting Streets of Barcelona


Step 1: Introduce the destination.
Welcome to Barcelona, a vibrant city nestled along the stunning Mediterranean coast of Spain [...]


Step 2: Share personal adventures during the trip.
Exploring the narrow, winding streets of the Gothic Quarter, I stumbled upon hidden gems at every turn.
[...]


Step 3: Summarize the journey.
As my journey through Barcelona came to an end, I couldn't help but feel grateful for the incredible
experiences and memories I had made [...]
```

Analyzing solution correctness

- Checking solution correctness requires multiple steps
- Example:
  - Python code for calculation functions



Analyzing solution correctness

Typical solution to check

```
calculator = """
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
def multiply(a, b):
    return a * b
def divide(a, b):
    return a / b
"""
```

Single-Step Prompt:

```
prompt = f"""Determine if the code delimited
by triple backticks is correct or not.
Answer by yes or no.
```{calculator}```"""

print(get_response(prompt))
```

```
Yes
```
```

Multi-step prompting to analyze solution correctness

Multi-Step Prompt:

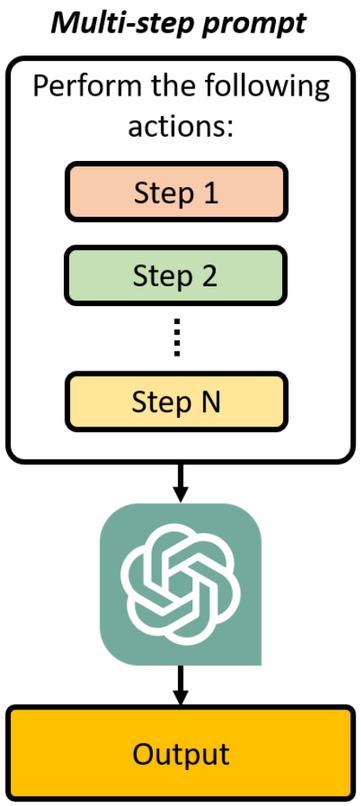
```
prompt = f"""Determine the correctness of the code delimited by triple backticks
as follows:
Step 1: Check the code correctness in each function.
Step 2: Verify if the divide function handles the case when dividing by 0.
Code: ```{calculator}```"""

print(get_response(prompt))
```

```
The code appears to be correct. It defines four functions: add, subtract, multiply, and
divide. Each function performs the corresponding mathematical operation and returns the
result. However, it does not handle the case when dividing by 0, which can result in a
ZeroDivisionError.
```

Multi-step versus few-shot prompt

Steps

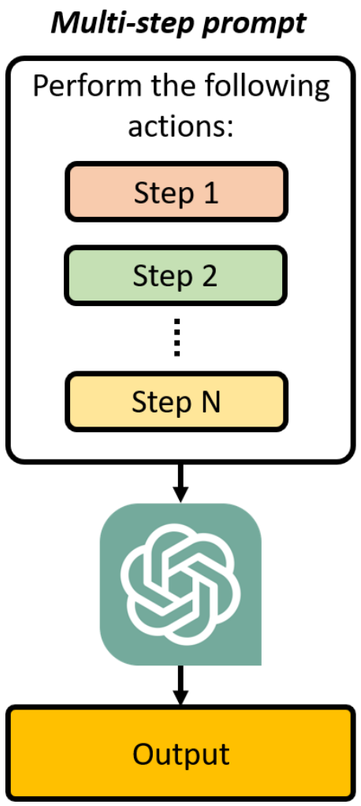
- Explicitly tell model what to do



Multi-step versus few-shot prompt

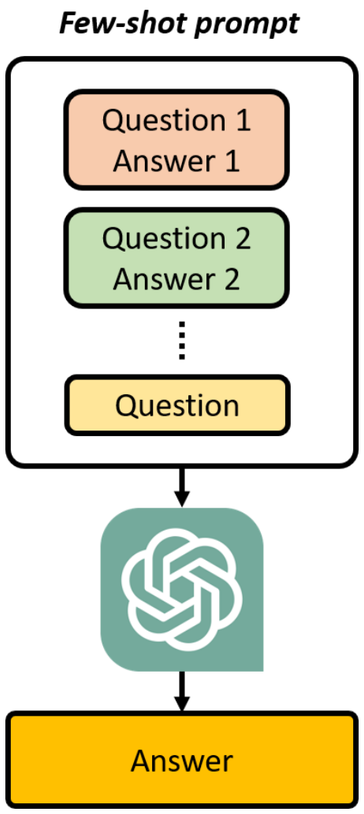
Steps

- Explicitly tell model what to do



Shots

- Questions and answers model learns from



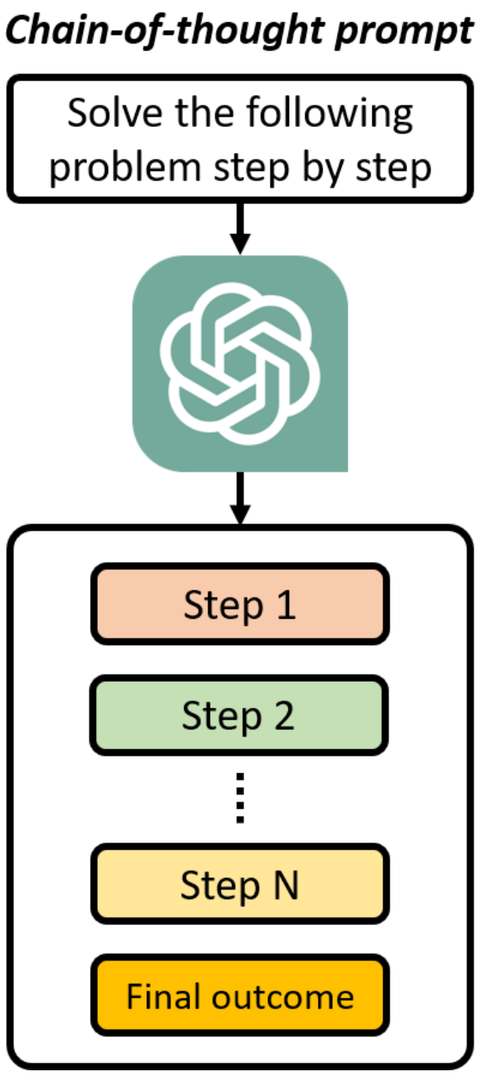
TriveraTech
TECHNOLOGY TRAINING

Chain-of-thought and self-consistency prompting

Experience is Everything

Chain-of-thought prompting

- Requires LLMs to provide reasoning steps (thoughts) before giving answer
- Used for complex reasoning tasks
- Help reduce model errors



Chain-of-thought prompting

STANDARD PROMPTING TO SOLVE A REASONING TASK

```
prompt = """Q: You start with 15 books in your collection. At the bookstore, you
purchase 8 new books. Then, you lend 3 to your friend and 2 to your cousin. Later,
you visit another bookstore and buy 5 more books. How many books do you have now?
A: The answer is"""

print(get_response(prompt))
```

```
25 books
```

Chain-of-thought prompting

CHAIN-OF-THOUGHT PROMPTING TO SOLVE A REASONING TASK

```
prompt = """Q: You start with 15 books in your collection. At the bookstore, you
purchase 8 new books. Then, you lend 3 to your friend and 2 to your cousin. Later,
you visit another bookstore and buy 5 more books. How many books do you have now?
A: Let's think step by step"""
print(get_response(prompt))
```

```
Step 1: Start with the number of books in your collection: 15 books
Step 2: Purchase 8 new books at the bookstore: 15 + 8 = 23 books
Step 3: Lend 3 books to your friend: 23 - 3 = 20 books
Step 4: Lend 2 books to your cousin: 20 - 2 = 18 books
Step 5: Visit another bookstore and buy 5 more books: 18 + 5 = 23 books
Therefore, you have 23 books now.
```

Chain-of-thought prompting with few-shots

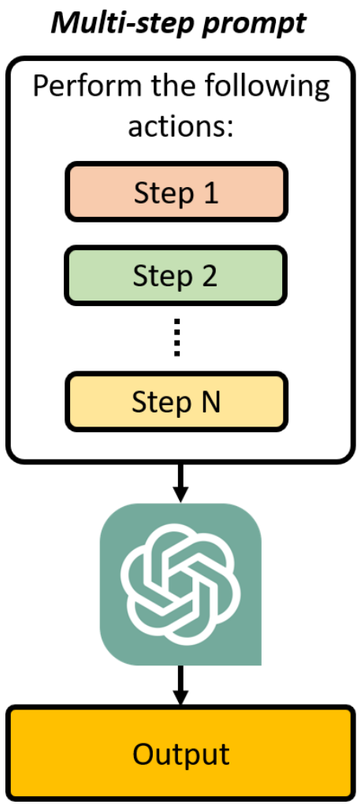
```
example = """
Q: The odd numbers in this group add up to an even number:  9, 10, 13, 4, 2.
A: Adding all the odd numbers (9, 13) gives 22. The answer is True.
"""
question = """
Q: The odd numbers in this group add up to an even number: 15, 13, 82, 7.
A:
"""
prompt = example + question
print(get_response(prompt))
```

```
Adding all the odd numbers (15, 13, 7) gives 35. The answer is False.
```

Chain-of-thought versus multi-step prompting

Multi-step prompts

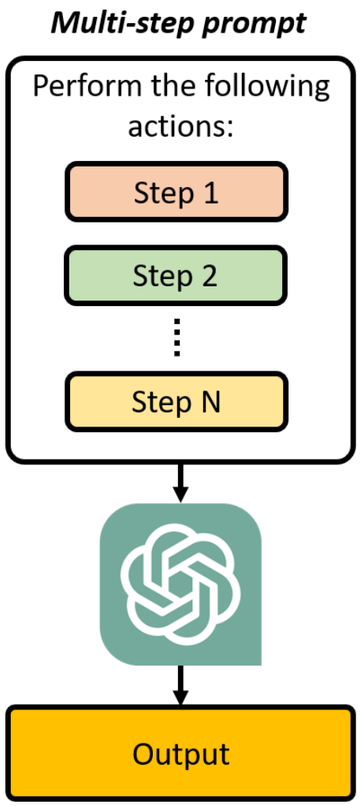
- Incorporate steps inside the prompt



Chain-of-thought versus multi-step prompting

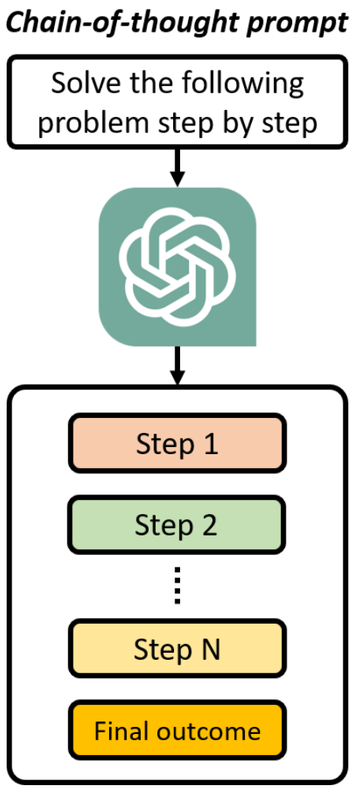
Multi-step prompts

- Incorporate steps inside the prompt



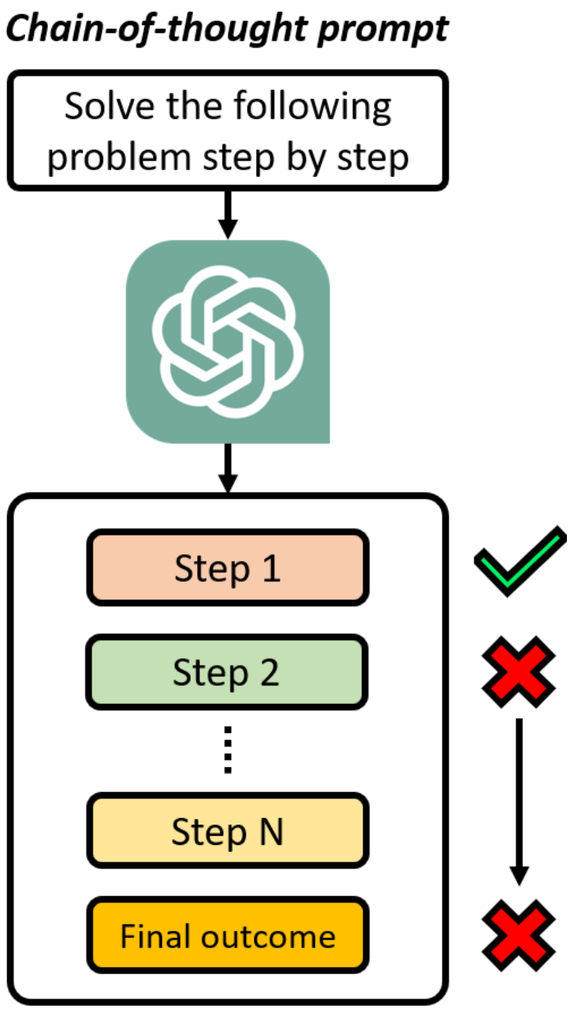
Chain-of-thought prompts

- Ask model to generate intermediate steps



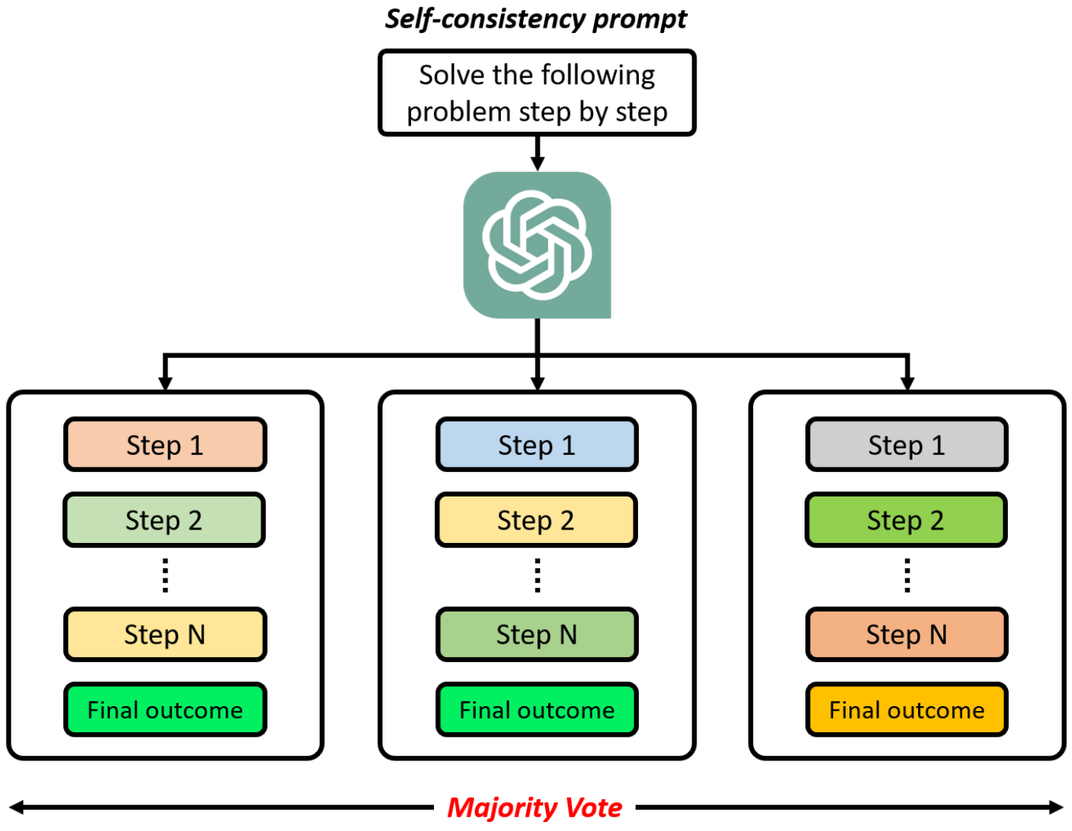
Chain-of-thought limitation

- One unsuccessful thought -->unsuccessful outcome
- **Self-consistency prompts** were introduced



Self-consistency prompting

- Generates multiple chain-of-thoughts by prompting the model several times
- Majority vote to obtain final output



Self-consistency prompting

Can be done by defining **multiple prompts** or a **prompt generating multiple** responses.

```
self_consistency_instruction = "Imagine three completely independent experts who
reason differently are answering this question. The final answer is obtained by
majority vote. The question is: "

problem_to_solve = "If there are 10 cars in the parking lot and 3 more cars arrive.
Half the original number of cars leave. Then, half of the current number of cars
arrive. How many cars are there in the parking?"

prompt = self_consistency_instruction + problem_to_solve

print(get_response(prompt))
```

Self-consistency prompt

```
Expert 1: Let's go step by step [...] Therefore, the total number of cars in the
parking lot is 8 + 4 = 12.

Expert 2: First, let's calculate [...] Therefore, the total number of cars in the
parking lot is now 5 + 2 = 7 cars.

Expert 3: Initially, there are 10 cars [...] Thus, the final answer is 8 + 4 = 12
cars in the parking lot.

Based on the majority vote, the final answer is that there are 12 cars in the
parking lot.
```

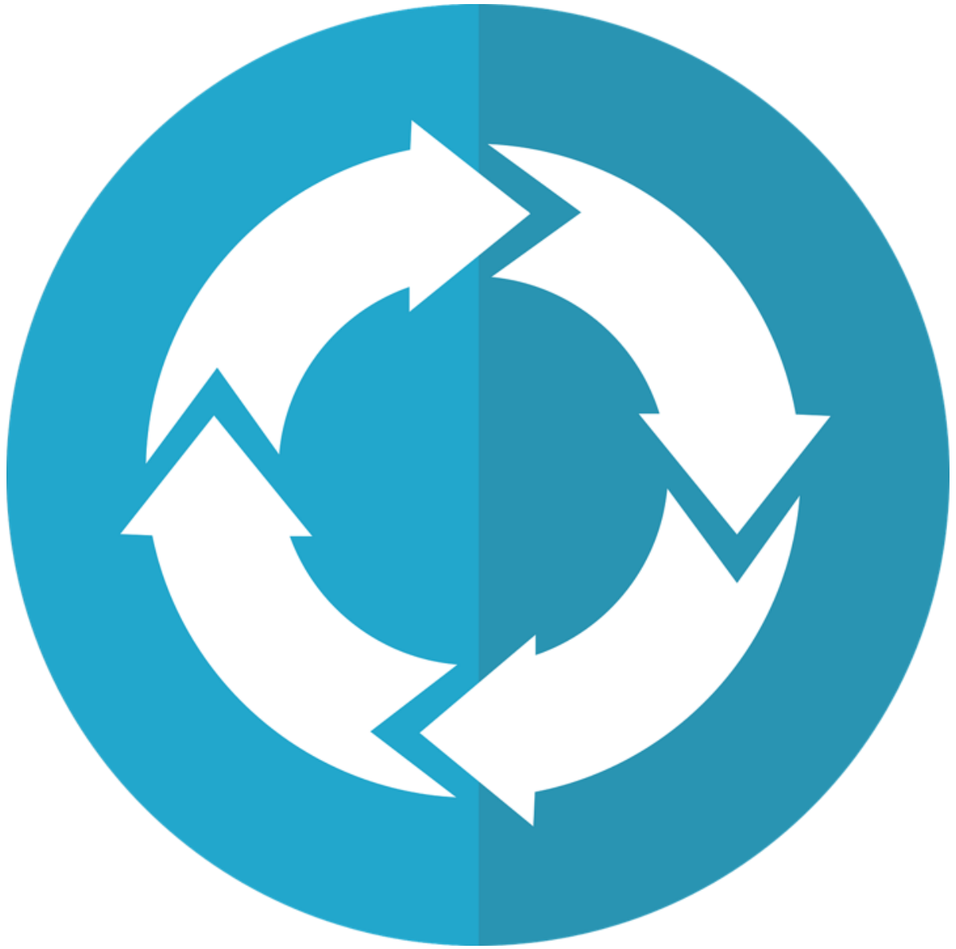
TriveraTech
TECHNOLOGY TRAINING

Iterative Prompt Engineering and Refinement

Experience is Everything

Iterative prompt engineering

- No prompt can be perfect at the beginning
- Prompt Engineering is an iterative process where we:
  - Build a prompt
  - Feed it to the model
  - Observe and analyze the output
  - Reiterate to make the prompt better



Refining prompts

Initial prompt

```
prompt = "Generate an Excel sheet containing five student names and their grades"

print(get_response(prompt))
```

```
I'm sorry, but as a text-based AI, I am unable to directly provide an Excel sheet. However, I can help you generate a sample representation of the data you requested.
```

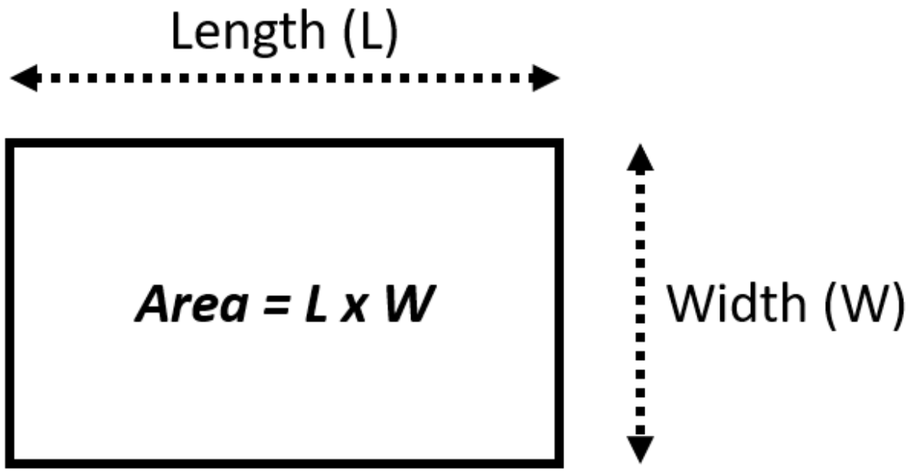
Refined prompt

```
prompt = "Generate a table that I can copy to Excel, containing five student names and their grades"
print(get_response(prompt))
```

```
| Student Name | Grade |
|--------------|-------|
| Student 1    |  90   |
| Student 2    |  85   |
| Student 3    |  95   |
| Student 4    |  88   |
| Student 5    |  92   |
```

Example: analyzing a python function

```
code = '''
def calculate_rectangle_area(length, width):
    area = length * width
    return area
'''
```



Example: initial prompt

```
prompt = f"""
  Analyze the code delimited by triple backticks with one sentence
  ```{code}```.
"""
print(get_response(prompt))
```

```
The code calculates the area of a rectangle based on its length and width.
```
```

Example: prompt refinement

We modify prompt to get programming language

```
prompt = f"""
  Analyze the code delimited by triple backticks and provide its programming
  language with one sentence
  ```{code}```.
"""
print(get_response(prompt))
```

```
The provided code is a function written in Python that calculates the area of a
rectangle based on its length and width.
```
```

Example: prompt refinement

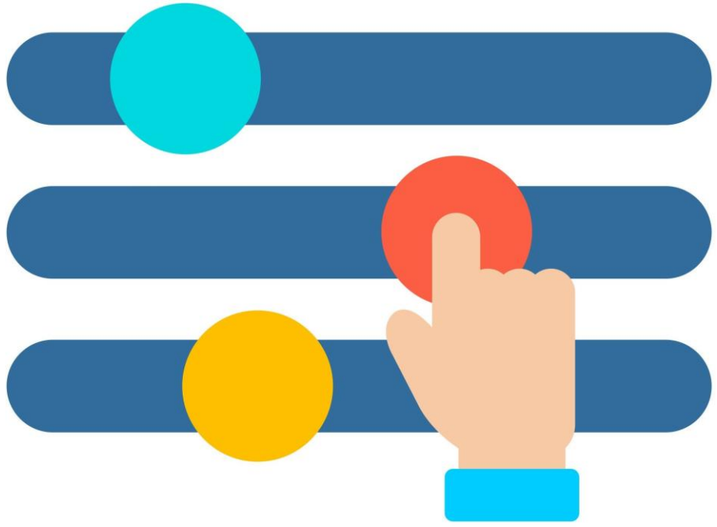
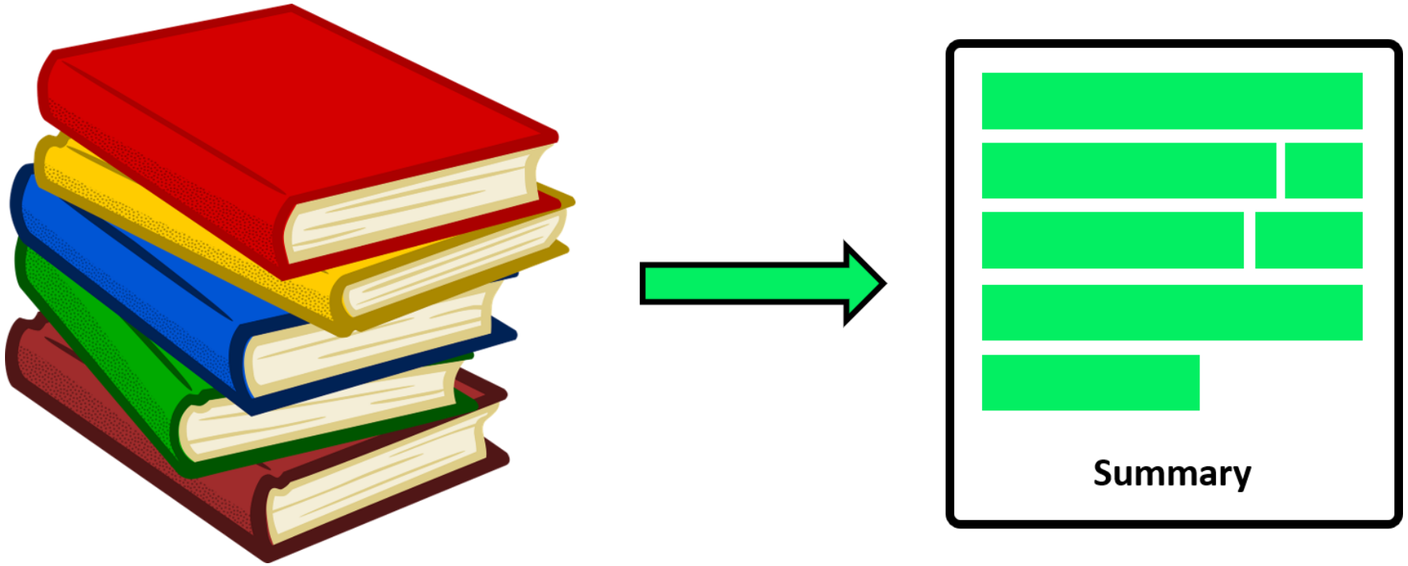
We modify prompt to get structured output

