AI BOOTCAMP

Gustavo Inostroza Ruiz
Head of Data Science

EL CLUB DE LA IA

wingsoft

Microsoft

Ocular

Agenda

01

Python Basics

Variables
Listas
Ciclos
Funciones

02

OpenAI y LangChain

LLM
Memoria
Tools
Agente

03

AI Tools

Voice-Text
Text-Speech
Text-Image
Image-Video
Llama2

OPENAI Y LANGCHAIN

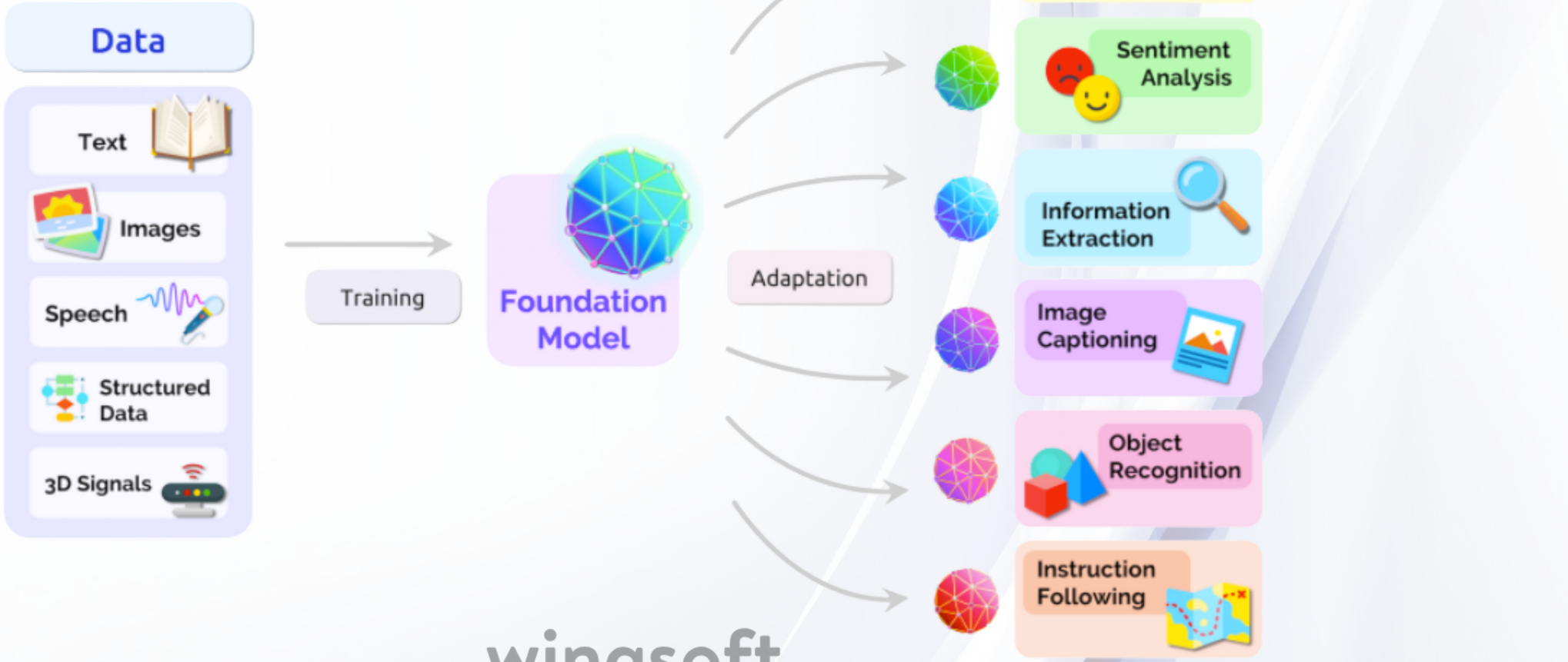
Let’s Go!

github.com
inostroza7/OpenAI-LangChain-Basics

wingsoft

OPENAI Y LANGCHAIN

Large Language Model



OPENAI Y LANGCHAIN

Large Language Model

wingsoft

OPENAI Y LANGCHAIN

OpenAI

```
pip install openai
```

```
import openai
```

```
openai.api_key = 'sk-xx'
```

OPENAI Y LANGCHAIN

OpenAI

prompt

- Persona
- Contexto
- Tarea
- Ejemplo
- Formato
- Tono

model

gpt-3.5-turbo

gpt-4

max_tokens

role

system

user

temperature

0 —●— 1

OPENAI Y LANGCHAIN

OpenAI

```
prompt = 'Hola, Chatbot'
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": prompt},],
    temperature = 0.2)

print(response)
```

OPENAI Y LANGCHAIN

OpenAI


```
{
  "id": "chatcmpl-7uv03ZD6IzBm5LCoDzBN9Ts0frj4F",
  "object": "chat.completion",
  "created": 1693799831,
  "model": "gpt-3.5-turbo-0613",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "\u00a1Hola! \u00bfEn qu\u00e9 puedo ayudarte hoy?"
      },
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 20,
    "completion_tokens": 11,
    "total_tokens": 31
  }
}
```

OPENAI Y LANGCHAIN

OpenAI

```
response.choices[0].message['content']
```

✓ 0.0s

```
'¡Hola! ¿En qué puedo ayudarte hoy?'
```