```
prompt = f"""
  For the function delimited by triple backticks, provide in a structured format
  the following:
  - description: one sentence short description
  - language: the programming language used
  - input: the inputs to the function
  - output: the output returned by the function
  ```{code}```.
"""
print(get_response(prompt))
```
```

Example: prompt refinement

```
description: This function calculates the area of a rectangle.

language: Python


input:

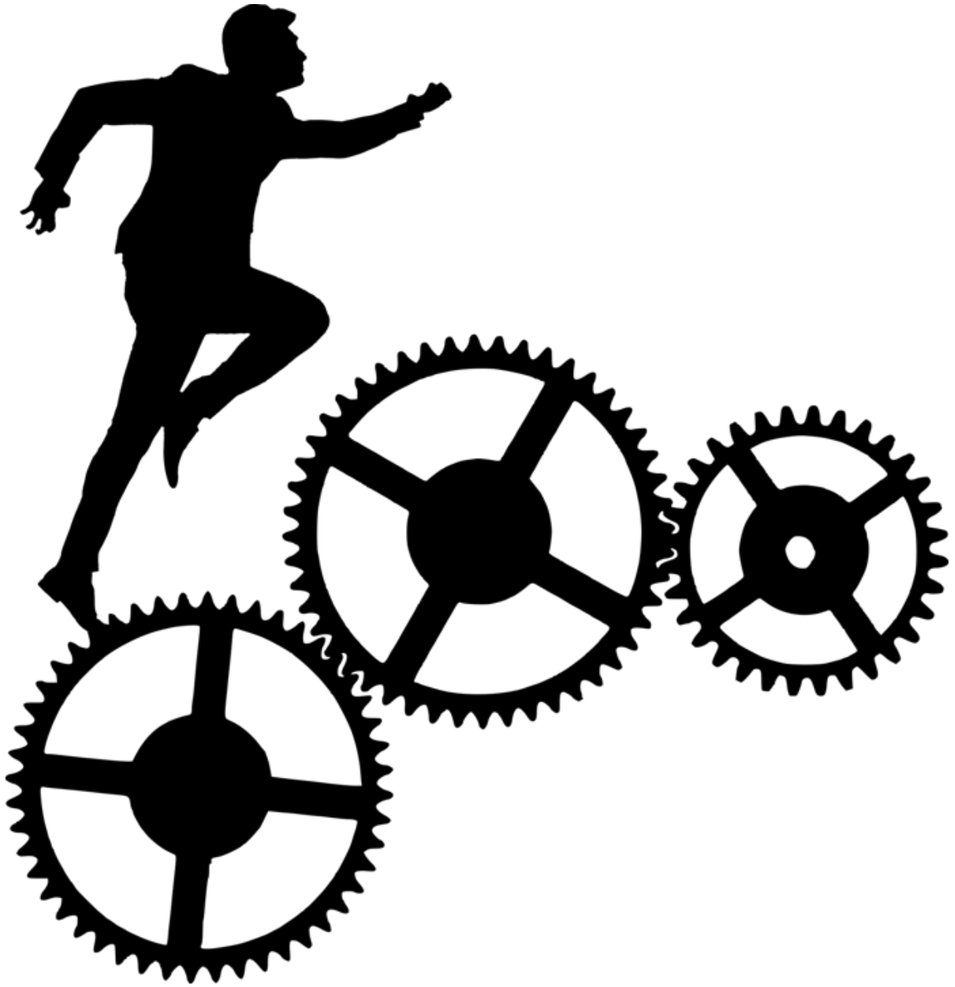
 - length: The length of the rectangle.
 - width: The width of the rectangle.


output:
 -area: The calculated area of the rectangle, which is the product of the length
 and width.
```

Few-shot prompt refinement

- Weather description classification

Initial prompt

```
prompt = """
Clear skies and a gentle breeze. -> Sunny
Heavy rain and thunderstorms expected. -> Rainy
Fresh snowfall with freezing temperatures. ->
"""
print(get_response(prompt))
```

```
Snowy
```

Few-shot prompt refinement

- Weather description classification

Initial prompt

```
prompt = """
Clear skies and a gentle breeze. -> Sunny
Heavy rain and thunderstorms expected. -> Rainy
The wind of change brought a refreshing breeze to the company's operations. ->
"""
print(get_response(prompt))
```

```
Windy
```

Few-shot prompt refinement

Refined prompt

```
prompt = """
Clear skies and a gentle breeze. -> Sunny
Heavy rain and thunderstorms expected. -> Rainy
The political climate in the country was stormy -> Unknown
The wind of change brought a refreshing breeze to the company's operations. ->
"""
print(get_response(prompt))
```

```
Unknown
```

Prompt refinement for various prompt types

- Few-shot prompts: refine examples
- Multi-step prompts: refine guiding steps
- Chain-of-thought and self-consistency prompts: refine problem description



TriveraTech
TECHNOLOGY TRAINING

Text summarization and expansion

Experience is Everything

Text summarization

- Condenses text into shorter format
- Streamlines business processes
  - Finance -> summarizes lengthy reports
  - Marketing -> transforms customer feedback into actionable insights
- LLMs can summarize texts with effective prompts



Ineffective prompt

- Only specifies text to summarize

```
text = "I recently purchased your XYZ Smart Watch and wanted to provide some feedback
based on my experience with the product. I must say that I'm impressed with the sleek design
and build quality of the watch. It feels comfortable on the wrist and looks great with any
outfit. The touchscreen is responsive and easy to navigate through the various features."

prompt = f"""Summarize the text delimited by triple backticks:
           ```{text}```"""
print(get_response(prompt))
```

```
The author purchased the XYZ Smart Watch and is impressed with its sleek design, build
quality, and comfortable fit on the wrist. They find the touchscreen responsive and
user-friendly for navigating the watch's features.
```
```

Prompt improvement

- Output limits
- Output structure
- Summarization focus



Effective prompt: output limits

- Specify number of sentences, words, characters

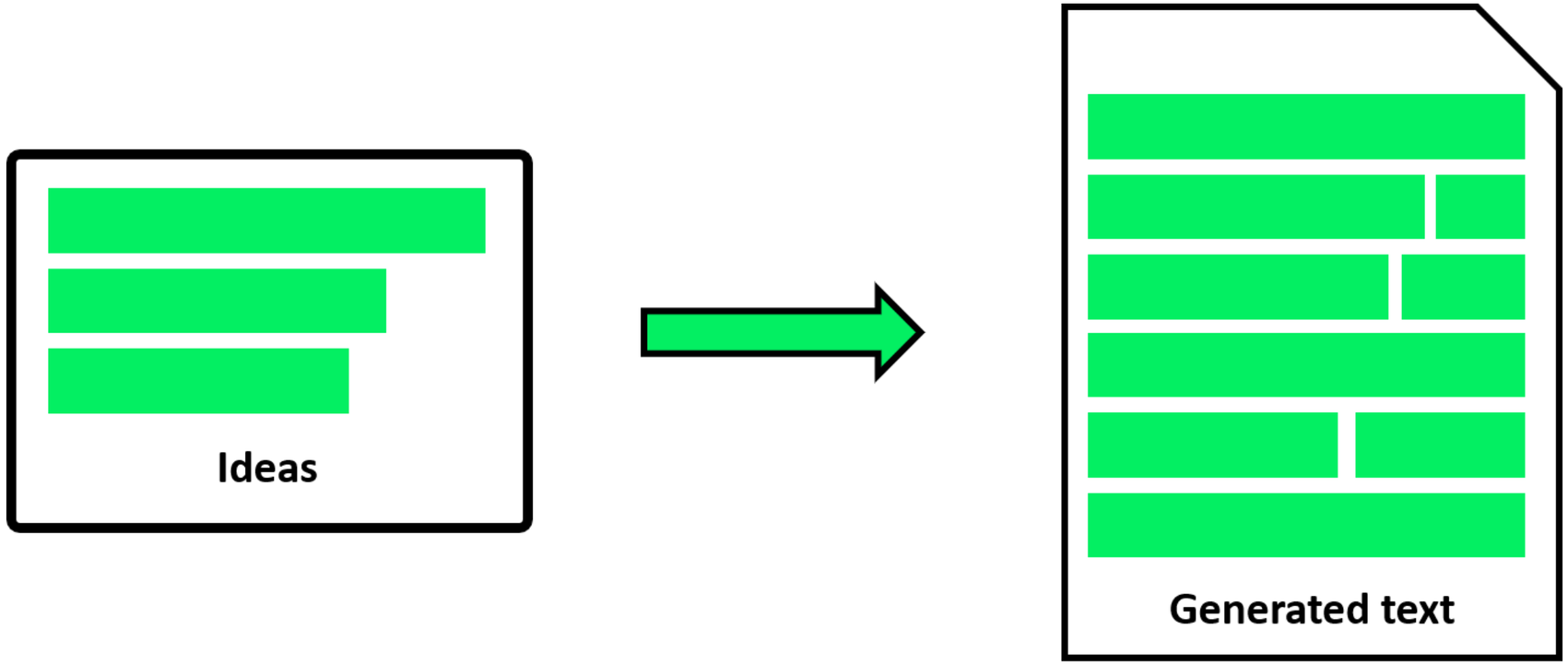
```
text = "I recently purchased your XYZ Smart Watch and wanted to provide some feedback
based on my experience with the product. I must say that I'm impressed with the sleek design
and build quality of the watch. It feels comfortable on the wrist and looks great with any
outfit. The touchscreen is responsive and easy to navigate through the various features."

prompt = f"""Summarize the text delimited by triple backticks in one sentence:
           ```{text}```"""
print(get_response(prompt))
```

```
The customer is impressed with the sleek design, build quality, comfort, and responsiveness
of the XYZ Smart Watch's touch screen.
```
```

Effective prompt: output structure

- Specify output structure

```
text = "I recently purchased your XYZ Smart Watch and wanted to provide some feedback
based on my experience with the product. I must say that I'm impressed with the sleek design
and build quality of the watch. It feels comfortable on the wrist and looks great with any
outfit. The touchscreen is responsive and easy to navigate through the various features."

prompt = f"""Summarize the text delimited by triple backticks, in at most three bullet points.
           ```{text}```"""
print(get_response(prompt))
```

```
- The XYZ Smart Watch has a sleek and impressive design with  excellent build quality.
- It feels comfortable on the wrist and complements any outfit.
- The touch screen is responsive and user-friendly for easy navigation through the features.
```
```

Effective prompt: summarization focus

- Ask model to focus on specific parts of text

```
text = "I recently purchased your XYZ Smart Watch and wanted to provide some feedback
based on my experience with the product. I must say that I'm impressed with the sleek design
and build quality of the watch. It feels comfortable on the wrist and looks great with any
outfit. The touchscreen is responsive and easy to navigate through the various features."

prompt = f"""Summarize the review delimited by triple backticks, in three sentences,
focusing on the key features and user experience:
            ```{text}```"""
print(get_response(prompt))
```
```

Effective prompt: summarization focus

```
The customer purchased the XYZ Smart Watch and was impressed with its sleek design
and build quality.
They found it comfortable to wear and versatile enough to match any outfit.
The touch screen was responsive and user-friendly, making it easy to navigate
through the watch's features.
```

Text expansion

- Generates text from ideas or bullet points
- Improves efficiency and productivity
- LLMs can expand text with well-crafted prompts



Text expansion prompts

- Ask model to expand delimited text
- Highlight aspects to focus on
- Provide output requirements (tone, length, structure, audience)

Expanding service description

```
service_description = """Service: Social XYZ
- Social Media Strategy Development
- Content Creation and Posting
- Audience Engagement and Community Building
- Increased Brand Visibility
- Enhanced Customer Engagement
- Data-Driven Marketing Decisions"""
```

```
prompt = f"""Expand the description for the Social XYZ service delimited by triple
backticks to provide an overview of its features and benefits, without bypassing
the limit of two sentences. Use a professional tone.
```{service_description}```"""
print(get_response(prompt))
```
```

Expanding service description

```
Social XYZ is a comprehensive social media service that offers strategic
development, content creation, and posting to help businesses effectively engage
with their target audience and build a strong online community.

With a focus on increasing brand visibility and enhancing customer engagement,
Social XYZ enables businesses to make data-driven marketing decisions for optimal
results.
```

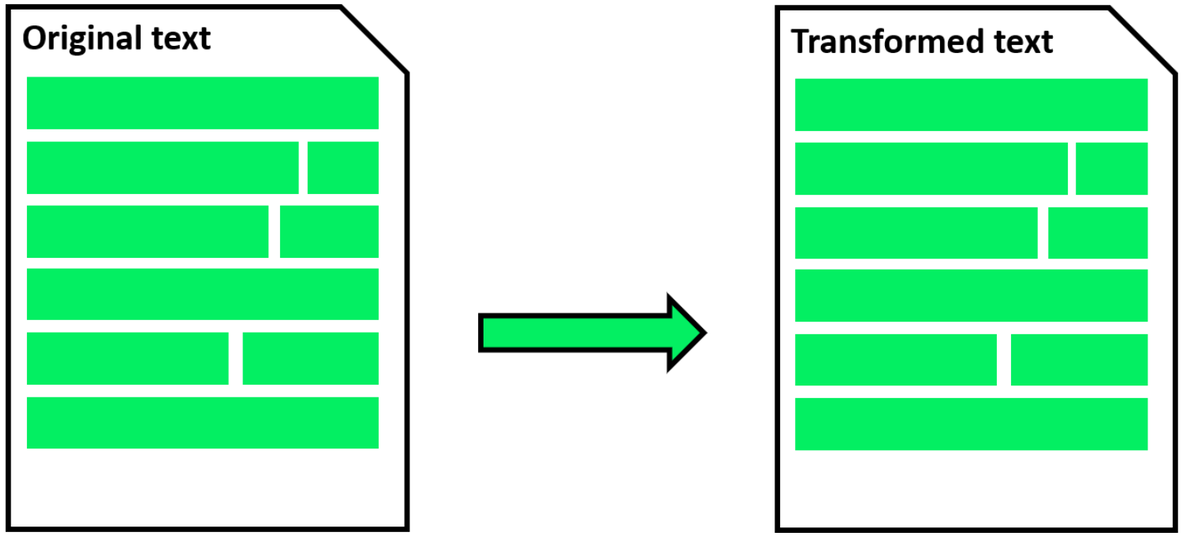
TriveraTech
TECHNOLOGY TRAINING

Text transformation

Experience is Everything

Text transformation

- Transforms given text to create a new text
- Has many applications
  - Language translation
  - Tone adjustments
  - Writing improvement



Language translation

- Specify input and output language in prompt

```
text = "XYZ Scooter is a cutting-edge electric scooter designed for urban adventurers. Its sleek design, long-lasting battery, and smart connectivity offer a seamless and eco-friendly way to navigate city streets."
prompt = f"""Translate the English text delimited by triple backticks to French:
            ```{text}```"""
print(get_response(prompt))
```

```
XYZ Scooter est une trottinette électrique de pointe conçue pour les aventuriers urbains. Avec son design élégant, sa batterie longue durée et sa connectivité intelligente, elle offre un moyen fluide et respectueux de l'environnement de se déplacer dans les rues de la ville.
```
```

If text language is unknown ...

```
text = "Das Produkt ist wirklich schön, und der Preis ist fair."
```

Ask the model to specify text language

```
prompt = f"""Tell me which language is the text delimited by triple backticks:
            ```{text}```"""
print(get_response(prompt))
```

```
The text delimited by triple backticks (```) is in the German language.
```
```

Multilingual translation

- Translate to multiple languages simultaneously

```
text = "The product combines top quality with a fair price."
prompt =  f"""Translate the English text delimited by triple backticks to French,
Spanish, and German:
```{text}```"""
```

```
French: Le produit allie une qualité supérieure à un prix équitable.
Spanish: El producto combina una calidad superior con un precio justo.
German: Das Produkt vereint Spitzenqualität mit einem fairen Preis.
```

- **Note:** Model translations need to be verified if customers are involved
```

Tone adjustment

- Re-write text in a different tone

```
text = "Hey there! Check out our awesome summer deals! They're super cool, and you
won't want to miss them. Grab 'em now!"

prompt = f"""Write the text delimited by triple backticks using a formal and
persuasive tone:
```{text}```"""

print(get_response(prompt))
```
```

Tone adjustment

```
Dear [Recipient's Name],

We cordially invite you to explore our exceptional summer offers! These exclusive
deals are truly extraordinary and should not be missed. Time is limited, so we
encourage you to seize these incredible opportunities promptly.

[...]

Don't miss out on this chance! Take swift action to secure these remarkable summer
deals before they expire. You won't find a more enticing selection of offers
elsewhere.

[...]
```

Tone adjustment: specify audience

```
text = "Our cutting-edge widget employs state-of-the-art microprocessors and
advanced algorithms, delivering unparalleled efficiency and performance for a wide
range of applications."

prompt = f"""Write the text delimited by triple backticks to be suitable for
a non-technical audience:
```{text}```"""

print(get_response(prompt))
```

```
Our advanced widget uses the latest technology to provide exceptional efficiency
and performance for a variety of uses.
```
```

Grammar and writing improvements

- Correct spelling, grammar, and punctuation mistakes without modifying other aspects

```
text = "Dear sir, I wanted too discus a potentiel opportunity for colabration.
Pls let me know wen you're avaliable."
```

```
prompt = f"""Proofread the text delimited by triple backticks while keeping the
original text structure intact:
            ```{text}```"""
print(get_response(prompt))
```

```
Dear sir, I wanted to discuss a potential opportunity for collaboration. Please
let me know when you are available.
```
```

Grammar and writing improvements

- Enhance clarity by modifying text structure

```
text = "Dear sir, I wanted too discus a potentiel opportunity for colabration.
Pls let me know wen you're avaliable."
```

```
prompt = f"""Proofread and restructure the text delimited by triple backticks for
enhanced readability, flow, and coherence:
            ```{text}```"""
print(get_response(prompt))
```

```
Dear sir,

I wanted to discuss with you a potential opportunity for collaboration.
Please let me know your availability.

Thank you.
```
```

Multiple transformations

- Ask for multiple transformations at once -> multi-step prompts

```
text = "omg, I cant believe how awesome this product is! Its like the best thing ever! You guys gotta try it out!"
prompt = f"""Transform the text delimited by triple backticks with the following two steps:
             Step 1 - Proofread it without changing its structure
             Step 2 - Change the tone to be professional
           ```      ```"""
response = get_response(prompt)
```

```
Step 1 - OMG, I can't believe how awesome this product is! It's like the best thing ever! You guys gotta try it out!
Step 2 - I am astounded by the exceptional qualities of this product. It represents a remarkable solution that surpasses expectations. I highly recommend trying it out, as its benefits are truly impressive.
```
```

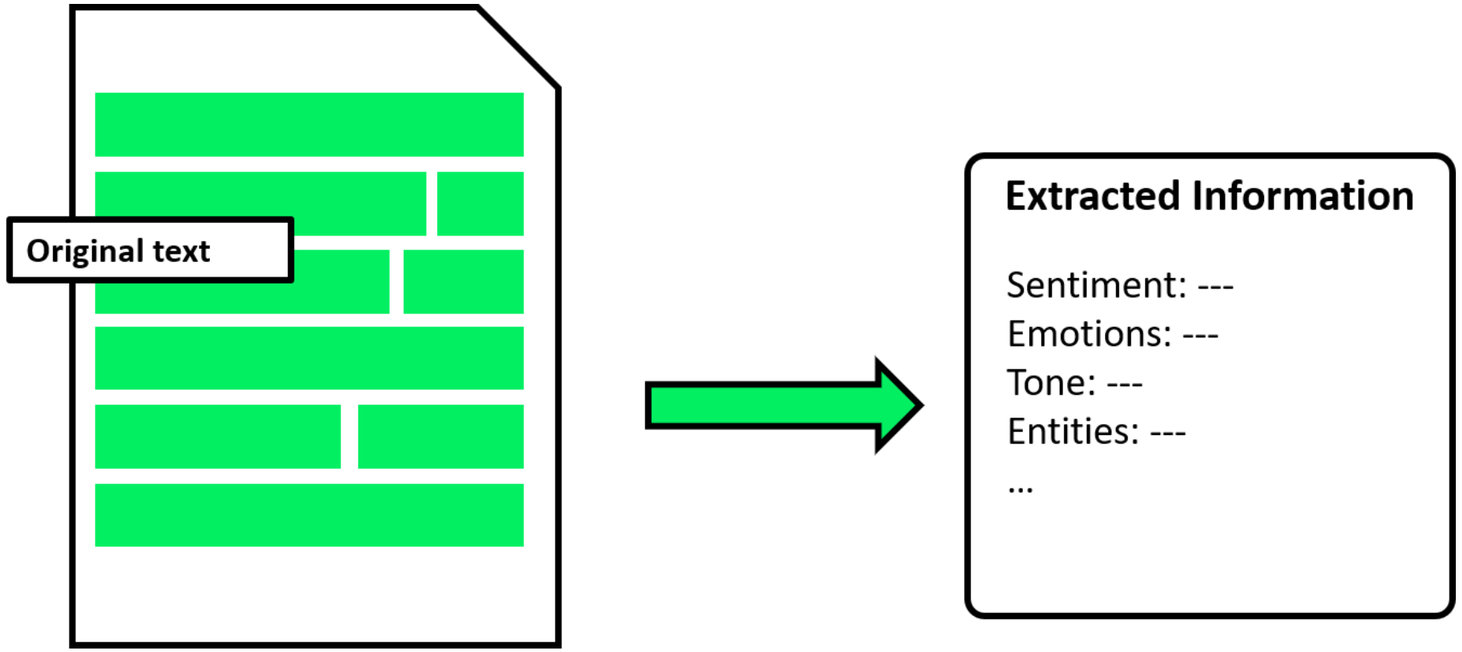
TriveraTech
TECHNOLOGY TRAINING

Text analysis

Experience is Everything

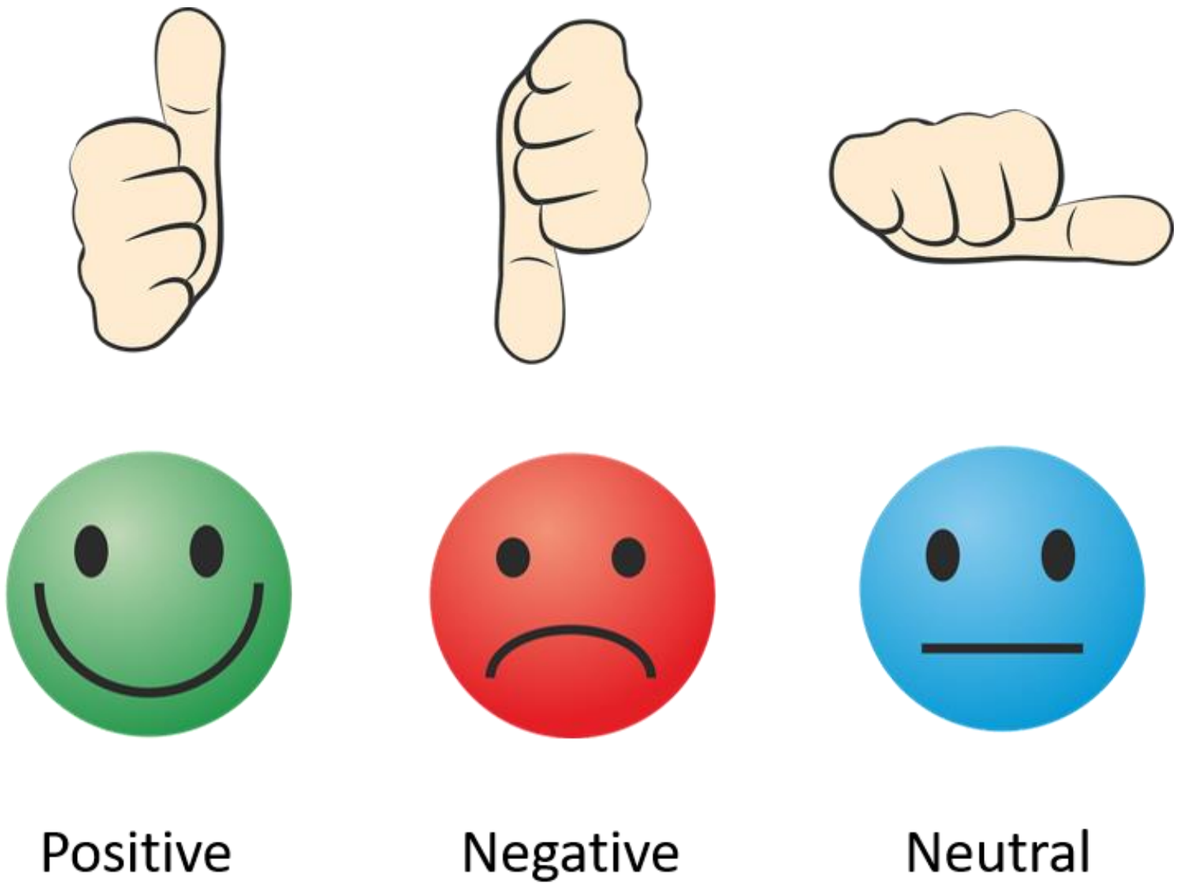
Text analysis

- Examining text to extract information
  - Text classification
  - Entity extraction
- Companies should seek legal advice when using customer data



Text classification

- Assign categories to text
- Example: sentiment analysis



Specified categories

- Specify classification categories when known
- Mention output requirements

```
text = "I bought your XYZ Smart Watch and wanted to share my positive experience.
Impressed with its sleek design, comfort, and touchscreen usability."

prompt = f"""Classify the sentiment of the text delimited by triple backticks as
positive, negative, or neutral. Give your answer as a single word:
            ```{text}```"""

print(get_response(prompt))
```

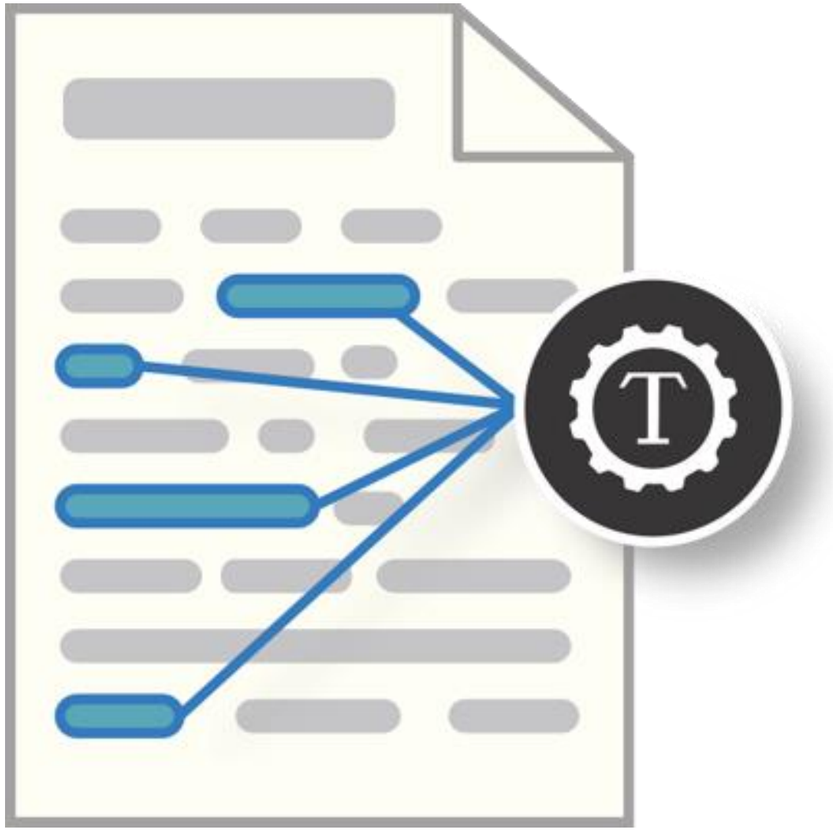
```
positive
```
```

Unspecified categories

- Model uses its knowledge when categories are not specified

```
text = "I bought your XYZ Smart Watch and wanted to share my positive experience.
Impressed with its sleek design, comfort, and touchscreen usability."


prompt = f"""Classify the sentiment of the text delimited by triple backticks.
            Give your answer as a single word.
          ```{text}```"""
print(get_response(response))
```

```
positive.
```

- For some open-ended problems this might not work well
```

Multiple classes

- Text can fit into multiple classes
- Define a maximum number of classes if they are not known

```
text = "I bought your XYZ Smart Watch and wanted to share my positive experience.
Impressed with its sleek design, comfort, and touchscreen usability."
prompt = f"""Identify emotions used in this text. Don't use more than 3 emotions.
             Format your answer as a list of words separated by commas:
           ```{text}```"""

print(get_response(prompt))
```

```
impressed, positive, comfortable
```
```

Entity extraction

- Extracting specific entities from text
- Examples: names, places, organizations, dates



Entity extraction: specify entities

- Specify entities to extract
- Mention output format

```
text = "The XYZ Mobile X200: a sleek 6.5-inch Super AMOLED smartphone with a 48MP
triple-camera, octa-core processor, 5000mAh battery, 5G connectivity, and Android
11 OS. Secure with fingerprint and facial recognition. 128GB storage, expandable up
to 512GB."
prompt = f"""Identify the following entities from the text delimited by triple backticks
             - Product Name




           ```{text}```"""
print(get_response(prompt))
```
```

Entity extraction: specify entities

- Specify entities to extract
- Mention output format

```
text = "The XYZ Mobile X200: a sleek 6.5-inch Super AMOLED smartphone with a 48MP
triple-camera, octa-core processor, 5000mAh battery, 5G connectivity, and Android
11 OS. Secure with fingerprint and facial recognition. 128GB storage, expandable up
to 512GB."
prompt = f"""Identify the following entities from the text delimited by triple backticks
            - Product Name
            - Display Size



          ```{text}```"""
print(get_response(prompt))
```
```

Entity extraction: specify entities

- Specify entities to extract
- Mention output format

```
text = "The XYZ Mobile X200: a sleek 6.5-inch Super AMOLED smartphone with a 48MP
triple-camera, octa-core processor, 5000mAh battery, 5G connectivity, and Android
11 OS. Secure with fingerprint and facial recognition. 128GB storage, expandable up
to 512GB."
prompt = f"""Identify the following entities from the text delimited by triple backticks
            - Product Name
            - Display Size
            - Camera Resolution

          ```{text}```"""
print(get_response(prompt))
```
```

Entity extraction: specify entities

- Specify entities to extract
- Mention output format

```
text = "The XYZ Mobile X200: a sleek 6.5-inch Super AMOLED smartphone with a 48MP
triple-camera, octa-core processor, 5000mAh battery, 5G connectivity, and Android
11 OS. Secure with fingerprint and facial recognition. 128GB storage, expandable up
to 512GB."
prompt = f"""Identify the following entities from the text delimited by triple backticks
            - Product Name
            - Display Size
            - Camera Resolution
            Format the answer as an unordered list.
          ```{text}```"""
print(get_response(prompt))
```
```

Entity extraction: specify entities

```
Product Name: XYZ Mobile X200

Display Size: 6.5-inch

Camera Resolution: 48MP triple-camera
```

Entity extraction with few-shot prompts

- For complex structures

```
ticket_1 = "Hello, I'm Emma Adams. I'd
like to ask about my reservation with
the code CAR123.
You can reach me at +123456 if needed."

ticket_2 = "This is Sarah Williams.
I would like to request some information
regarding my upcoming flight with
reservation code FLIGHT987. Thank you."
```

```
entities_1 = """
* Customer Details:
  - Name: Emma Adams
  - Phone: +123456
* Reservation Details:
- Reservation Code: CAR123"""
entities_2 = """
* Customer Details:
  - Name: Sarah Williams
* Reservation Details:
  - Reservation Code: FLIGHT987"""
```

Entity extraction with few-shot prompts

```
ticket_3 = "Hello, I'm David Brown (CUST123). I need assistance with my reservation under
the code HOTEL456. There are some questions and issues related to my upcoming stay that
require your attention."

prompt = f"""Text:                -> Entities:
             Text:                -> Entities:
             Text:                -> Entities: """
print(get_response(prompt))
```

```
* Customer Details:
  - Name: David Brown
  - Customer ID: CUST123
* Reservation Details:
  - Reservation Code: HOTEL456
```

TriveraTech
TECHNOLOGY TRAINING

Code generation and explanation

Experience is Everything

Code generation

- Creating source code to solve a given problem
- Essential in any domain using software solutions
- Understanding generated code is essential for effective use

Code generation prompts

- Problem description
- Programming language
- Format (script, function, class)

```
prompt = "Write a Python function that accepts a list of quarterly sales, and outputs the average sales per quarter."
print(get_response(prompt))
```

```
def calculate_average_sales(quarterly_sales):
    total_sales = sum(quarterly_sales)
    average_sales = total_sales / len(quarterly_sales)
    return average_sales
```

Input-output examples

- Give model input-output examples to generate a program that maps them

```
examples = """
Input: [150000, 180000, 200000, 170000] -> Output: 175000.0
Input: [10000, 25000, 30000, 15000] -> Output: 20000.0
Input: [50000, 75000, 60000, 45000] -> Output: 57500.0
"""
prompt = f"""You are provided with input-output examples delimited by triple
backticks for a python program that receives a list of quarterly sales data.
Write code for this program.
```{examples}```"""
print(get_response(prompt))
```
```

Input-output examples

```
Based on the provided input-output examples, it seems like you want to calculate the average of
quarterly sales data for each set of input values.

Here's a Python program that does that:

def calculate_average_quarterly_sales(sales_data):
    total_sales = sum(sales_data)
    num_quarters = len(sales_data)
    average_sales = total_sales / num_quarters
    return average_sales

print(calculate_average_quarterly_sales([150000, 180000, 200000, 170000]))   # Output: 175000.0
print(calculate_average_quarterly_sales([10000, 25000, 30000, 15000])) # Output: 20000.0
print(calculate_average_quarterly_sales([50000, 75000, 60000, 45000]))   # Output: 57500.0
```

Code modification

- Ask model to modify code according to requirements

```
script = """
quarterly_sales = [150, 180, 200, 170]
total_sales = sum(quarterly_sales)
print("Total sales: ", total_sales)
"""
prompt = f"""Modify the script delimited by triple backticks to a function that we
can call to compute the total sales given quarterly sales
```{script}```"""
print(get_response(prompt))
```
```

Code modification

```
def compute_total_sales(quarterly_sales):
    total_sales = sum(quarterly_sales)
    return total_sales


quarterly_sales = [150, 180, 200, 170]
total_sales = compute_total_sales(quarterly_sales)
print("Total sales: ", total_sales)
```

Multiple code modifications

```
script = """
quarterly_sales = [150, 180, 200, 170]
total_sales = sum(quarterly_sales)
print("Total sales: ", total_sales)
"""
prompt = f"""Modify the script delimited by triple backticks as follows:
            - Let user input parameters interactively



          ```{script}```"""
print(get_response(prompt))
```
```

Multiple code modifications

```
script = """
quarterly_sales = [150, 180, 200, 170]
total_sales = sum(quarterly_sales)
print("Total sales: ", total_sales)
"""
prompt = f"""Modify the script delimited by triple backticks as follows:
            - Let user input parameters interactively
            -Make sure to verify inputs are positive, otherwise, display a
            message for the user, and ask them to provide their input again.
          ```{script}```"""
print(get_response(prompt))
```
```

Multiple code modifications

```
quarterly_sales = []

for i in range(4):
    while True:
        sales = float(input("Enter quarterly sales for quarter {}: ".format(i+1)))
        if sales <= 0:
            print("Sales must be positive. Please try again.")
            continue
        quarterly_sales.append(sales)
        break

total_sales = sum(quarterly_sales)
print("Total sales: ", total_sales)
```

Code explanation

- Code can be difficult to interpret
- LLMs can be used to explain code

Code explanation requirements

```
code = """
def compute_average_sales_per_quarter(quarterly_sales):
    average_sales = sum(quarterly_sales) / len(quarterly_sales)
    return average_sales
"""
```

- Specify length of explanation

```
prompt = f"""Explain in one sentence what the code delimited by triple backticks does
             ```{code}```"""
print(get_response(prompt))
```

```
The code calculates the average sales per quarter by summing up the values in the
"quarterly_sales" list and dividing the sum by the number of elements in the list.
```
```

Detailed code explanation

```
code = """
def compute_average_sales_per_quarter(quarterly_sales):
    average_sales = sum(quarterly_sales) / len(quarterly_sales)
    return average_sales
"""
```

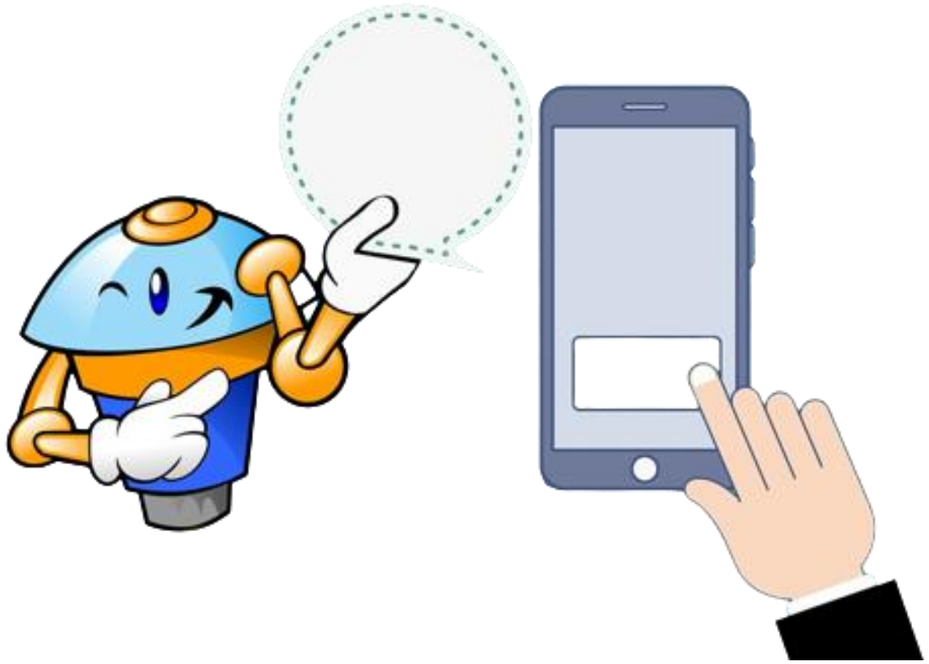
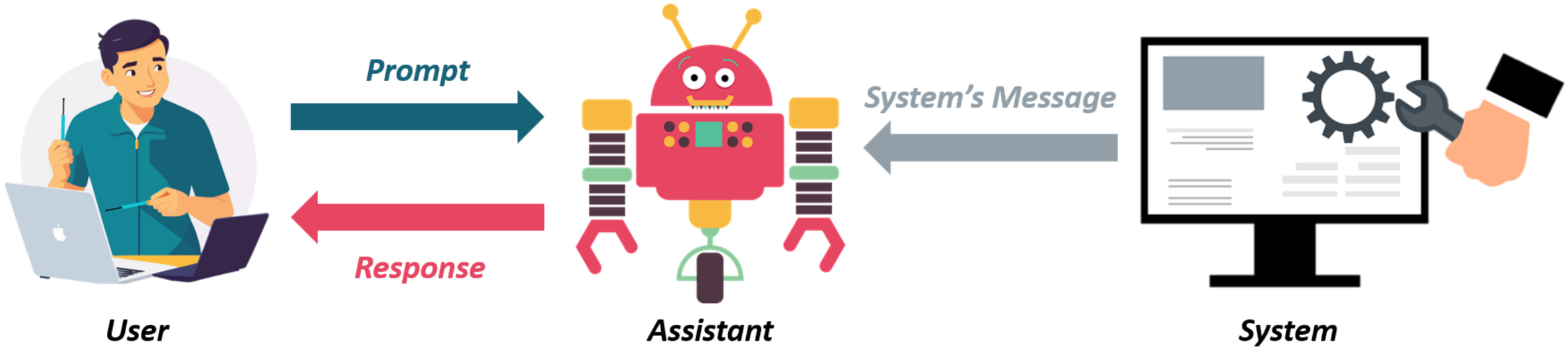
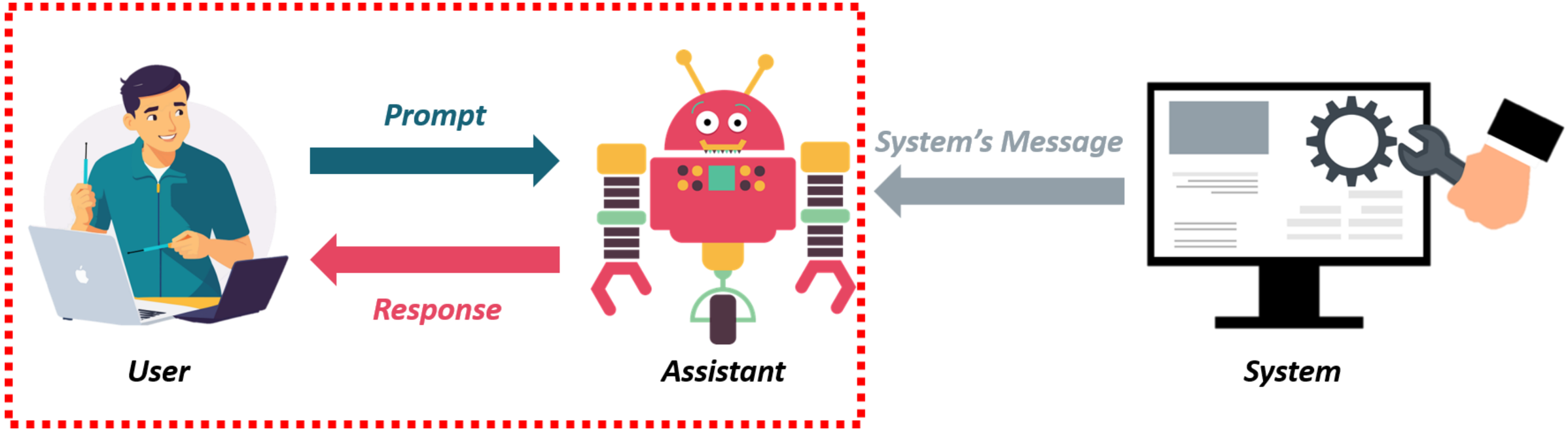
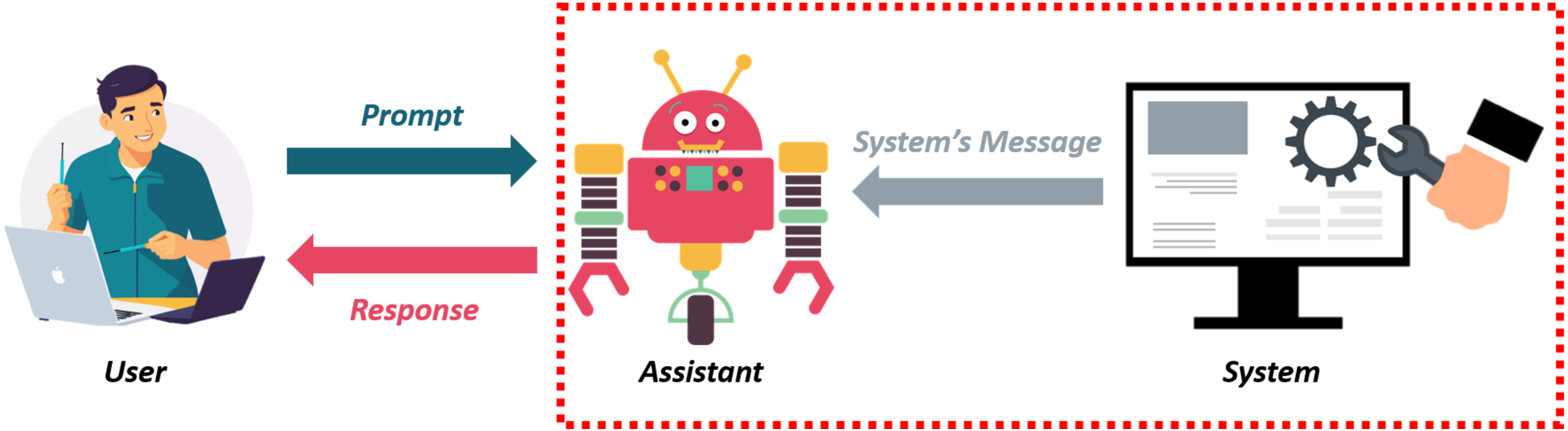
- Detailed explanation with chain-of-thought prompt

```
prompt = f"""Explain what the code delimited by triple backticks does.
Let's think step by step.
```{code}```"""
print(get_response(prompt))
```
```

Detailed code explanation

```
The code defines a function called `compute_average_sales_per_quarter` that takes in a list of `quarterly_sales` as a parameter.

Inside the function, it calculates the average sales by summing up all the values in the `quarterly_sales` list using the `sum()` function, and then dividing it by the length of the list using the `len()` function.

The result is stored in the variable `average_sales`.

Finally, the function returns the calculated average sales value.

This code snippet can be used to calculate the average sales per quarter given a list of quarterly sales data.
```

The need for prompt engineering for chatbots

- Difficult to predict user questions
- Challenge to guarantee effective responses
- Prompt engineering guides chatbot behavior



Chatbot prompt engineering with OpenAI API



- Each message has a designated role

Chatbot prompt engineering with OpenAI API



- Each message has a designated role
- Focus has been on user prompts

Chatbot prompt engineering with OpenAI API



- Each message has a designated role
- Focus has been on user prompts
- System prompts guide chatbot's behavior

Chat completions endpoint for chatbot development

- Sends series of messages to model as a list

```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[{"role": "system",
             "content": "You are an expert data scientist that explains complex concepts in simple terms"},
            {"role": "user",
             "content": "What is prompt engineering?"}]
)
print(response.choices[0].message.content)
```

```
Imagine you're giving instructions to a computer program, like teaching a robot to make a sandwich.
Prompt engineering is all about crafting those instructions, or "prompts," in a way that helps the
computer understand and perform the task better.
```

Changing get_response() for chatbot

- Sending one prompt

```
def get_response(prompt):
    messages = [
      {"role": "user", "content": prompt}
    ]
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        temperature=0,
    )
    return response.choices[0].message.content

prompt = "<PROMPT>"
print(get_response(prompt))
```

- Sending two prompts

```
def get_response(system_prompt, user_prompt):
    messages = [
      {"role": "system", "content": system_prompt},
      {"role": "user", "content": user_prompt}]
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        temperature=0,
    )
    return response.choices[0].message.content

system_prompt = "<SYSTEM_PROMPT>"
user_prompt = "<USER_PROMPT>"
print(get_response(system_prompt, user_prompt))
```

System message: define purpose

```
system_prompt = "You are a chatbot that answers financial questions."

user_prompt = "Who are you?"

print(get_response(system_prompt, user_prompt))
```

```
I'm a financial chatbot that answers financial questions. How can I help you?
```

- Allow chatbot to offer domain-accurate assistance
- Not defining purpose might lead to contextually irrelevant answers

System message: response guidelines

- Specify audience, tone, length, structure

```
system_prompt = """You are a chatbot that answers financial questions.
Your answers should be precise, formal and objective"""

user_prompt = "What do you think about cryptocurrencies?"

print(get_response(system_prompt, user_prompt))
```

```
Cryptocurrencies are digital or virtual currencies that use cryptography for security and operate on
decentralized networks based on blockchain technology.
[...]

Advantages of cryptocurrencies include:
- Decentralization: [...]
- Security:[...]
- Global Accessibility: [...]

However, there are also notable concerns:
- Volatility: [...]
- Regulatory Uncertainty: [...]
- Lack of Consumer Protection: [...]

In summary, cryptocurrencies have the potential to offer various benefits, but their adoption and impact
on the financial landscape are still evolving [...]
```

System message: behavior guidance

- Conditional prompts to respond to questions

```
system_prompt = """You are a chatbot that answers financial questions.
Your answers should be precise, formal and objective.



"""
```

System message: behavior guidance

- Conditional prompts to respond to questions

```
system_prompt = """You are a chatbot that answers financial questions.
Your answers should be precise, formal and objective.
If the question you receive is within the financial field, answer it to the best of your knowledge.

"""
```

System message: behavior guidance

- Conditional prompts to respond to questions

```
system_prompt = """You are a chatbot that answers financial questions.
Your answers should be precise, formal and objective.
If the question you receive is within the financial field, answer it to the best of your knowledge.
Otherwise, answer with 'Sorry, I only know about finance.'
"""
```

```
user_prompt = "How's the weather today?"


print(get_response(system_prompt, user_prompt))
```

```
Sorry, I only know about finance.
```

Role-playing prompts

- Tell chatbot to play a specific role
- Chatbot ->actor in a play
- Chatbot adjusts to match role
- Tailored language and content to fit the persona
- More effective interactions

Role-playing example

Question: Could you give me technical specifications of Product X that your company offers?

Customer Support

Role-playing example

Question: Could you give me technical specifications of Product X that your company offers?

Customer Support

Product Manager

Role-playing example

Question: Could you give me technical specifications of Product X that your company offers?

Customer Support

Product Manager

Sales Engineer

Customer support agent

- Provides guidance
- Directs customers to website
- Offers assistance

- **Product manager**
- Highlights strategic benefits Focuses on product alignment

Sales engineer

- Focus on technical specifics
  - Processor
  - Features
  - Security

Role-playing prompts

- Tell model to **act as** as **specific role**

```
system_prompt = "Act as an expert financial analyst."
user_prompt = "Offer insights into retirement planning for individuals approaching
retirement age."

print(get_response(system_response, user_response))
```

Expert financial analyst

```
Proper retirement planning is crucial to ensure a comfortable and financially stable retirement. Here are some key
considerations:

- Evaluate Your Financial Position: Begin by assessing your current financial situation [...]
- Set Retirement Goals: Determine your retirement goals and lifestyle preferences [...]
- Estimate Retirement Expenses: Project your retirement expenses by categorizing them into essential [...]
[...]

Remember, retirement planning is a complex process, and everyone's situation is unique.
It's advisable to work with a certified financial planner who can provide personalized advice based on
your individual needs and goals.
```

More effective role-playing

- Specific requirements within the role Incorporate traits like personality and expertise

```
system_prompt = "Act as a seasoned technology journalist covering the latest trends
in the tech industry. You're known for your thorough research and insightful analysis."

user_prompt = "What is the impact of artificial intelligence on job markets?"

print(get_response(system_response, user_response))
```

```
Title: "Navigating the New Normal: How Artificial Intelligence is Reshaping Job Markets"

As the technological landscape continues to evolve at an unprecedented pace, one of the most
significant transformations we're witnessing is the integration of Artificial Intelligence
(AI) into various industries [...].

While AI's potential to streamline processes [...]. Let's explore the multifaceted impact
of AI on employment...

1. Automation and Job Displacement: [...]
2. Augmentation and Enhanced Creativity: [...]
[...]

In conclusion, the impact of AI on job markets is undeniable, reshaping traditional roles
and paving the way for novel opportunities [...]
```

Role-playing with requirements

- We can specify response guidelines and behavior guidance in role-playing prompts

```
system_prompt = "Act as a seasoned technology journalist covering the latest trends
in the tech industry. You're known for your in-depth research and insightful analysis."
```

Role-playing with requirements

- We can specify response guidelines and behavior guidance in role-playing prompts.

```
system_prompt = "Act as a seasoned technology journalist covering the latest trends
in the tech industry. You're known for your in-depth research and insightful analysis.
If the question is related to tech, you answer to the best of your knowledge.
Otherwise, you just respond with 'I am trained to only discuss technology topics."

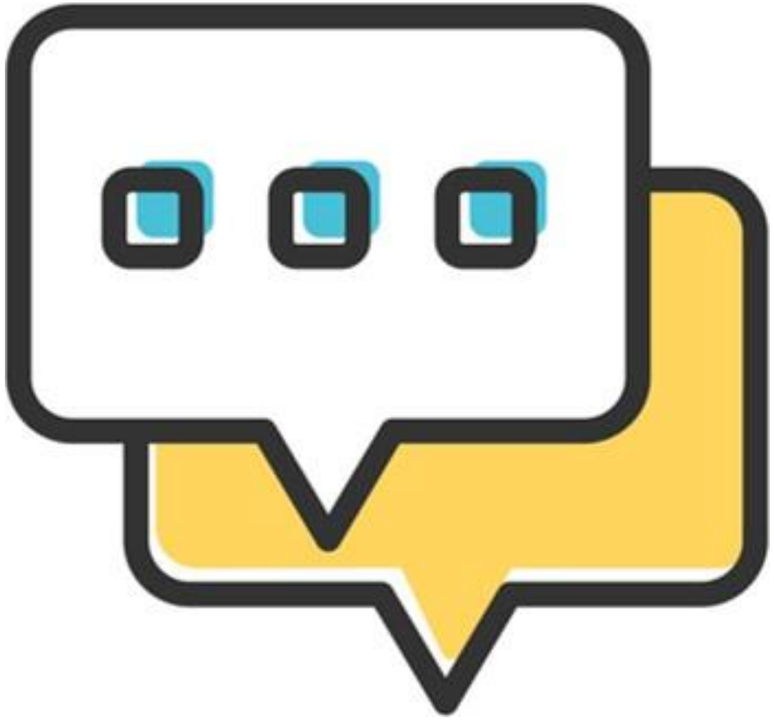
user_prompt = "Which American literature books do you recommend?"

print(get_response(system_prompt, user_prompt))
```

```
I am trained to only discuss technology topics.
```

The need for external context

- Pre-trained language models recognize information they are trained on
- Need to provide more context
- More accuracy and effectiveness

Lack of information in LLMs

- Knowledge cut-off

```
system_prompt = "Act as a financial expert that knows about the latest trends."

user_prompt = "What are the top financial trends in 2023?"

print(get_response(system_prompt, user_prompt))
```

```
I apologize for any inconvenience, but as of my last knowledge update in
September 2021, I don't have information about financial trends in 2023.
```

Lack of information in LLMs

- Requested non-public information

```
system_prompt = "Act as a study buddy that helps me with my studies to succeed
in exams."

user_prompt = "What is the name of my favorite instructor?"

print(get_response(system_prompt, user_prompt))
```

```
I don't have personal information about you, including the name of your
favorite instructor.
```

How to give extra information?

- Sample previous conversations System's prompt



- Guide model to answer specific questions

```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[{"role": "system",
             "content": "You are a customer service chatbot that responds to user queries in a gentle way"},
            {"role": "user",
             "content": "What services do you offer?"},
           {"role": "assistant",
             "content": "We provide services for web application development, mobile app development, and
            custom software solutions."},
           {"role": "user",
             "content": "How many services do you have?"}])
print(response.choices[0].message.content)
```

```
We have 3 services including web application development, mobile app development, and custom software solutions.
```

- Disadvantage: might need a lot of samples

System prompt

- Includes required context for chatbot answers

```
services = "ABC Tech Solutions, a leading IT company, offers a range of services:
application development, mobile app development, and custom software solutions."
system_prompt = f"""You are a customer service chatbot that responds to user
queries in a gentle way. Some information about our services are delimited by
triple backticks.
```{services}```"""
user_prompt = "How many services do you offer?"
print(get_response(system_prompt, user_prompt))
```

```
We have 3 services including web application development, mobile app development,
and custom software solutions.
```
```

Final note

- Previous methods work well for small contexts
- Larger contexts require more sophisticated techniques

Review 1

Key principles of prompt engineering

- Delimited prompts
- Conditional prompts
- Structured outputs
-

Review 2

Key principles of prompt engineering

- Delimited prompts
- Structured outputs
- Conditional prompts
-

Few-shot prompting

- Multi-step prompting
- Chain-of-thought and self-consistency
- Iterative prompt engineering
-

Review 3

- Key principles of prompt engineering
- Delimited prompts
- Structured outputs
- Conditional prompts

- Few-shot prompting
- Multi-step prompting
- Chain-of-thought and self-consistency
- Iterative prompt engineering

- Text summarization and expansion
- Text transformation
- Text analysis
- Code generation and explanation

Review 4

- Key principles of prompt engineering
- Delimited prompts
- Structured outputs
- Conditional prompts

- Few-shot prompting
- Multi-step prompting
- Chain-of-thought and self-consistency
- Iterative prompt engineering

- Text summarization and expansion
- Text transformation
- Text analysis
- Code generation and explanation

- System prompt engineering for chatbots
- Role-playing prompts
- External context

Final note

- LLMs have limitations
- Always make sure to review the answers

Embeddings

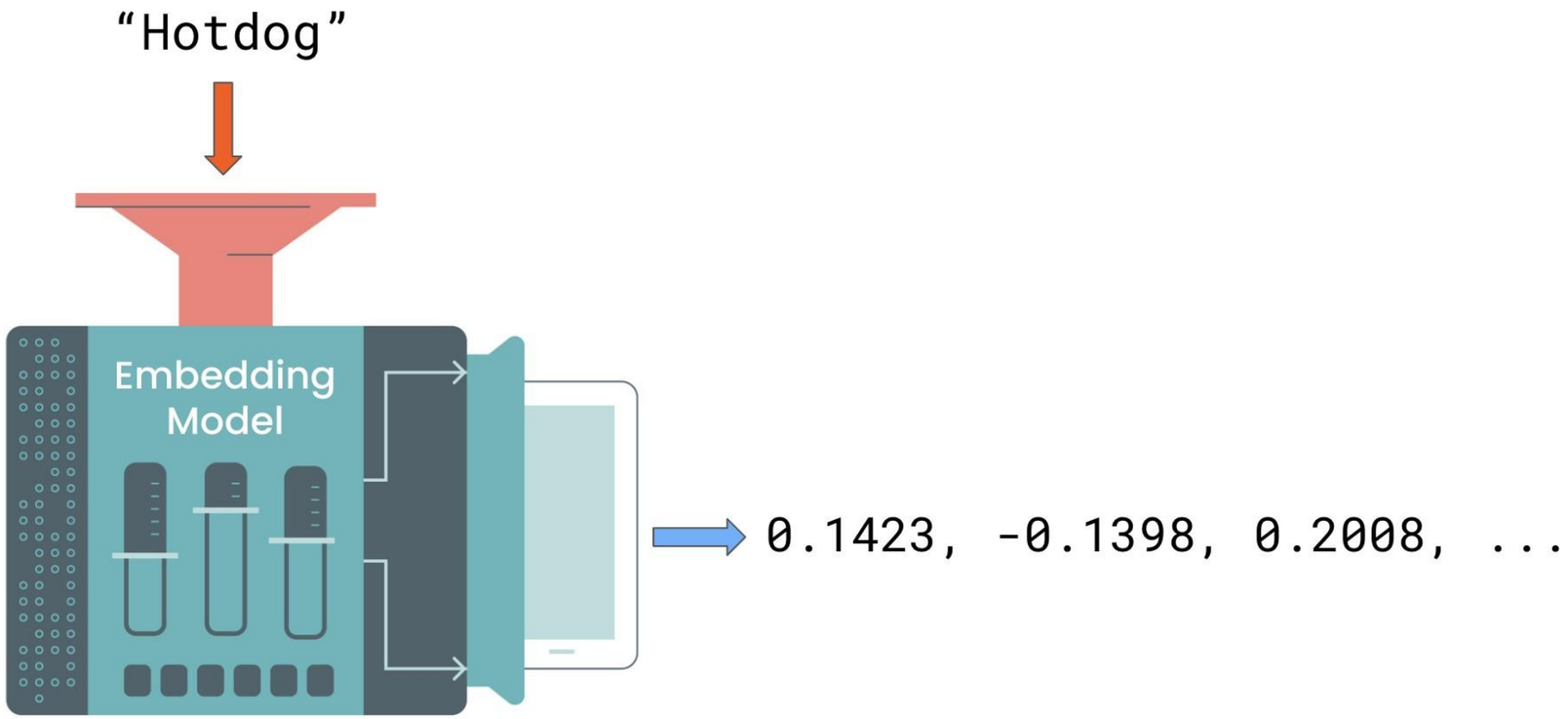
TriveraTech
TECHNOLOGY TRAINING

The wonderful world of embeddings!

Experience is Everything

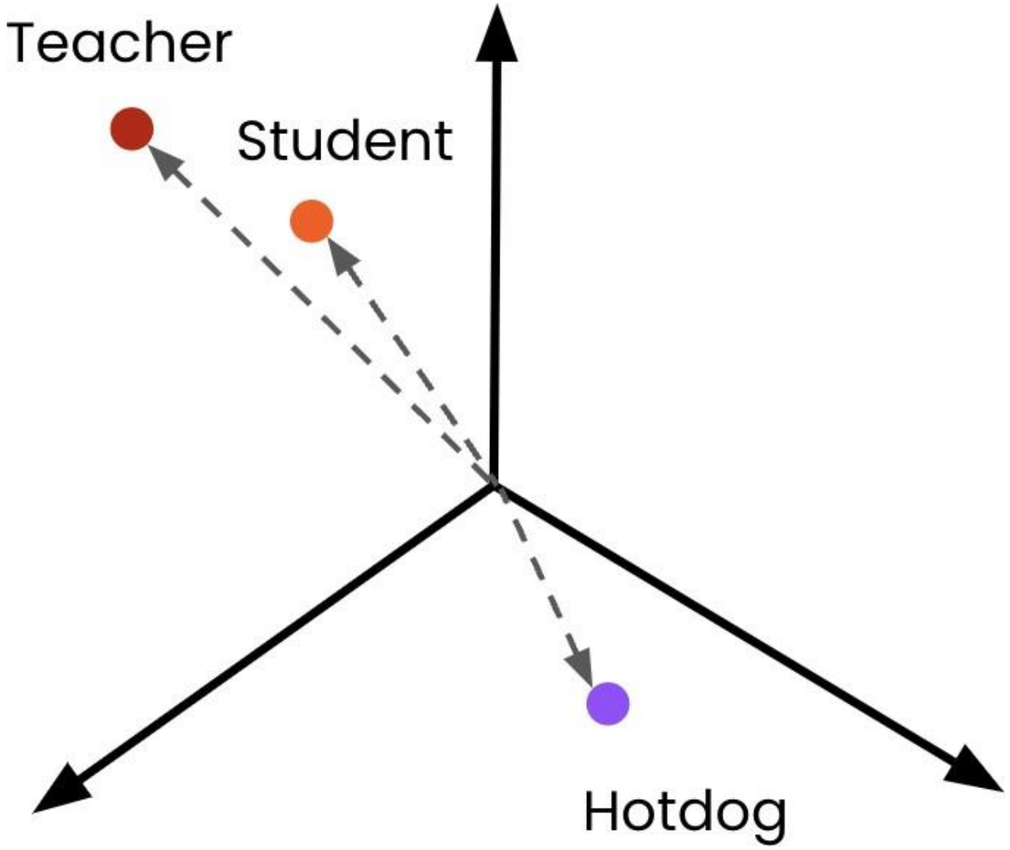
What are embeddings?

- Concept from **N**atural **L**anguage **P**rocessing (**NLP**)
- Numerical representation of text



What are embeddings?

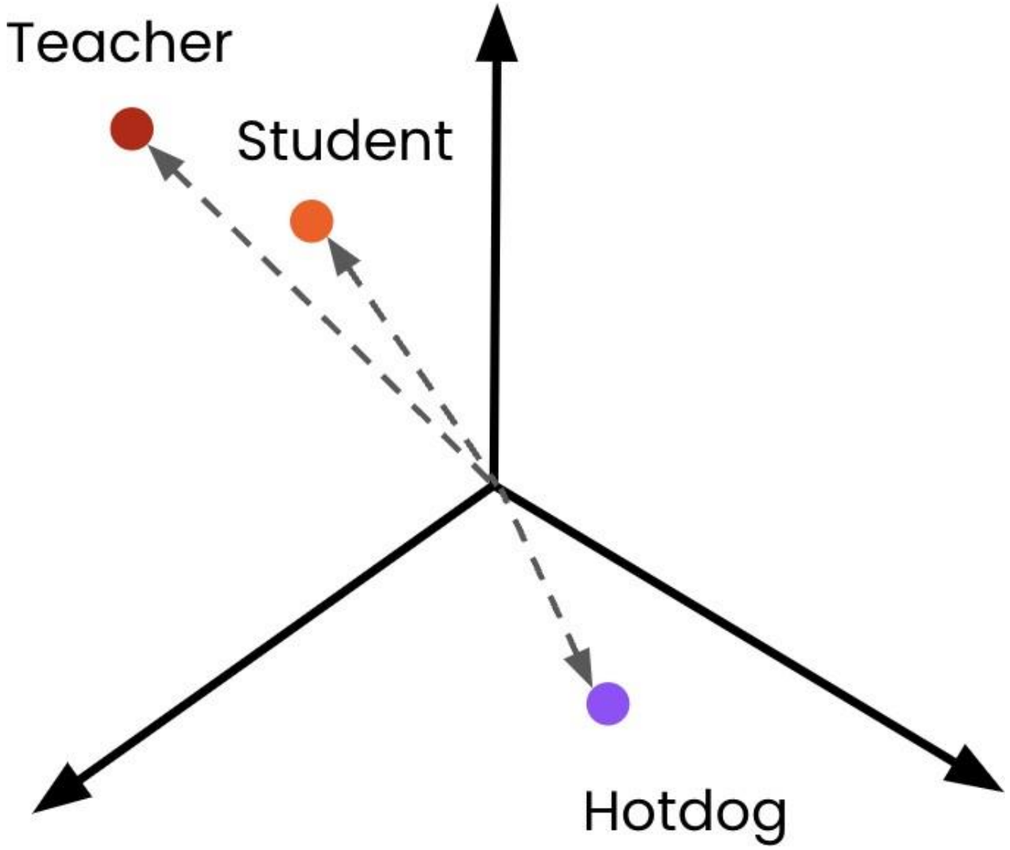
- Text is mapped onto a *multi-dimensional* **vector space**
- The numbers outputted by the model are the text's location in the space
- Similar words appear *closer together*
- Dissimilar words appear *further away*



Why are embeddings useful?

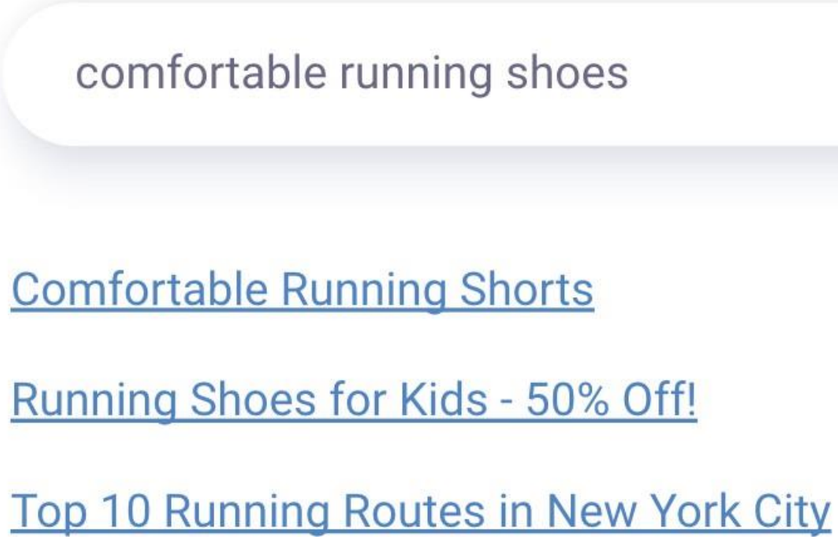
- Embeddings allow *semantic meaning* to be captured
- **Semantic meaning**: context and intent behind text

- **Example**:
  - "Which way is it to the supermarket?"
  - "Could I have directions to the shop?"



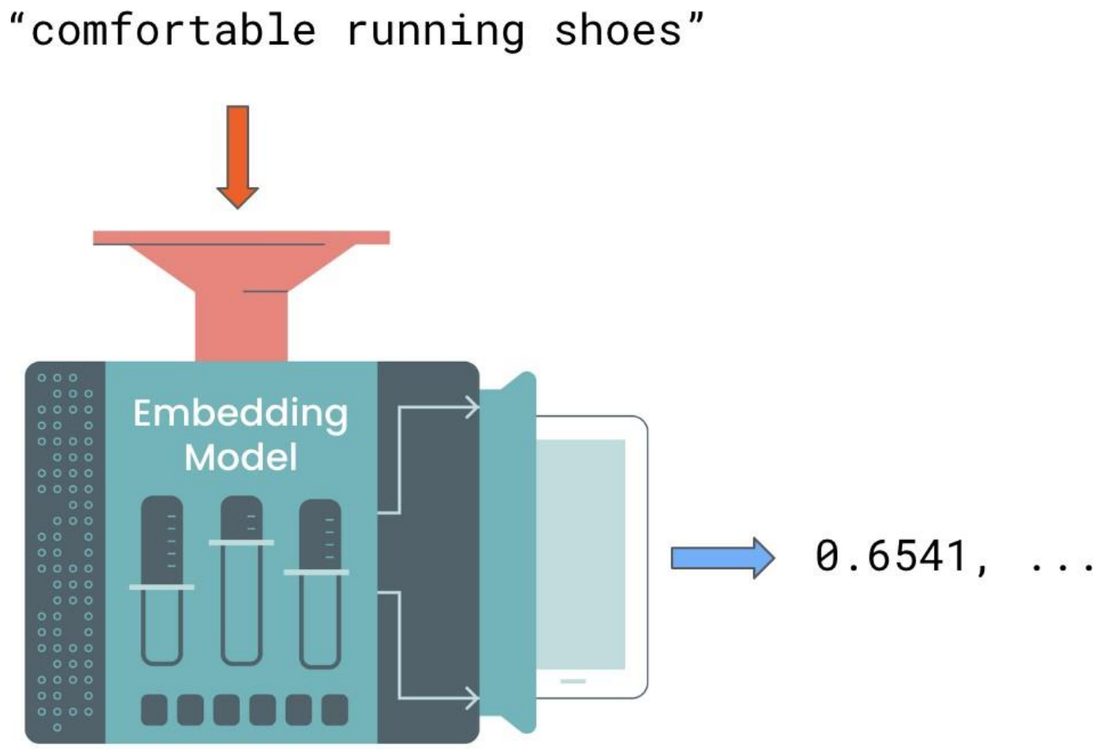
Semantic search engines

- Traditional search engines
  - Use **keyword** pattern matching
  - May miss the true intent
  - Will miss word variations



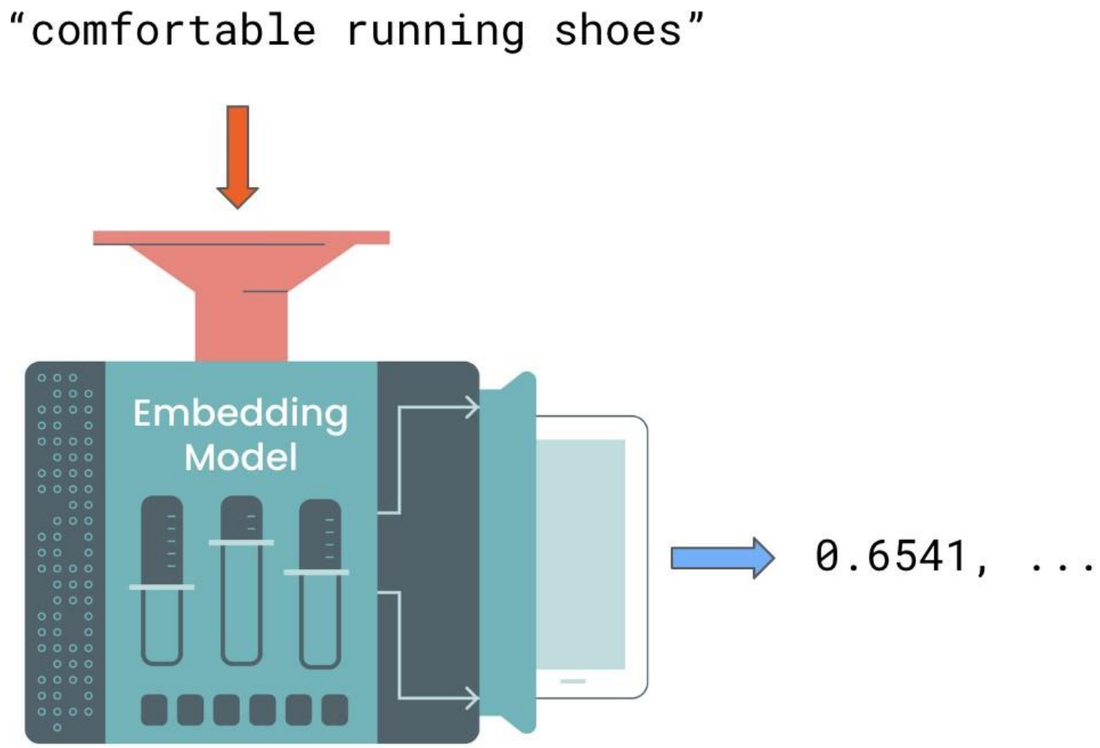
Semantic search engines

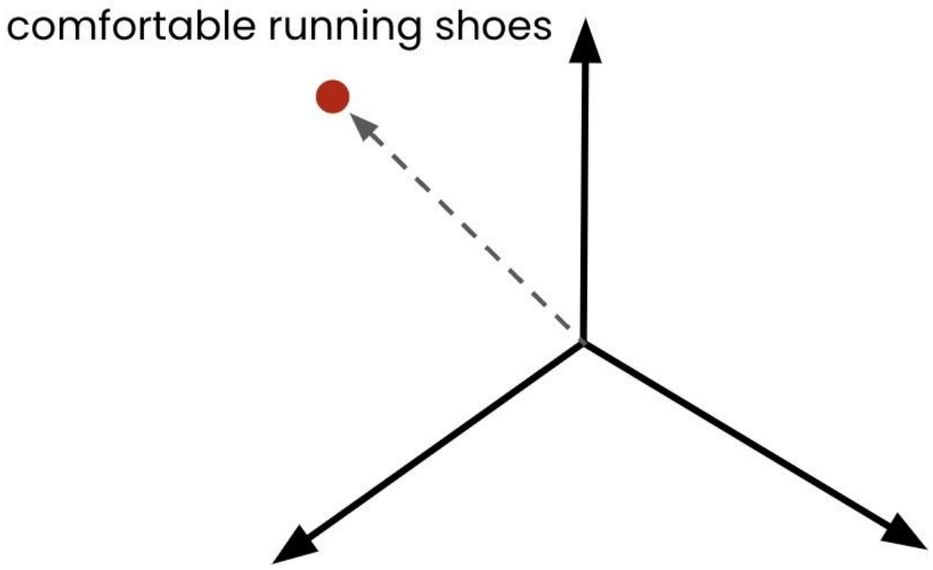
- Use **embeddings** to understand intent and context



Semantic search engines

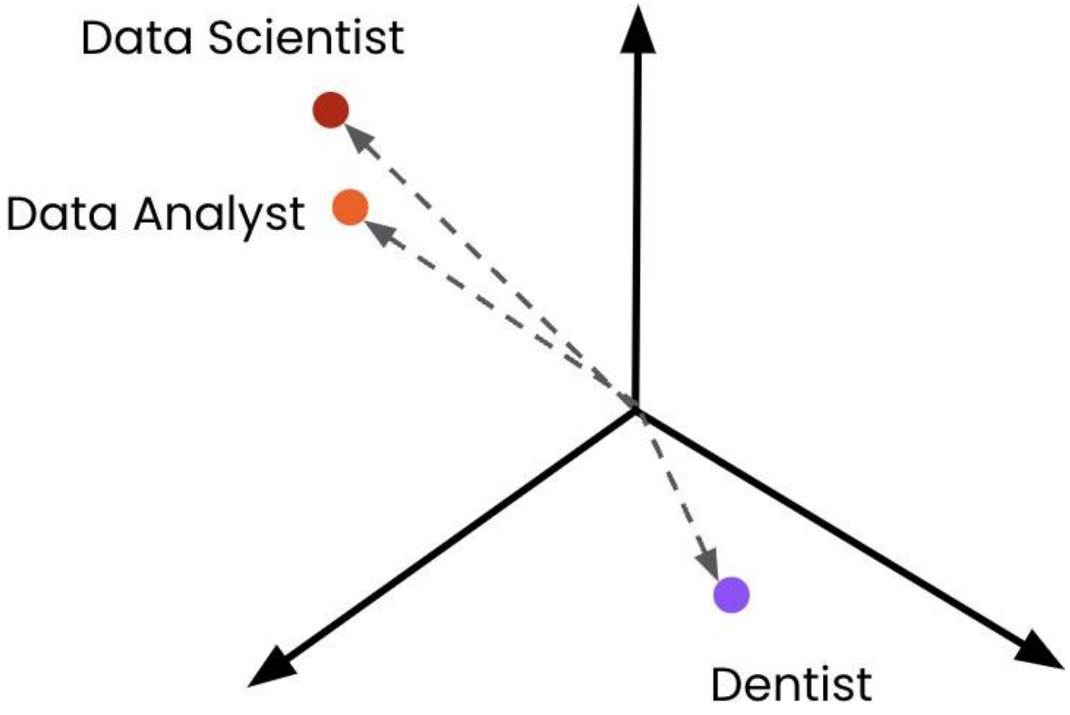
- Use **embeddings** to understand intent and context





Recommendation systems

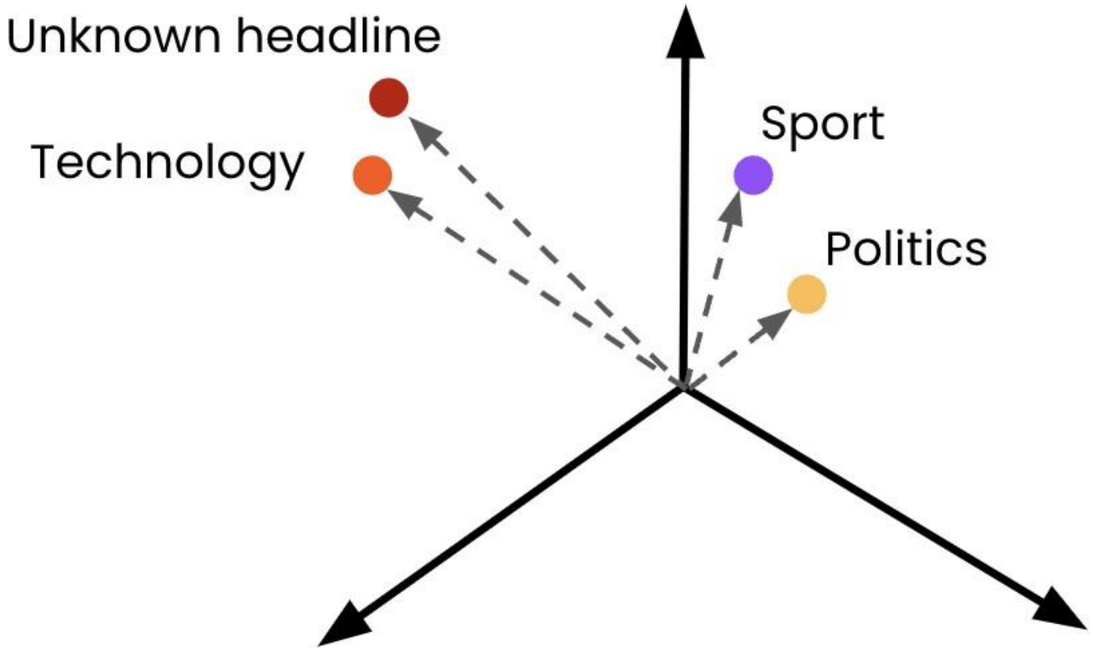
- **Example**: Job post recommendations
  - Recommend jobs based on descriptions already viewed
  - Mitigates variations in job title



Classification

Classification tasks:

- Classify sentiment
- Cluster observations
- Categorization



Example:

- Classifying news headlines

Creating an Embeddings request

- Embeddings endpoint

```
from openai import OpenAI

client = OpenAI(api_key="<OPENAI_API_KEY>")
response = client.embeddings.create(
  model="text-embedding-3-small",
  input="Embeddings are a numerical representation of text that can be used to
measure the relatedness between two pieces of text."
)

response_dict = response.model_dump()
print(response_dict)
```

Embeddings response

```
{'object': 'list',
 'data': [
    {
      "embedding": [0.0023064255, ..., -0.0028842222],
      "index":  0,
      "object": "embedding"
    }
  ],
 'model': 'text-embedding-3-small',
 'usage': {
  "prompt_tokens": 24,
  "total_tokens": 24
  }
}
```

Extracting the embeddings

```
print(response_dict['data'][0]['embedding'])
```

```
[0.0023064255, ...., -0.0028842222]
```

TriveraTech
TECHNOLOGY TRAINING

Investigating the Vector Space

Experience is Everything

Example: Embedding headlines

```
articles = [
    {"headline": "Economic Growth Continues Amid Global Uncertainty", "topic": "Business"},
    {"headline": "Interest rates fall to historic lows", "topic": "Business"},
    {"headline": "Scientists Make Breakthrough Discovery in Renewable Energy", "topic": "Science"},
    {"headline": "India Successfully Lands Near Moon's South Pole", "topic": "Science"},
    {"headline": "New Particle Discovered at CERN", "topic": "Science"},
    {"headline": "Tech Company Launches Innovative Product to Improve Online Accessibility", "topic": "Tech"},
    {"headline": "Tech Giant Buys 49% Stake In AI Startup", "topic": "Tech"},
    {"headline": "New Social Media Platform Has Everyone Talking!", "topic": "Tech"},
    {"headline": "The Blues get promoted on the final day of the season!", "topic": "Sport"},
    {"headline": "1.5 Billion Tune-in to the World Cup Final", "topic": "Sport"}
]
```

Example: Embedding headlines

```
[
  {
      'headline': 'The Blues get promoted on the final day!',
      'topic': 'Sport',
      'embedding': [0.0015748793, ..., -0.0052598542]
  },
  {
      'headline': 'Interest rates fall to historic levels',
      'topic': 'Business',
      'embedding': [-0.030351446, ..., -0.0044289114]
  },
  ...
]
```

Embedding multiple inputs

```
headline_text = [article['headline'] for article in articles]
headline_text
```

```
["Economic Growth Continues Amid Global Uncertainty",
 ...,
 "1.5 Billion Tune-in to the World Cup Final"]
```

```
response = client.embeddings.create(
  model="text-embedding-3-small",
  input=headline_text
)
response_dict = response.model_dump()
```

- **Batching** is more efficient than using multiple API calls

Embedding multiple inputs

```
articles = [
    {"headline": "Economic Growth Continues Amid Global Uncertainty", "topic":
    "Business"},
     ...
]
```

```
for            in
for i, article in enumerate(articles):
    article['embedding'] = response_dict['data'][i]['embedding']


print(articles[:2])
```

```
[{'headline': 'Economic Growth Continues Amid Global
  Uncertainty', 'topic': 'Business',
  'embedding': [-0.017142612487077713, ..., -0.0012911480152979493]}
 {'headline': 'Interest rates fall to historic
  lows', 'topic': 'Business',
  'embedding': [-0.032995883375406265, ..., -0.00286053004674613 48]}]
```

How long is the embeddings vector?

- "Economic Growth Continues Amid Global Uncertainty"

```
len(articles[0]['embedding'])
```

```
1536
```

- "Tech Company Launches Innovative Product to Improve Accessibility"

```
len(articles[5]['embedding'])
```

```
1536
```

- **Always returns 1536 numbers!**

Dimensionality reduction and t-SNE

- Various techniques to *reduce* the number of dimensions
- **t-SNE** (**t**-distributed **S**tochastic **N**eighbor **E**mbedding)

Implementing t-SNE

```
from sklearn.manifold import TSNE
import numpy as np

embeddings = [article['embedding'] for article in articles]

tsne = TSNE(n_components=2, perplexity=5)

embeddings_2d = tsne.fit_transform(np.array(embeddings))
```

- `n_components` : the resulting number of dimensions
- `perplexity` : used by the algorithm, must be *less than number of data points*
- Will result in **information loss**

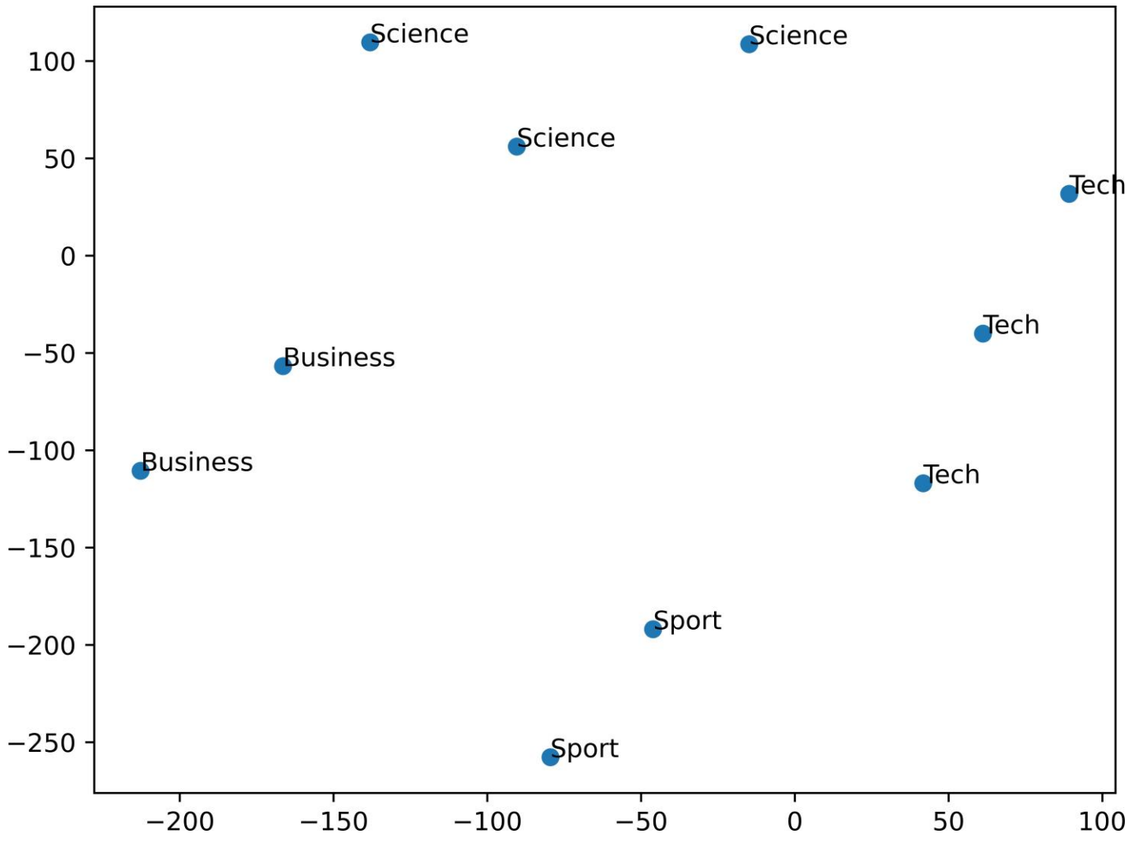
Visualizing the embeddings

```
import matplotlib.pyplot as plt

plt.scatter(embeddings_2d[:, 0], embeddings_2d[:, 1])

topics = [article['topic'] for article in articles]
for i, topic in enumerate(topics):
    plt.annotate(topic, (embeddings_2d[i, 0], embeddings_2d[i, 1]))

plt.show()
```

Visualizing the embeddings

- *Similar* articles are grouped together!
- Model captured the *semantic* meaning

- **Coming up**: Computing similarity



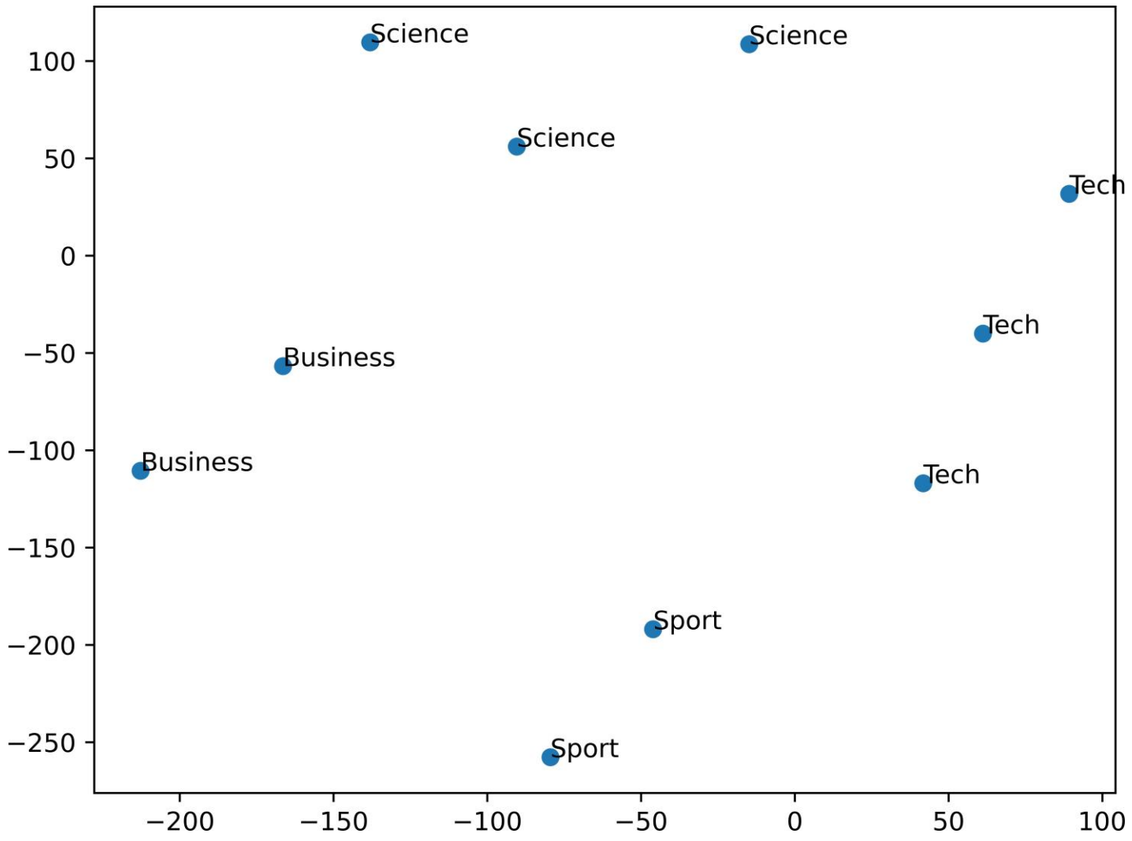
TriveraTech
TECHNOLOGY TRAINING

Text similarity

Experience is Everything

Recap...

- Semantically similar texts are embedded *more closely* in the **vector space**
- Measuring distance allows us to measure similarity
- Enables embeddings applications:
  - Semantic search
  - Recommendations
  - Classification



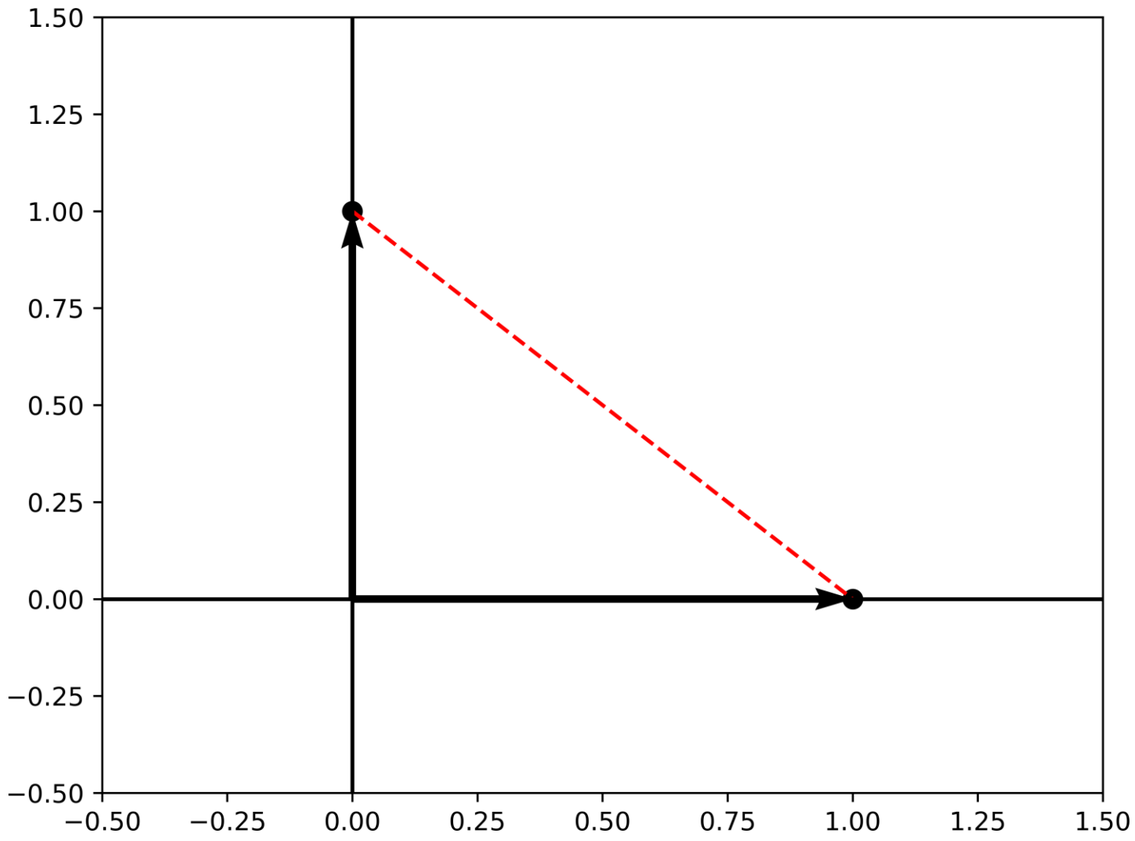
Measuring similarity

Cosine distance

```
from scipy.spatial import distance

distance.cosine([0, 1], [1, 0])
```

```
1.0
```

- Ranges from 0 to 2
- Smaller numbers = *Greater* similarity



Example: Comparing headline similarity

```
[
  {
      'headline': 'The Blues get promoted on the final day!',
      'topic': 'Sport',
      'embedding': [0.0015748793, ..., -0.0052598542]
  },
  {
      'headline': 'Interest rates fall to historic levels',
      'topic': 'Business',
      'embedding': [-0.030351446, ..., -0.0044289114]
  },
  ...
]
```

Example: Comparing headline similarity

```python
def create_embeddings(texts):
  response = client.embeddings.create(
    model="text-embedding-3-small",
    input=texts
  )
  response_dict = response.model_dump()

  return [data['embedding'] for data in response_dict['data']]
```

```python
print(create_embeddings(["Python is the best!", "R is the best!"]))
print(create_embeddings("Trivera is awesome!")[0])
```

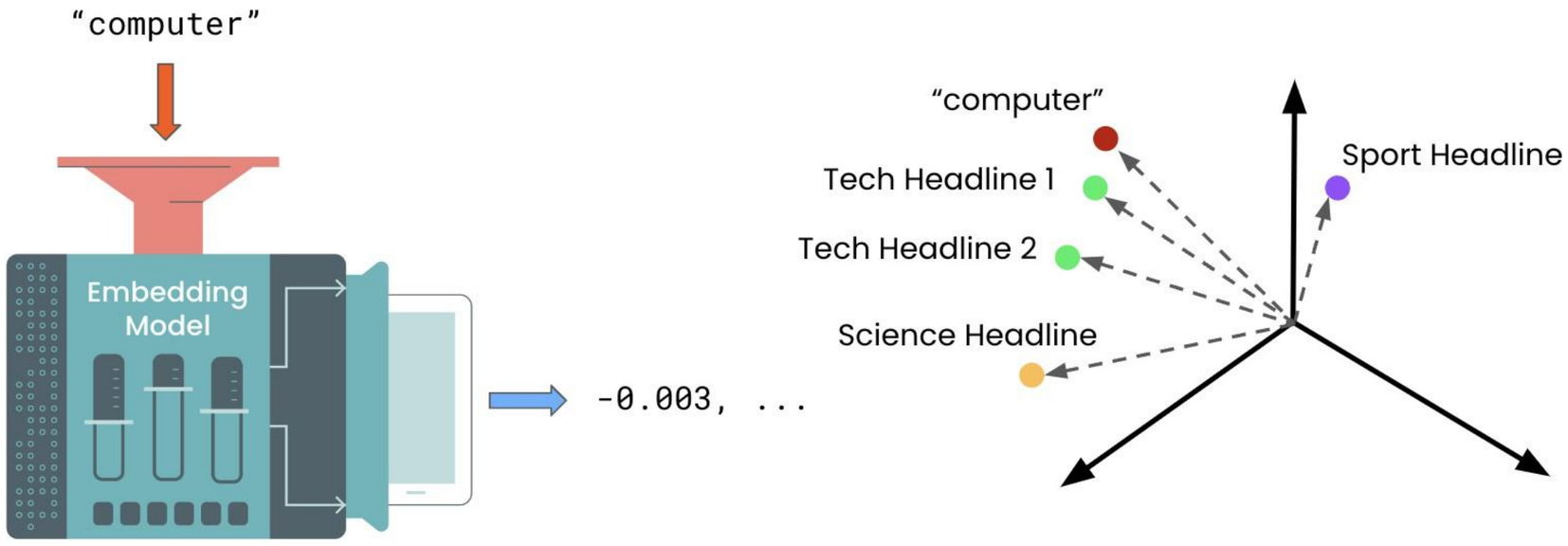
```
[[0.0050565884448587894, ..., , -0.040003236638319969],
 [-0.0018890155479311943, ..., -0.04085670784115791]]
[0.00037010075175203383, ..., -0.0217591002583503 72]
```

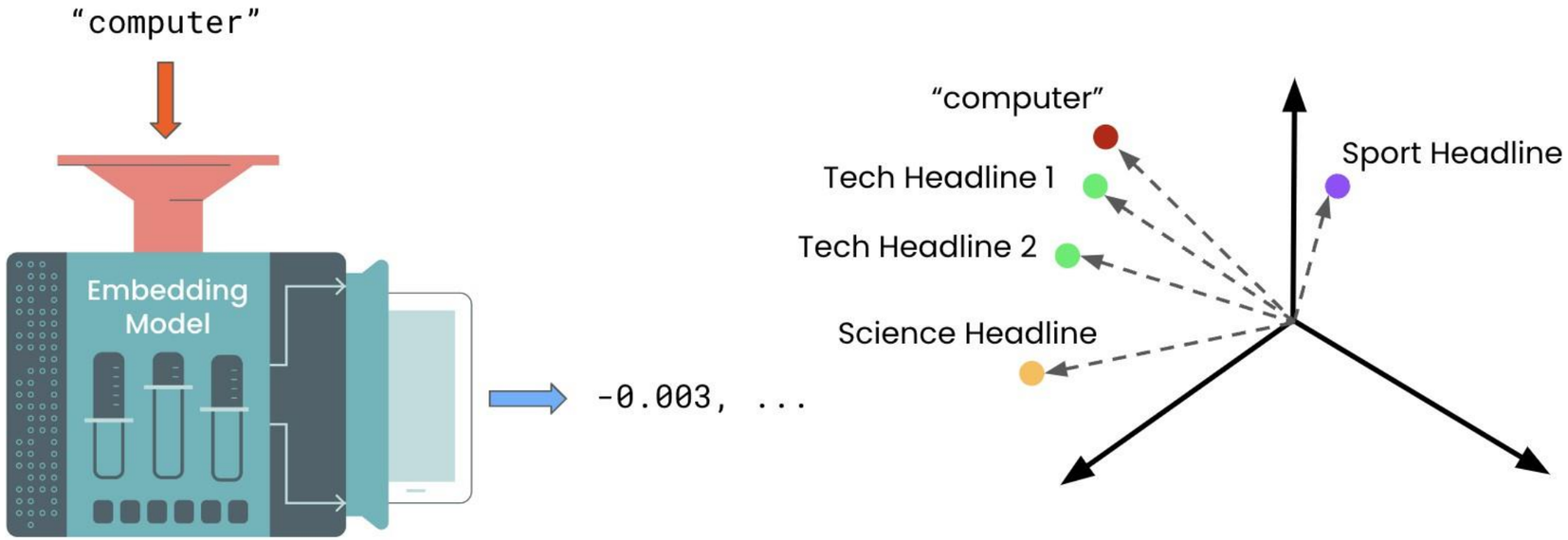
Example: Comparing headline similarity

```
from scipy.spatial import distance
import numpy as np

search_text = "computer"
search_embedding = create_embeddings(search_text)[0]

distances = []
for article in articles:
  dist = distance.cosine(search_embedding, article["embedding"])
  distances.append(dist)

min_dist_ind = np.argmin(distances)
print(articles[min_dist_ind]['headline'])
```

```
Tech Company Launches Innovative Product to Improve Online Accessibility
```

- Use embeddings to return *most similar* results to a search query
- **Example**: Semantic search for online news website





1. **Embed** the search query and other texts
2. Compute the **cosine distances**
3. **Extract** the texts with the *smallest* cosine distance

```
articles = [
    {"headline": "Economic Growth Continues Amid Global Uncertainty",
     "topic": "Business",
     "keywords": ["economy", "business", "finance"]},
    ...
    {"headline": "1.5 Billion Tune-in to the World Cup Final",
     "topic": "Sport",
     "keywords": ["soccer", "world cup", "tv"]}
]
```

```
Headline: Economic Growth Continues Amid Global Uncertainty
Topic: Business
Keywords: economy, business, finance
```

```python
articles = [..., {"headline": "1.5 Billion Tune-in to the World Cup ",
                  "topic": "Sport",
                  "keywords": ["soccer", "world cup", "tv"]}]

def create_article_text(article):
  return f"""Headline: {article['headline']}
Topic: {article['topic']}
Keywords: {', '.join(article['keywords'])}"""

print(create_article_text(articles[-1]))
```

```
Headline: 1.5 Billion Tune-in to the World Cup Final
Topic: Sport
Keywords: soccer, world cup, tv
```

Creating enriched embeddings

```
article_texts = [create_article_text(article) for article in articles]


article_embeddings = create_embeddings(article_texts)
print(article_embeddings)
```

```
[[-0.019609929993748665, -0.033318601548671 72, ...],
 ...,
 [..., -0.014373429119586945, -0.005235843360424042]]
```

```python
from scipy.spatial import distance

def find_n_closest(query_vector, embeddings, n=3):
  distances = []
  for index, embedding in enumerate(embeddings):
    dist = distance.cosine(query_vector, embedding)
    distances.append({"distance": dist, "index": index})
  distances_sorted = sorted(distances, key=lambda x: x["distance"])
  return distances_sorted[0:n]
```

```python
query_text = "AI"
query_vector = create_embeddings(query_text)[0]

hits = find_n_closest(query_vector, article_embeddings)


for hit in hits:
  article = articles[hit['index']]
  print(article['headline'])
```

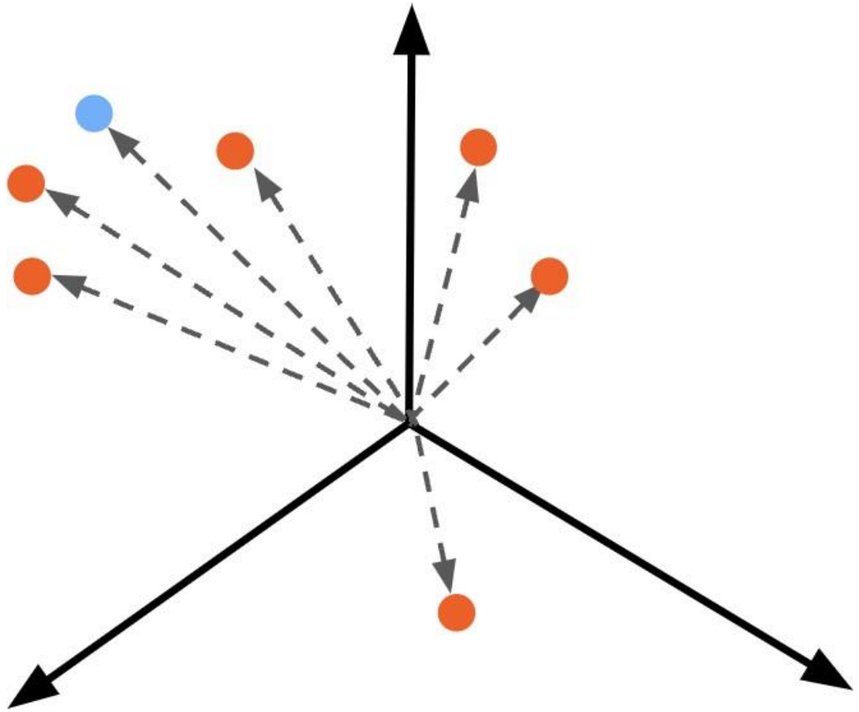
```
Tech Giant Buys 49% Stake In AI Startup
Tech Company Launches Innovative Product to Improve Online Accessibility
India Successfully Lands Near Moon's South Pole
```

Recommendation systems with embeddings

- Very similar to semantic search!

Process:

1. Embed the potential recommendations and data point

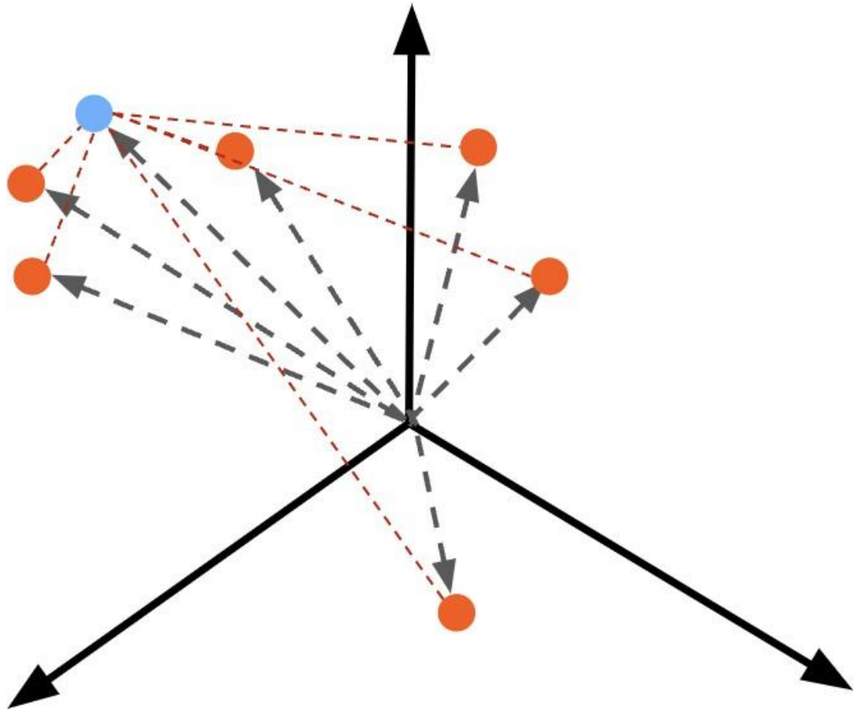


Recommendation systems with embeddings

- Very similar to semantic search!

Process:

1. Embed the potential recommendations and data point
2. Calculate cosine distances

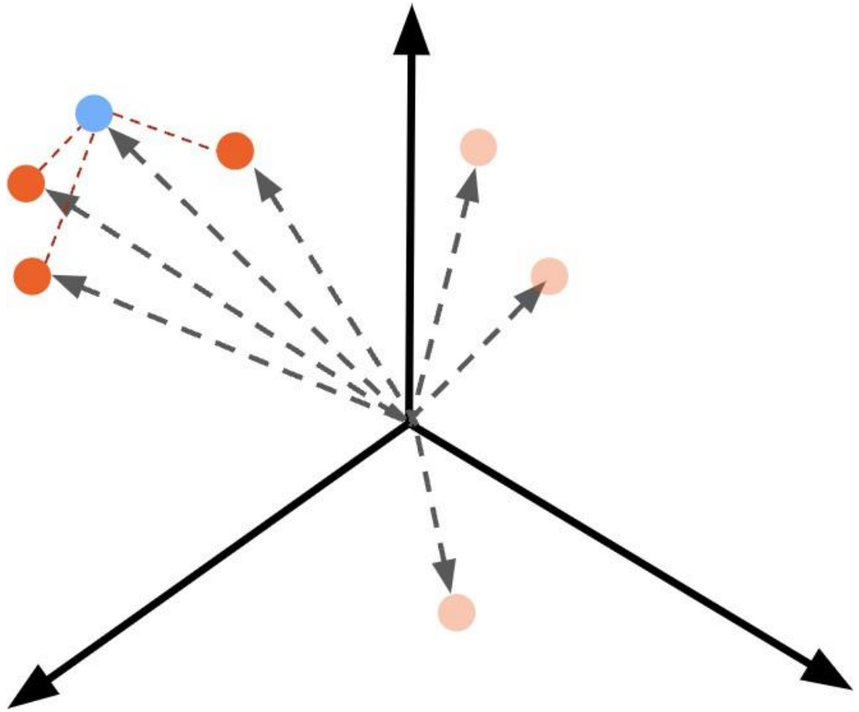


Recommendation systems with embeddings

- Very similar to semantic search!

Process:

1. Embed the potential recommendations and data point
2. Calculate cosine distances
3. Recommend closest items



```
articles = [
    {"headline": "Economic Growth Continues Amid Global Uncertainty",
     "topic": "Business",
     "keywords": ["economy", "business", "finance"]},
    ...
    {"headline": "1.5 Billion Tune-in to the World Cup Final",
     "topic": "Sport",
     "keywords": ["soccer", "world cup", "tv"]}
]

current_article = {"headline": "How NVIDIA GPUs Could Decide Who Wins the AI Race",
                   "topic": "Tech",
                   "keywords": ["ai", "business", "computers"]}
```

Combining features

```
def create_article_text(article):
  return f"""Headline: {article['headline']}
Topic: {article['topic']}
Keywords: {', '.join(article['keywords'])}"""
```

```
article_texts = [create_article_text(article) for article in articles]
current_article_text = create_article_text(current_article)
print(current_article_text)
```

```
Headline: How NVIDIA GPUs Could Decide Who Wins the AI Race
Topic: Tech
Keywords: ai, business, computers
```

```python
def create_embeddings(texts):
  response = openai.Embedding.create(
    model="text-embedding-3-small",
    input=texts
  )
  response_dict = response.model_dump()

  return [data['embedding'] for data in response_dict['data']]
```

```python
current_article_embeddings = create_embeddings(current_article_text)[0]
article_embeddings = create_embeddings(article_texts)
```

Finding the most similar article

```
def find_n_closest(query_vector, embeddings, n=3):
  distances = []
  for index, embedding in enumerate(embeddings):
    dist = spatial.distance.cosine(query_vector, embedding)
    distances.append({"distance": dist, "index": index})
  distances_sorted = sorted(distances, key=lambda x: x["distance"])
  return distances_sorted[0:n]
```

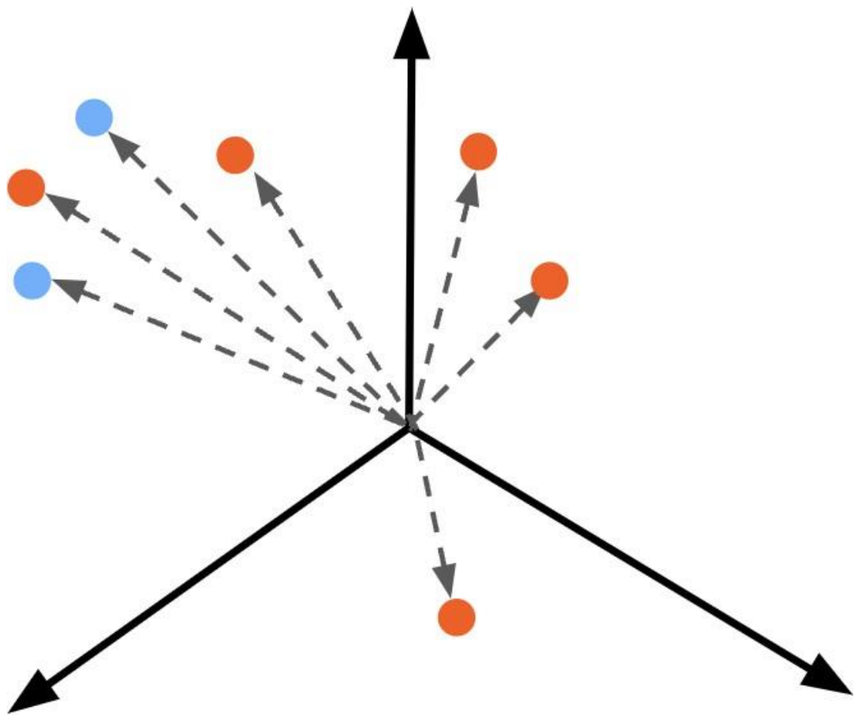
```
hits = find_n_closest(current_article_embeddings, article_embeddings)


for hit in hits:
  article = articles[hit['index']]
  print(article['headline'])
```

Finding the most similar article

```
Tech Giant Buys 49% Stake In AI Startup
Tech Company Launches Innovative Product to Improve Online Accessibility
Scientists Make Breakthrough Discovery in Renewable Energy
```

```
user_history = [
    {"headline": "How NVIDIA GPUs Could Decide Who Wins the AI Race",
     "topic": "Tech",
     "keywords": ["ai", "business", "computers"]},
    {"headline": "Tech Giant Buys 49% Stake In AI Startup",
     "topic": "Tech",
     "keywords": ["business", "AI"]}
]
```

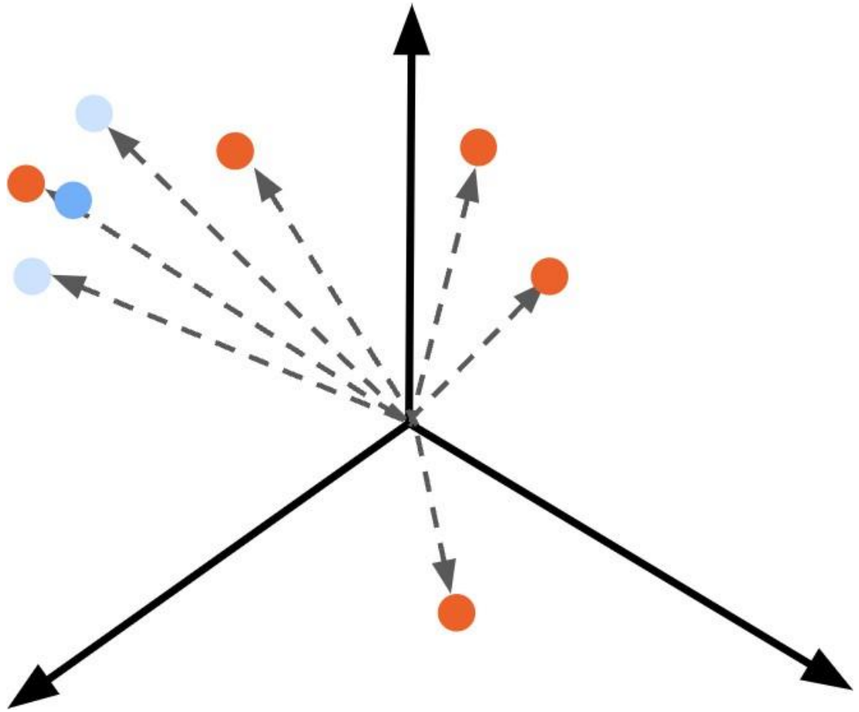
Recommendations on multiple data points



Recommendations on multiple data points

Process:

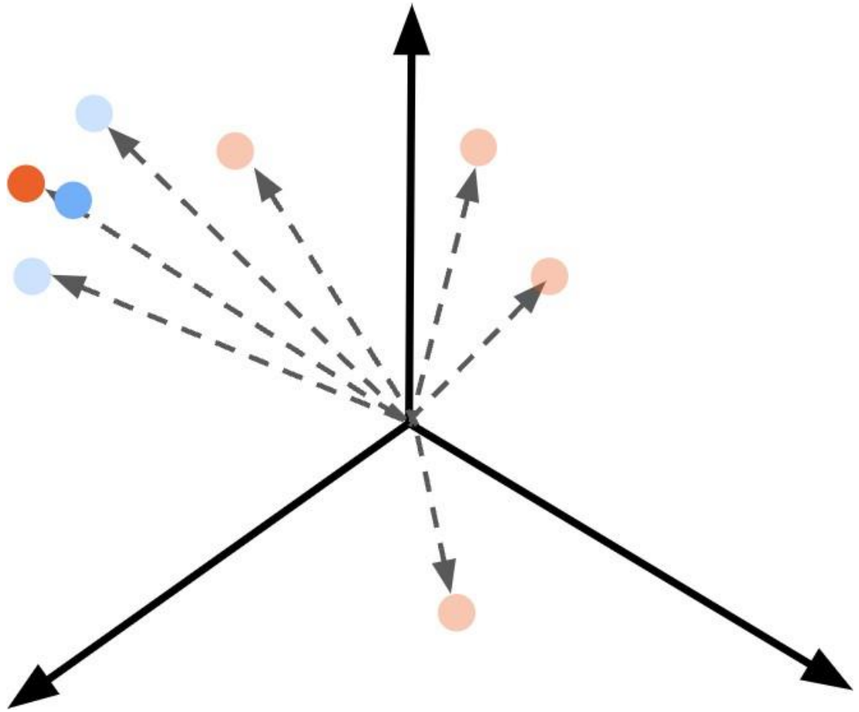
- Combine multiple vectors into one by taking the **mean**
- Compute cosine distances



Recommendations on multiple data points

Process:

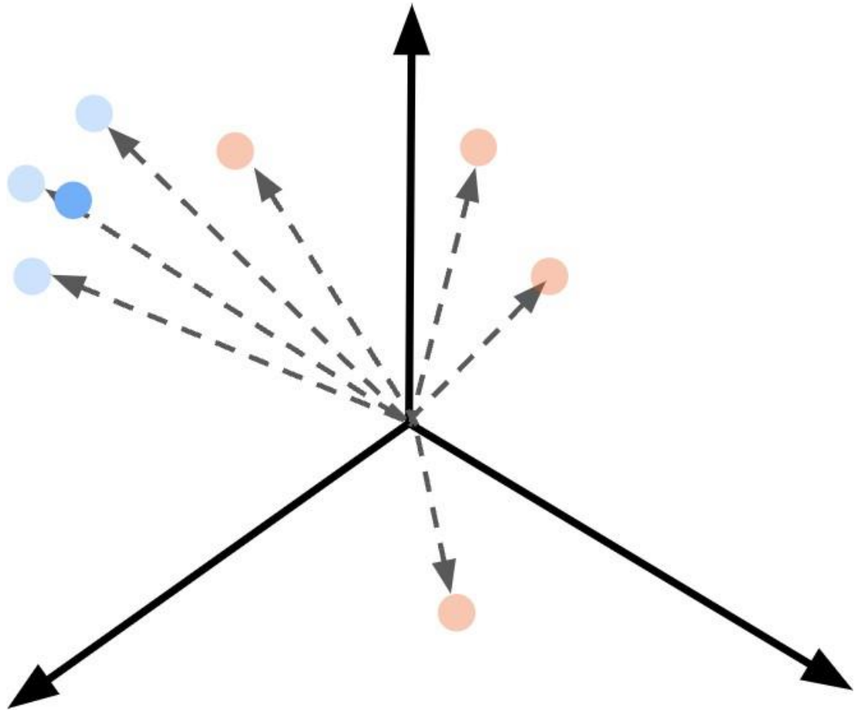
- Combine multiple vectors into one by taking the **mean**
- Compute cosine distances
- Recommend closest vector



Recommendations on multiple data points

Process:

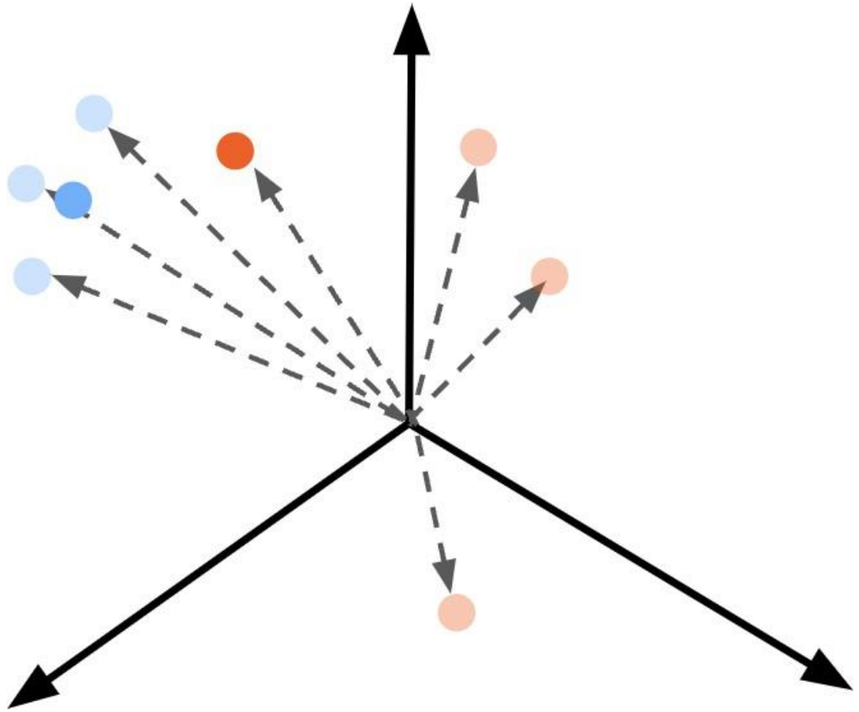
- Combine multiple vectors into one by taking the **mean**
- Compute cosine distances
- Recommend closest vector



Recommendations on multiple data points

Process:

- Combine multiple vectors into one by taking the **mean**
- Compute cosine distances
- Recommend closest vector
  - Ensure that it's unread



```python
def create_article_text(article):
  return f"""Headline: {article['headline']}
Topic: {article['topic']}
Keywords: {', '.join(article['keywords'])}"""

history_texts = [create_article_text(article) for article in user_history]
history_embeddings = create_embeddings(history_texts)
mean_history_embeddings = np.mean(history_embeddings, axis=0)

articles_filtered = [article for article in articles if article not in user_history]
article_texts = [create_article_text(article) for article in articles_filtered]
article_embeddings = create_embeddings(article_texts)
```

Recommendations on multiple data points

```
hits = find_n_closest(mean_history_embeddings, article_embeddings)


for hit in hits:
  article = articles_filtered[hit['index']]
  print(article['headline'])
```

```
Tech Company Launches Innovative Product to Improve Online Accessibility
New Social Media Platform Has Everyone Talking!
Scientists Make Breakthrough Discovery in Renewable Energy
```

Classification tasks

Assigning **labels** to items

- Categorization
  - **Example**: headlines into topics
- Sentiment analysis

Sport	Tech	Business	Science

Classification tasks

Assigning **labels** to items

- Categorization
  - **Example**: headlines into topics
- Sentiment analysis
  - **Example**: Classifying reviews as positive or negative

Sport	Tech	Business	Science

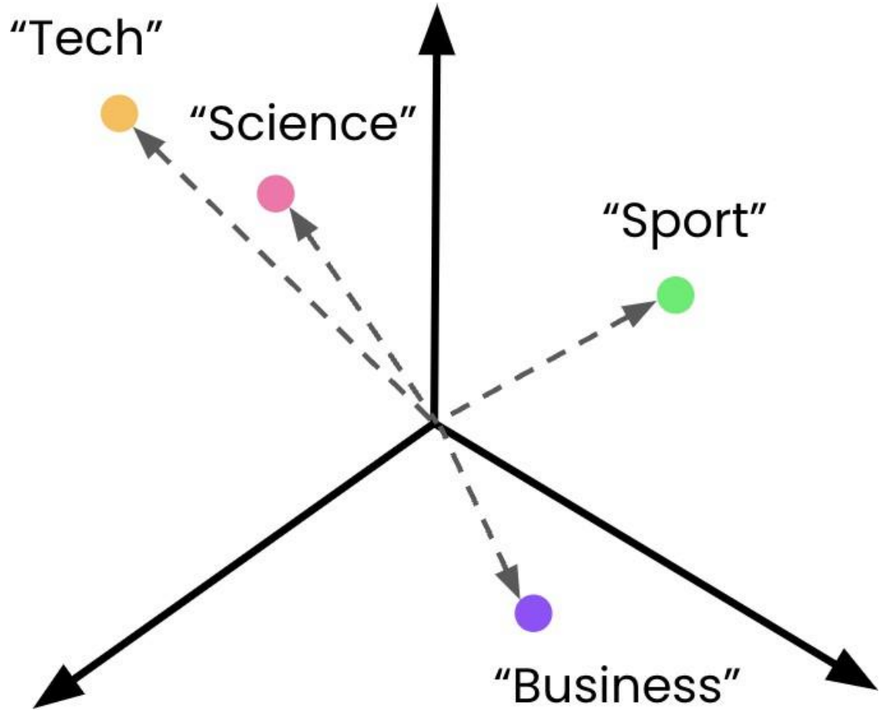
Embeddings capture *semantic* meaning

Classification with embeddings

- **Zero-shot classification**: Not using labeled data
  -

Process:

1. Embed class descriptions

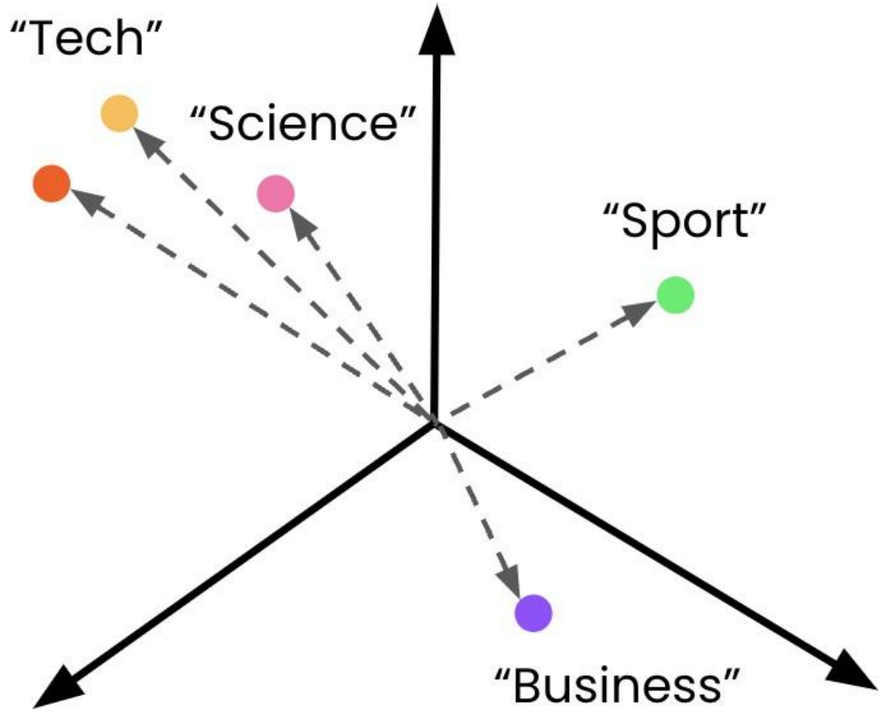


Classification with embeddings

- **Zero-shot classification**: Not using labeled data
  -

Process:

1. Embed class descriptions
2. Embed the item to classify
3. Compute cosine distances

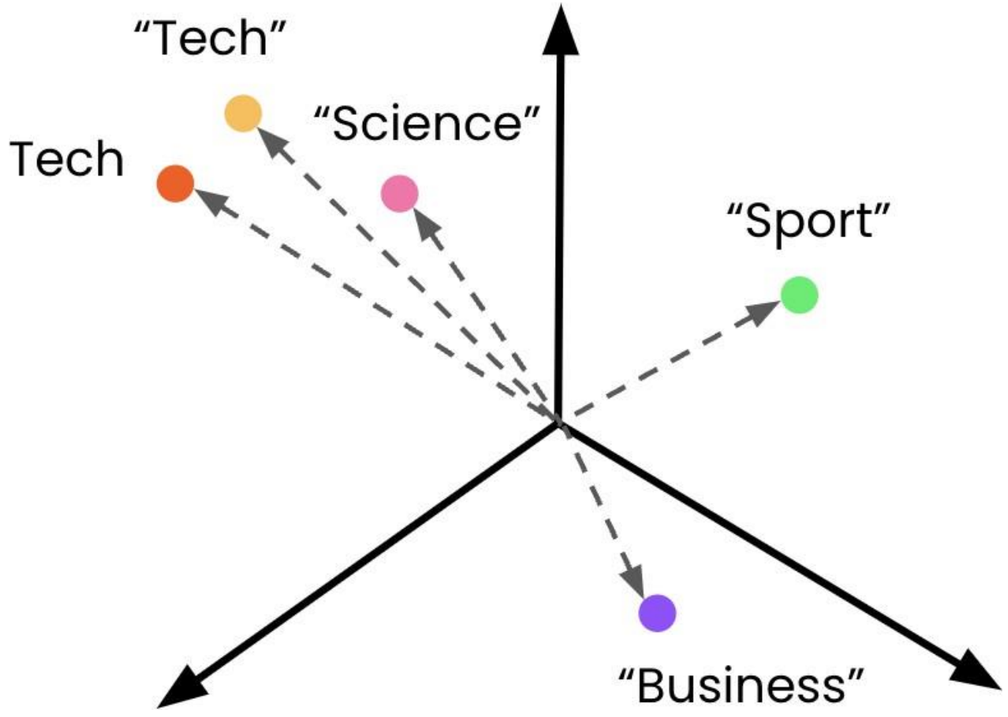


Classification with embeddings

- **Zero-shot classification**: Not using labeled data
  -

Process:

1. Embed class descriptions
2. Embed the item to classify
3. Compute cosine distances
4. Assign the most similar label



```
topics = [
  {'label': 'Tech'},
  {'label': 'Science'},
  {'label': 'Sport'},
  {'label': 'Business'},
]

class_descriptions = [topic['label'] for topic in topics]
class_embeddings = create_embeddings(class_descriptions)
```

```python
article = {"headline": "How NVIDIA GPUs Could Decide Who Wins the AI Race",
           "keywords": ["ai", "business", "computers"]}

def create_article_text(article):
  return f"""Headline: {article['headline']}
Keywords: {', '.join(article['keywords'])}"""

article_text = create_article_text(article)
article_embeddings = create_embeddings(article_text)[0]
```

Compute cosine distances

```
def find_closest(query_vector, embeddings):
  distances = []
  for index, embedding in enumerate(embeddings):
    dist = distance.cosine(query_vector, embedding)
    distances.append({"distance": dist, "index": index})
  return min(distances, key=lambda x: x["distance"])


closest = find_closest(article_embeddings, class_embeddings)
```

Extract the most similar label

```
label = topics[closest['index']]['label']
print(label)
```

```
Business
```

```
article = {"headline": "How NVIDIA GPUs Could Decide Who Wins the AI Race",
           "keywords": ["ai", "business", "computers"]}
```

Limitation:

- Class descriptions lacked sufficient detail

```python
topics = [
  {'label': 'Tech', 'description': 'A news article about technology'},
  {'label': 'Science', 'description': 'A news article about science'},
  {'label': 'Sport', 'description': 'A news article about sports'},
  {'label': 'Business', 'description': 'A news article about business'},
]

class_descriptions = [topic['description'] for topic in topics]
class_embeddings = create_embeddings(class_descriptions)
[...]
label = topics[closest['index']]['label']
print(label)
```

```
Tech
```

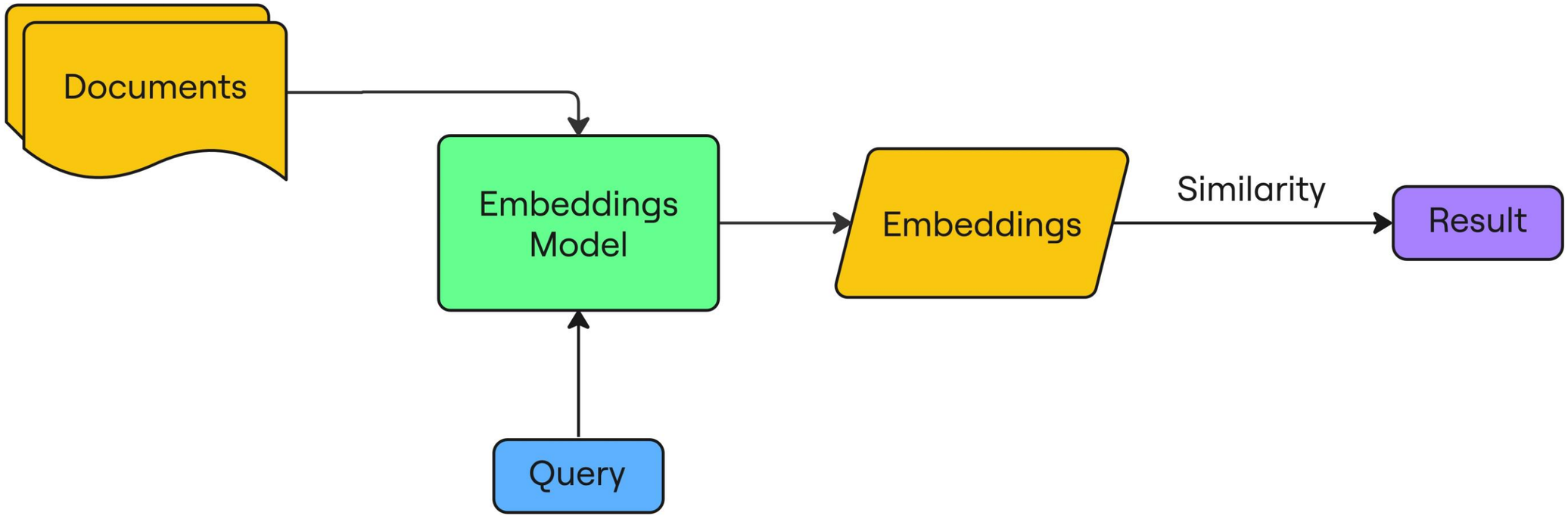
TriveraTech
TECHNOLOGY TRAINING

Vector databases for embedding systems

Experience is Everything

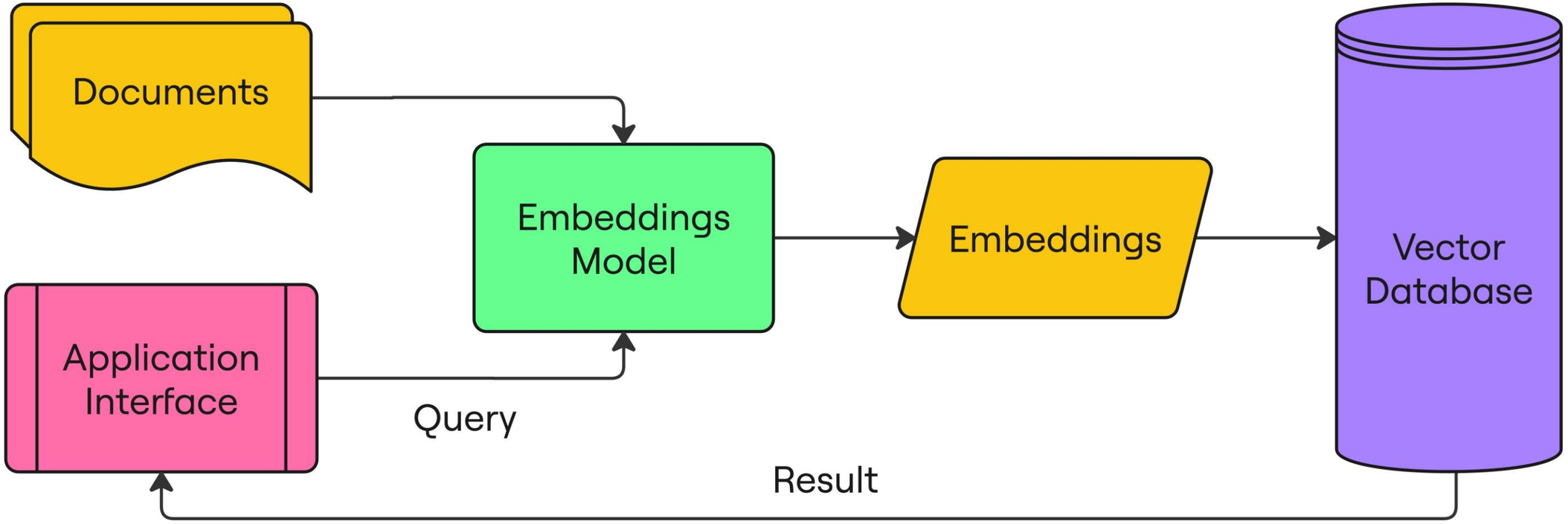
Limitations of the current approach

- Loading all the embeddings into memory (1536 floats ~ **13kB/embedding**)
- Recalculated embeddings for each new query
- Calculating cosine distances for every embedding and sorting is slow and **scales linearly**



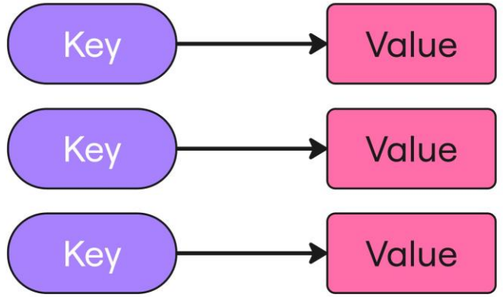
Vector databases

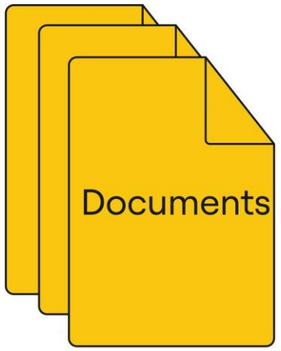
- Embedded documents are *stored* and *queried* from the **vector database**

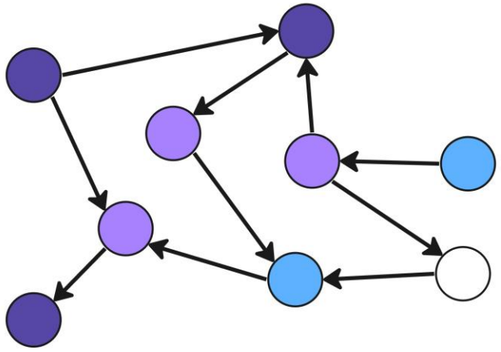


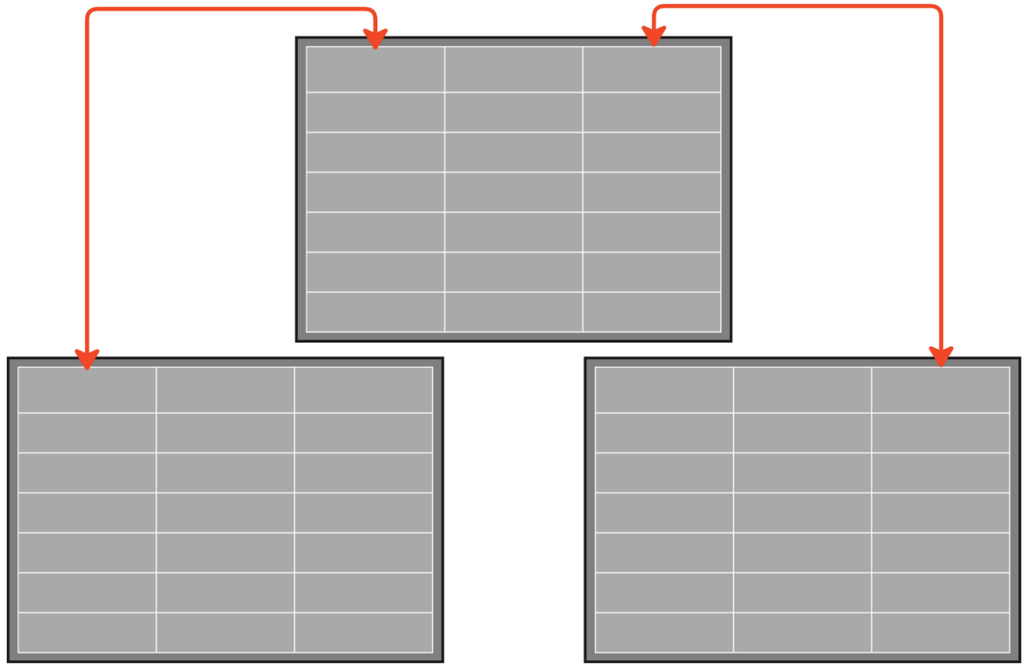
NoSQL Database

- More flexible structure that allows for *faster querying*
-







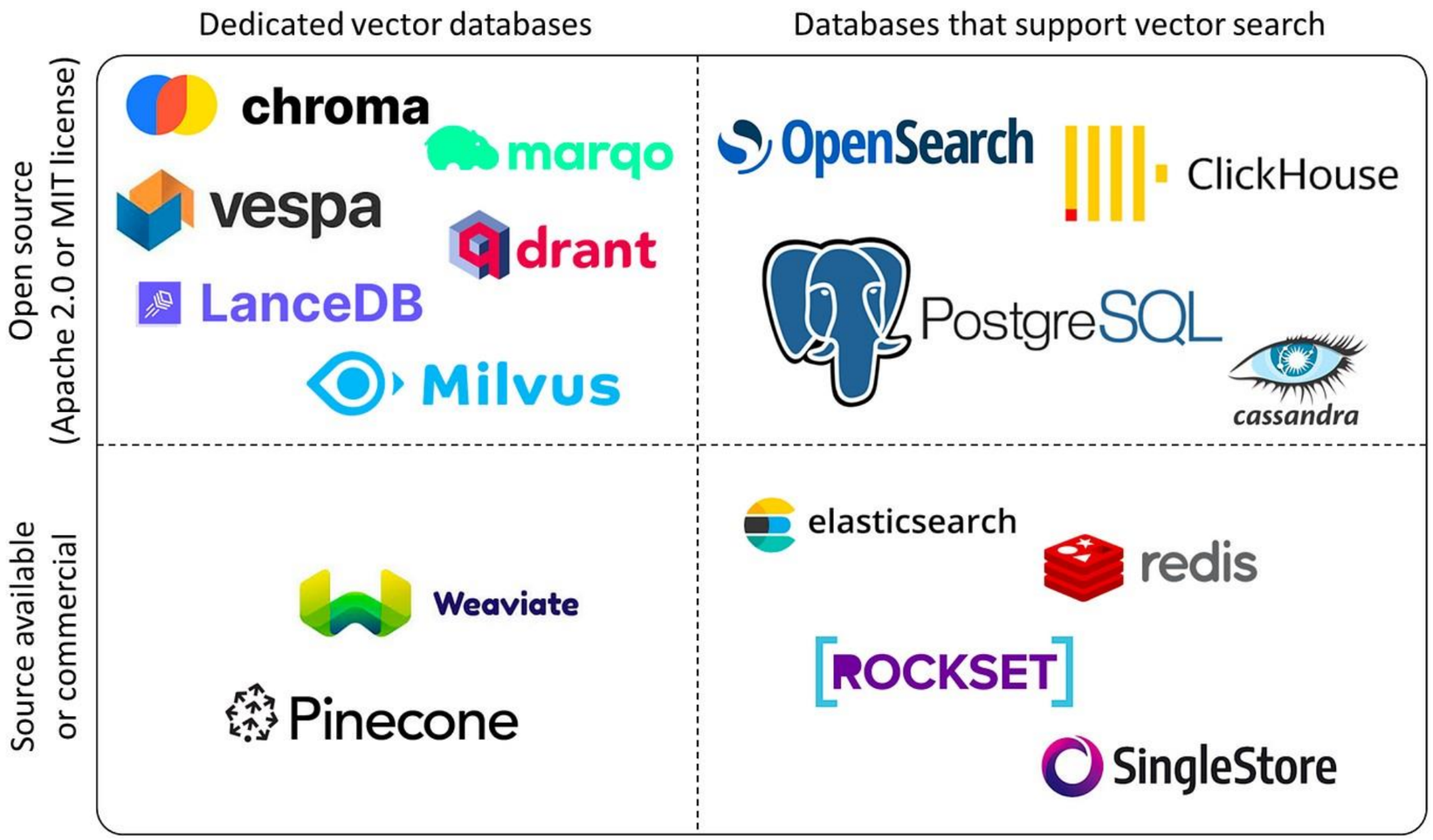


Components to store

- Embeddings
- Source texts
- Metadata
  - IDs and references
  - Additional data useful for filtering results

Top tip: Don't store the source text as metadata!

The vector database landscape



Which solution is best?

- **Database management**:
  - Managed →more expensive but lowers workload
  - Self-managed →cheaper but requires time and expertise
- **Open source or commercial?**
  - Open source →flexible and cost-effective
  - Commercial →better support, more advanced features, and compliance
- **Data models**: does the type of data lend itself to a particular database type?
- **Specific features**: does your use case depend on specific functionality, such as multi-modal storage?

chroma

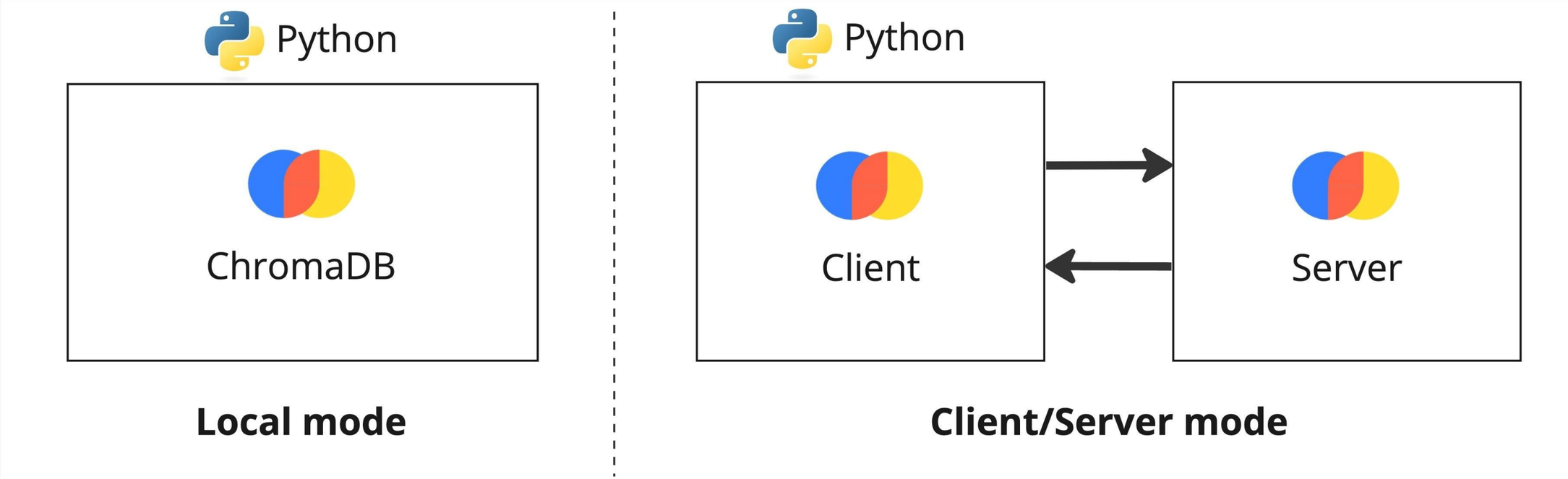
TriveraTech
TECHNOLOGY TRAINING

Creating vector databases with ChromaDB

Experience is Everything

Installing ChromaDB

- *ChromaDB* is a simple yet powerful vector database
- Two flavors:
  - **Local**: great for development and prototyping
  - **Client/Server**: made for production



Connecting to the database

```
import chromadb


client = chromadb.PersistentClient(path="/path/to/save/to")
```

- Data will be persisted to disk

Creating a collection

- Collections are analogous to tables

```
from chromadb.utils.embedding_functions import OpenAIEmbeddingFunction
```

```
collection = client.create_collection(
    name="my_collection",
    embedding_function=OpenAIEmbeddingFunction(
        model_name="text-embedding-3-small",
        api_key="..."
    )
)
```

- Collections are able to create embeddings automatically

Inspecting collections

```
client.list_collections()
```

```
[Collection(name=my_collection)]
```

Inserting embeddings

Single document

```
collection.add(ids=["my-doc"], documents=["This is the source text"])
```

- IDs must be provided
- Embeddings will be created by the collection!

Multiple documents

```
collection.add(
  ids=["my-doc-1", "my-doc-2"],
  documents=["This is document 1", "This is document 2"]
)
```

Inspecting a collection

Counting documents in a collection

```
collection.count()
```

```
3
```

Inspecting a collection

Peeking at the first 10 items

```
collection.peek()
```

```
{'ids': ['my-doc', 'my-doc-1', 'my-doc-2'],
 'embeddings': [[...], [...], [...]],
 'documents': ['This is the source text',
  'This is document 1',
  'This is document 2'],
 'metadatas': [None, None, None]}
```

Retrieving items

```
collection.get(ids=["s59"])
```

```
{'ids': ['s59'],
 'embeddings': None,
 'metadatas': [None],
 'documents': ['Title: Naruto Shippûden the Movie: The Will of Fire (Movie)\nDescription: When ...'],
 'uris': None,
 'data': None}
```

Netflix dataset

```
Title: Kota Factory (TV Show)
Description: In a city of coaching centers known to train India's finest...
Categories: International TV Shows, Romantic TV Shows, TV Comedies
```

```
Title: The Last Letter From Your Lover (Movie)
Description: After finding a trove of love letters from 1965, a reporter sets...
Categories: Dramas, Romantic Movies
```

Estimating embedding cost

- Embedding model (`text-embedding-3-small`) costs $0.00002/1k tokens

```
cost = 0.00002 * len(tokens)/1000
```

- Count tokens with the `tiktoken` library
  - `pip install tiktoken`

Estimating embedding cost

```
import tiktoken

enc = tiktoken.encoding_for_model("text-embedding-3-small")

total_tokens = sum(len(enc.encode(text)) for text in documents)

cost_per_1k_tokens = 0.00002


print('Total tokens:', total_tokens)
print('Cost:', cost_per_1k_tokens * total_tokens/1000)
```

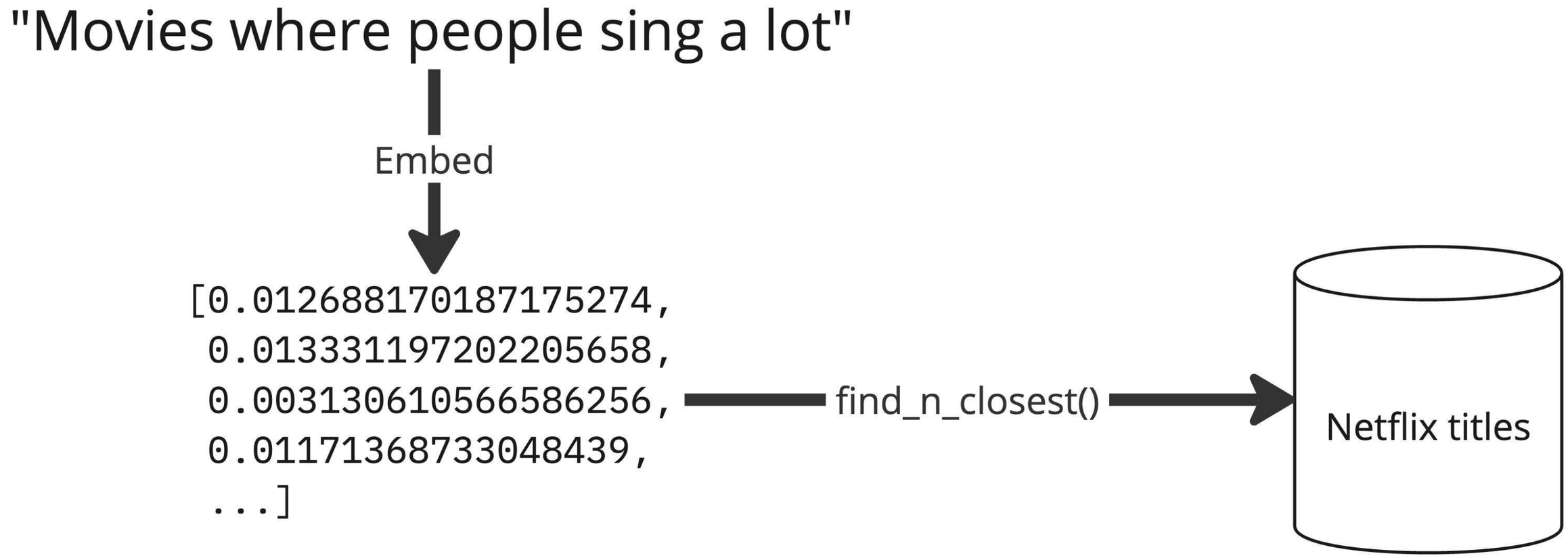
```
Total tokens: 444463
Cost: 0.00888926
```

TriveraTech
TECHNOLOGY TRAINING

Querying and updating the database

Experience is Everything

Querying the database



Querying the database

Retrieve the collection

```
from chromadb.utils.embedding_functions import OpenAIEmbeddingFunction

collection = client.get_collection(
    name="netflix_titles",
    embedding_function=OpenAIEmbeddingFunction(api_key="...")
)
```

- Must be specify *the same embedding function* used when adding data to the collection

Querying the collection

```
result = collection.query(
  query_texts=["movies where people sing a lot"],
  n_results=3
)
print(result)
```

```
{'ids': [['s4068', 's293', 's2213']],
 'embeddings': None,
 'documents': [['Title: Quién te cantará (Movie)\nDescription: When a near-...',
   'Title: Quartet (Movie)\nDescription: To save their posh retirement home, ...',
   'Title: Sing On! Spain (TV Show)\nDescription: In this fast-paced, high-...']],
 'metadatas': [[None, None, None]],
 'distances': [[0.350419282913208, 0.36049118638038635, 0.37080681324005127]]
```

`query()` returns a dict with multiple keys:

- `ids` : The ids of the returned items
- `embeddings` : The embeddings of the returned items
- `documents` : The source texts of the returned items
- `metadatas` : The metadatas of the returned items
- `distances` : The distances of the returned items from the query text

```
{'ids': [...],
 'embeddings': None,
 'documents': [...],
 'metadatas': [...],
 'distances': [...]}
```

```
'ids': [['s4068', 's293', 's2213']]
```

```
result = collection.query(
  query_texts=["movies where people sing a lot"],
  n_results=3
)
```

- First list corresponds to the first query_text
- Multiple query texts will return multiple lists

```
{'ids': [['s4068', 's293', 's2213']],

 'embeddings': None,

 'documents': [["Title: Quién te cantará (Movie)\nDescription: When a near-drowning leaves a famous singer from the '90s with amnesia, she hires a karaoke singer who can imitate her to prep her for a comeback tour.\nCategories: Dramas, Independent Movies, International Movies",

   'Title: Quartet (Movie)\nDescription: To save their posh retirement home, former opera stars plan a gala recital — until the biggest diva among them refuses to sing.\nCategories: Comedies, Dramas, Independent Movies',

   'Title: Sing On! Spain (TV Show)\nDescription: In this fast-paced, high-energy karaoke competition, singers from all walks of life battle it out for up to 30,000 euros!\nCategories: International TV Shows, Reality TV, Spanish-Language TV Shows']],

 'metadatas': [[None, None, None]],

 'distances': [[0.350419282913208, 0.36049118638038635, 0.37080681324005127]]}
```

Updating a collection

```
collection.update(
  ids=["id-1", "id-2"],
  documents=["New document 1", "New document 2"]
)
```

- Include *only* the fields to update, other fields will be unchanged
- Collection will automatically create embeddings

Upserting a collection

```
collection.upsert(
  ids=["id-1", "id-2"],
  documents=["New document 1", "New document 2"]
)
```

- If IDs are missing →add them
- If IDs are present →update them

Deleting

Delete items from a collection

```
collection.delete(ids=["id-1", "id-2"])
```

Delete all collections and items

```
client.reset()
```

- **Warning**: this will delete **everything** in the database!

TriveraTech
TECHNOLOGY TRAINING

Multiple queries and filtering

Experience is Everything

Movie recommendations based on multiple datapoints

- Terrifier (id: `'s8170'`)
- Strawberry Shortcake: Berry Bitty Adventures (id: `'s8103'`)

Multiple query texts

```
reference_ids = ['s8170', 's8103']

reference_texts = collection.get(ids=reference_ids)["documents"]


result = collection.query(

  query_texts=reference_texts,

  n_results=3

)
```

Multiple query texts result

```
{'ids': [['s8170', 's6939', 's7000'],['s8103', 's2968', 's3085']],
 'embeddings': None,
 'documents': [['Title: Terrifier (Movie)...',
   'Title: Haunters: The Art of the Scare (Movie)...',
   'Title: Horror Story (Movie)...'],
  ["Title: Strawberry Shortcake: Berry Bitty Adventures (TV Show)...",
   "Title: Shopkins (TV Show)...",
   "Title: Rainbow Ruby (TV Show)..."]],
 'metadatas': [[None, None, None], [None, None, None]],
 'distances': [[0.00, 0.25, 0.26], [0.00, 0.25, 0.28]]}
```

Adding metadata

```
import

  import csv


  ids = []  metadatas =
  []
with                                        as

  with open('netflix_titles.csv') as csvfile: reader =
    for                  in
      csv.DictReader(csvfile)
      for i, row in enumerate(reader):
          ids.append(row['show_id'])
          metadatas.append({
              "type":row['type'],
              "release_year": int(row['release_year']
          })
```

- Create a list of dicts for the metadatas
- Create a list of IDs to add them to the existing items

Adding and querying metadatas

```
collection.update(ids=ids, metadatas=metadatas)
```

```
result = collection.query(
    query_texts=reference_texts,
    n_results=3,
    where={
        "type": "Movie"
    }
)
```

Where operators

```
where={
  "type": "Movie"
}
```

is the same as

```
where={
  "type": {
    "$eq": "Movie"
  }
}
```

List of operators:

- `$eq` - equal to (string, int, float)
- `$ne` - not equal to (string, int, float)
- `$gt` - greater than (int, float)
- `$gte` - greater than or equal to (int, float)
- `$lt` - less than (int, float)
- `$lte` - less than or equal to (int, float)

Multiple where filters

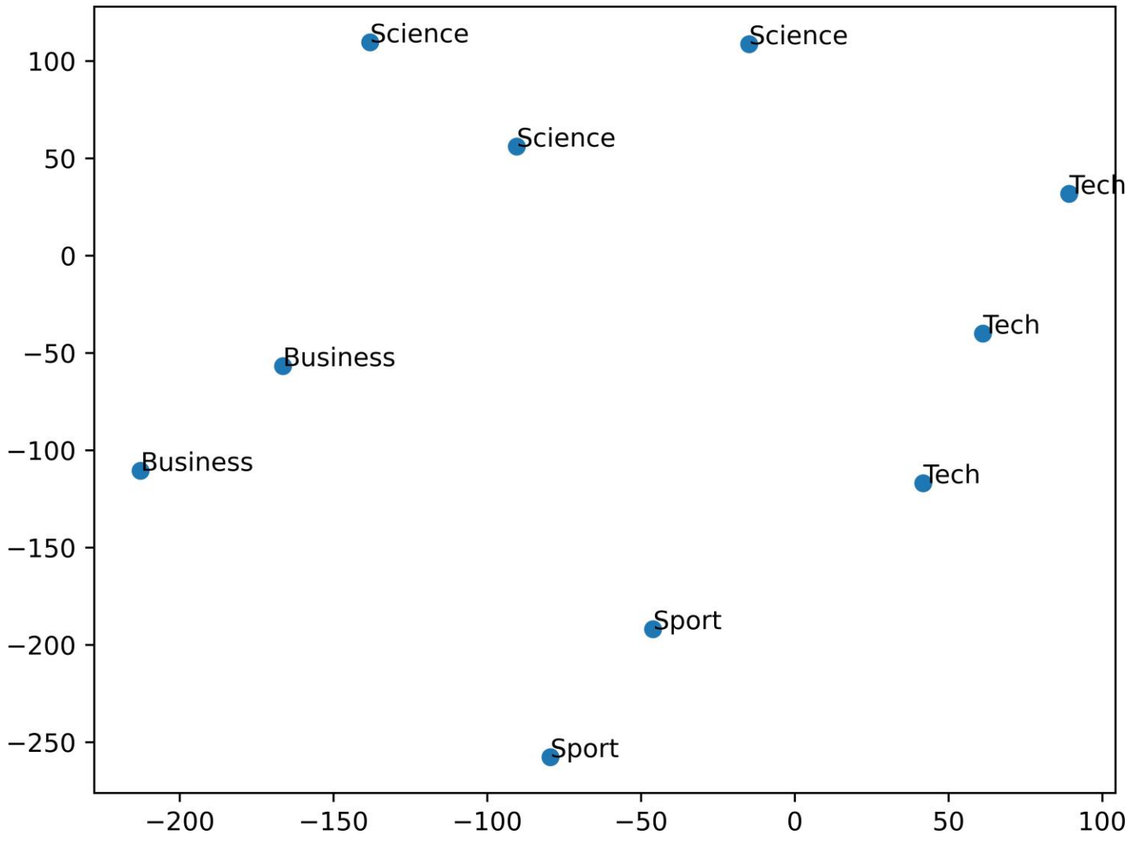
```
where={
    "$and": [
        {"type":
            {"$eq": "Movie"}
        },
        {"release_year":
             {"$gt": 2020}
        }
    ]
}
```

- `$or` : filter based on *at least* one condition

```
Title: A Classic Horror Story (Movie) [...]
===
Title: Nightbooks (Movie) [...]
===
Title: Irul (Movie) [...]
===
Title: Intrusion (Movie) [...]
===
Title: Things Heard & Seen (Movie) [...]
===
Title: A StoryBots Space Adventure (Movie) [...]
```

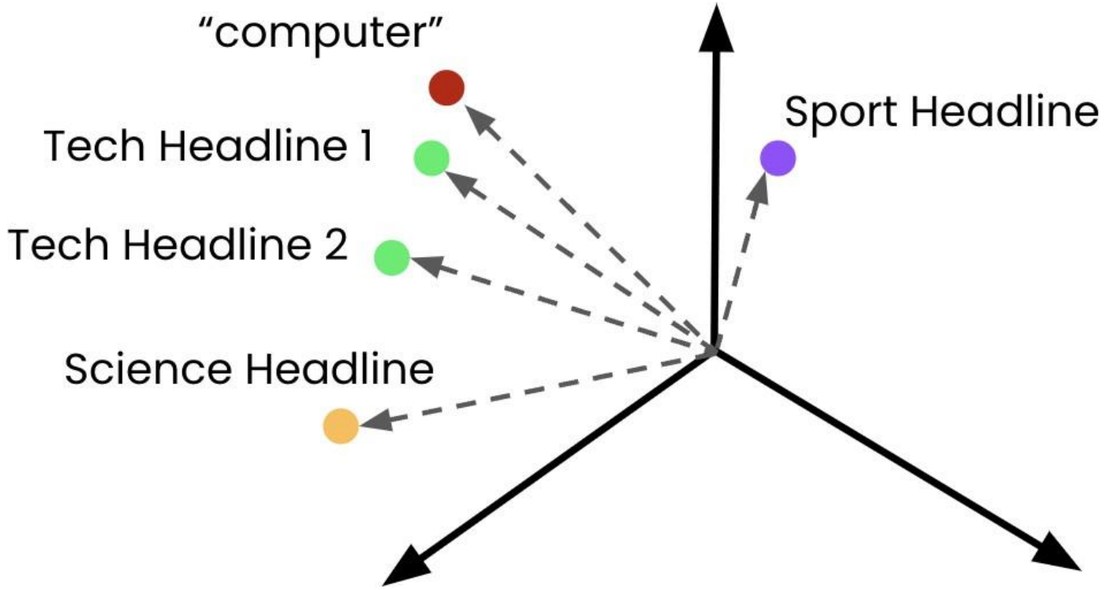
Review 1- What are Embeddings?

- **Embeddings**: *vector*/numerical representation of text
- Capture the *semantic meaning* of text
- Used OpenAI's Embedding model
- Can use the *cosine distance* to find similar texts
- Unlocks semantic search, recommendation engines, etc.

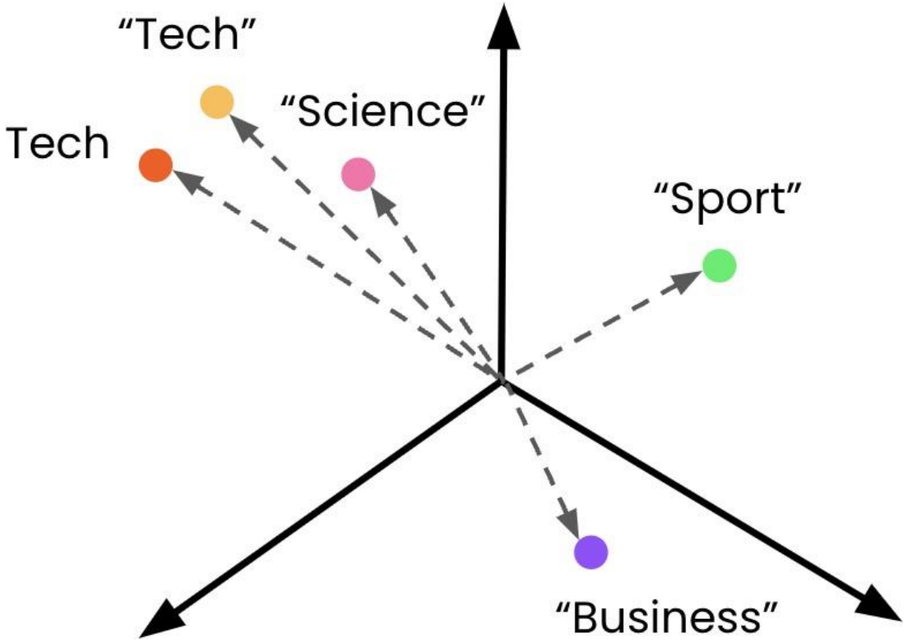


Review 2 - Embeddings for AI Applications

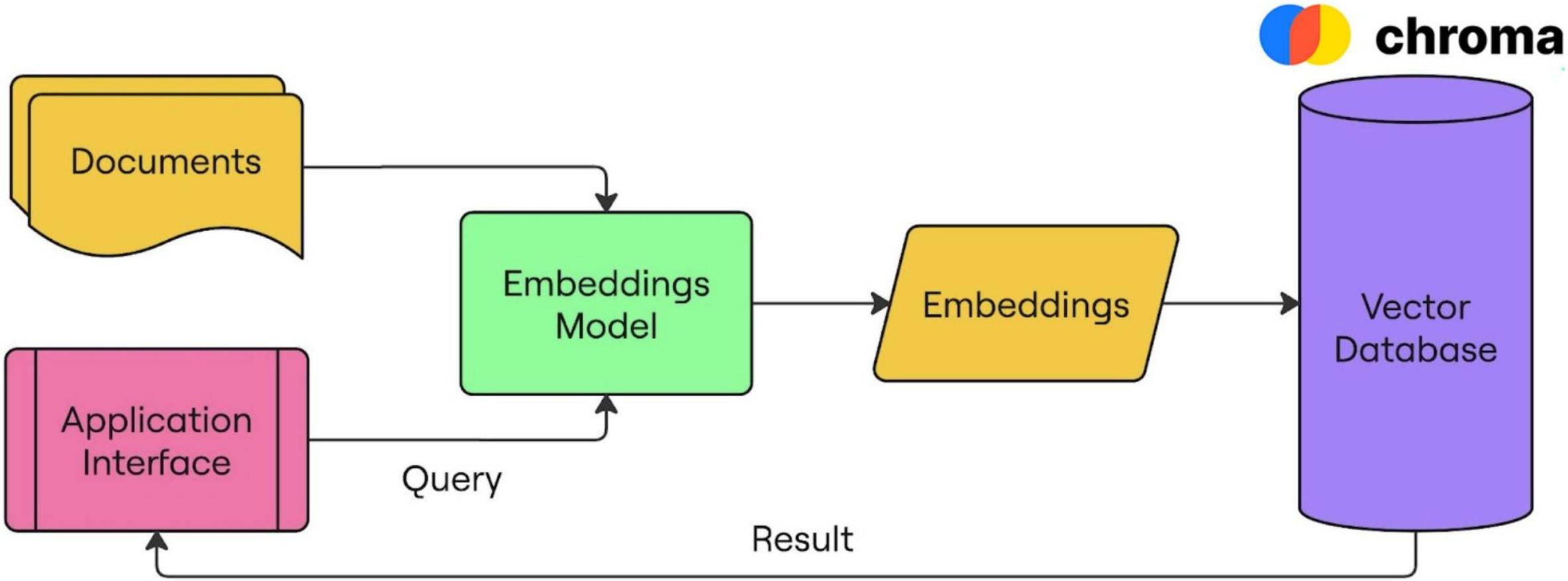
Semantic search and recommendation



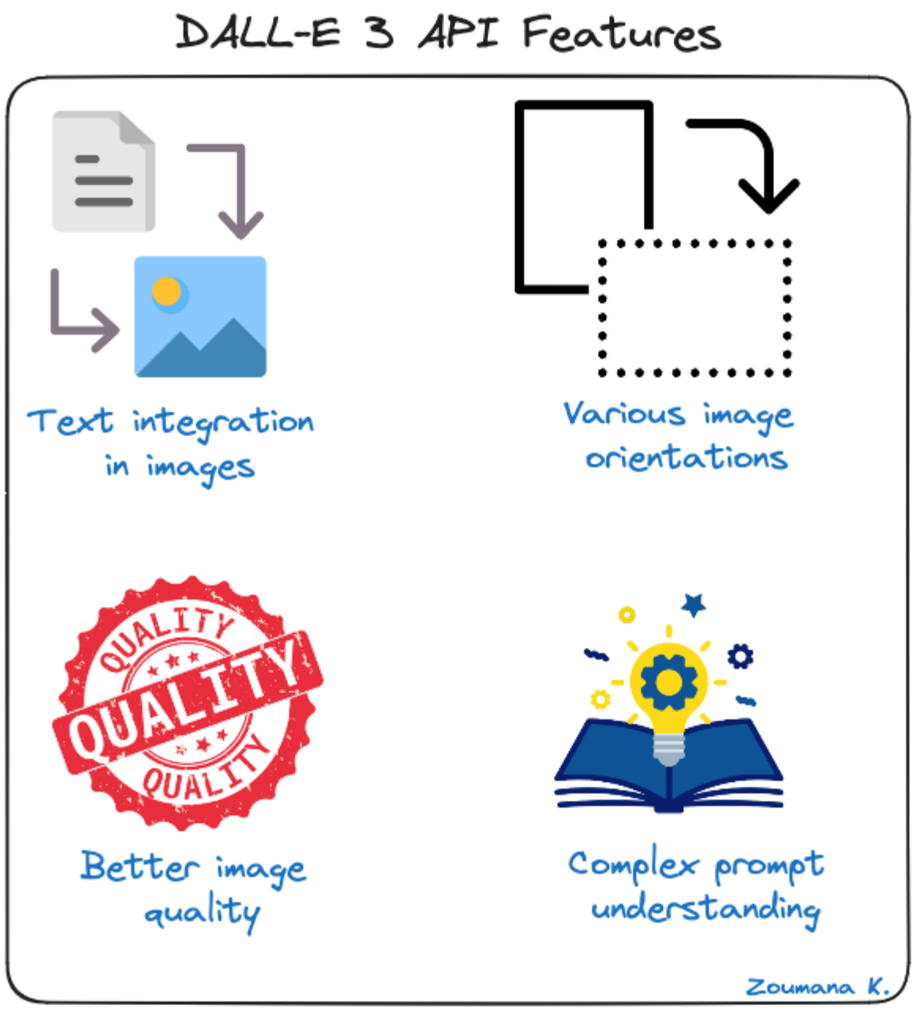
Classification

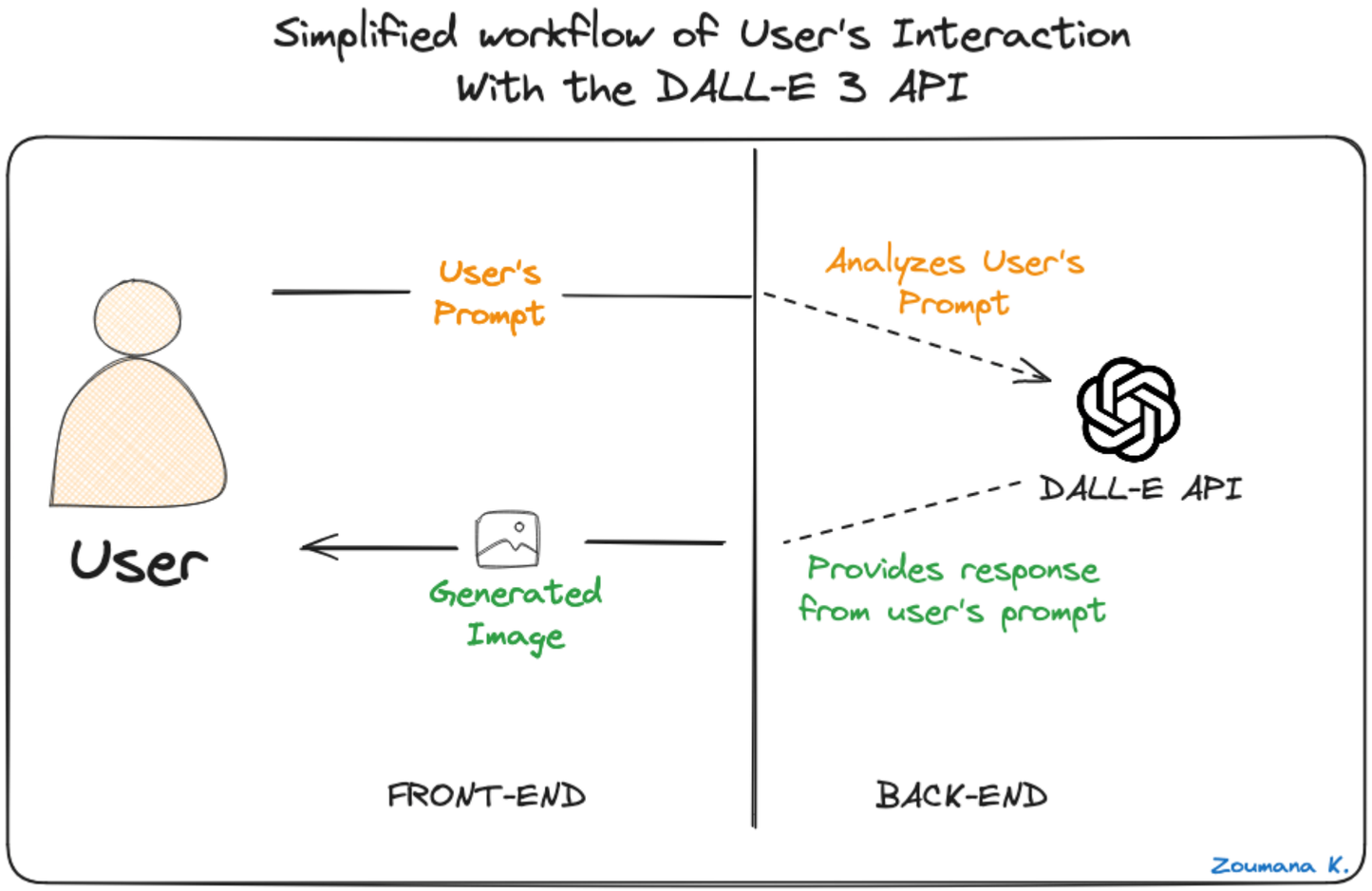


Review 3 - Vector Databases

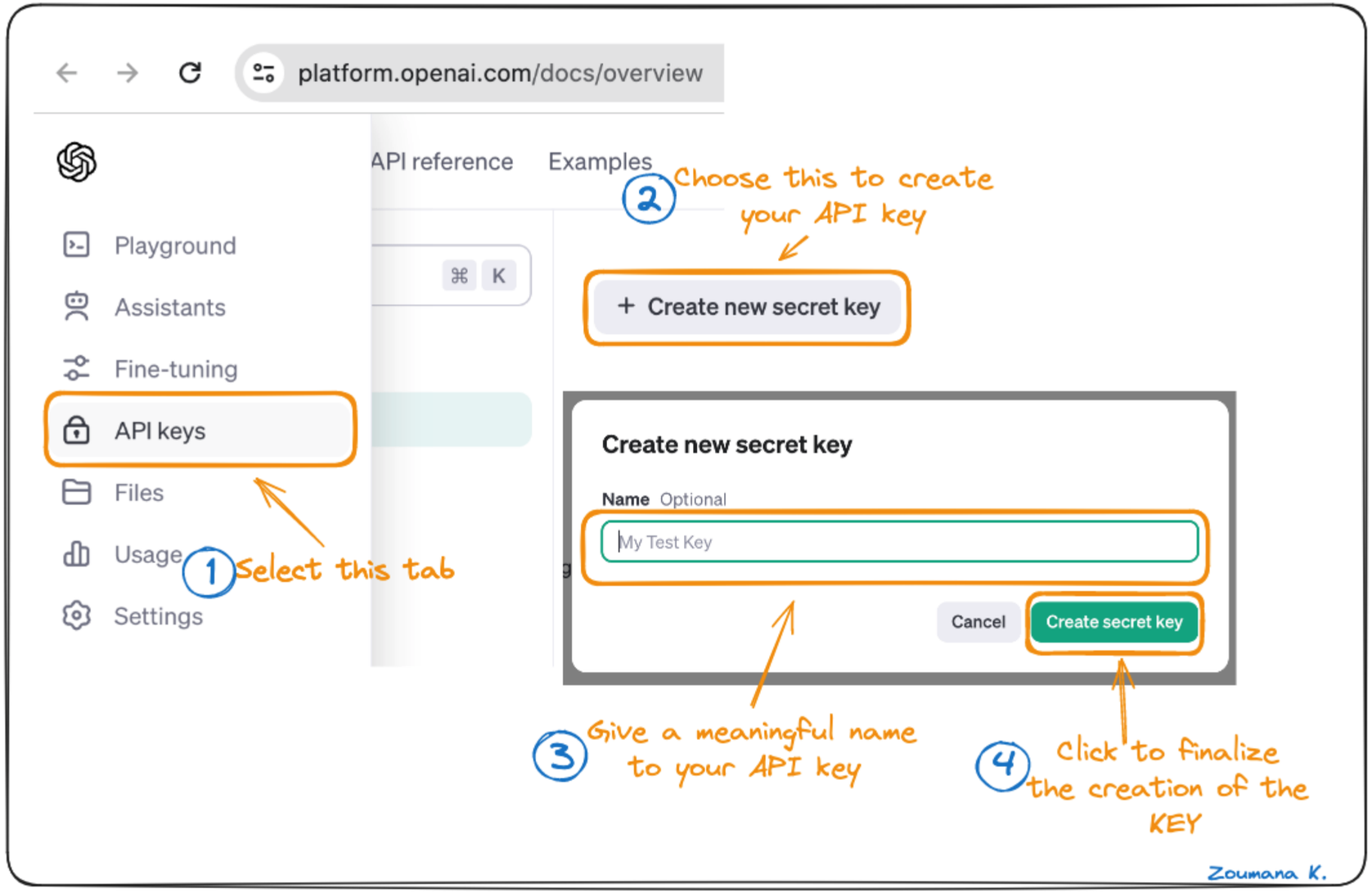


Dall-E2


DALL-E 3 API Features
Text integration in images
Various image orientations
QUALITY
QUALITY
QUALITY
Better image quality
Complex prompt understanding
Zoumana K.


Simplified workflow of User's Interaction
With the DALL-E 3 API
User's Prompt
Analyzes User's Prompt
DALL-E API
User
Generated Image
Provides response from user's prompt
FRONT-END
BACK-END
Zoumana K.

4 Main Steps to Create an OpenAI KEY



```python
from openai import OpenAI


# Instantiate the OpenAI client
client = OpenAI()
from IPython.display import Image
```

```python
def get_image_from_DALL_E_3_API(user_prompt,
                                image_dimension="1024x1024",
                                image_quality="hd",
                                model="dall-e-3",
                                nb_final_image=1):
   response = client.images.generate(
     model = model,
     prompt = user_prompt,
     size = image_dimension,
     quality = image_quality,
     n=nb_final_image,
   )


   image_url = response.data[0].url

   display(Image(url=image_url))
```

Project

Experience is Everything

Thank You for joining us!

We truly appreciate your time. Please complete the **End of Course Survey**.

Any questions?

- Review the full Course Guide for Course Tips, Resources & Next Step Learning Plans
- Feel Free to Reach Out: Info@triveratech.com
- See our full list of Courses & SkillJourneys: www.triveratech.com

Free Courses, Articles, Resources & Offers

- Follow Us for Free Courses, Articles, Resources & Offers: **LinkedIn: @TriveraTech**
- Subscribe to our Channel for Free Courses & Events: **YouTube: @TriveraTech**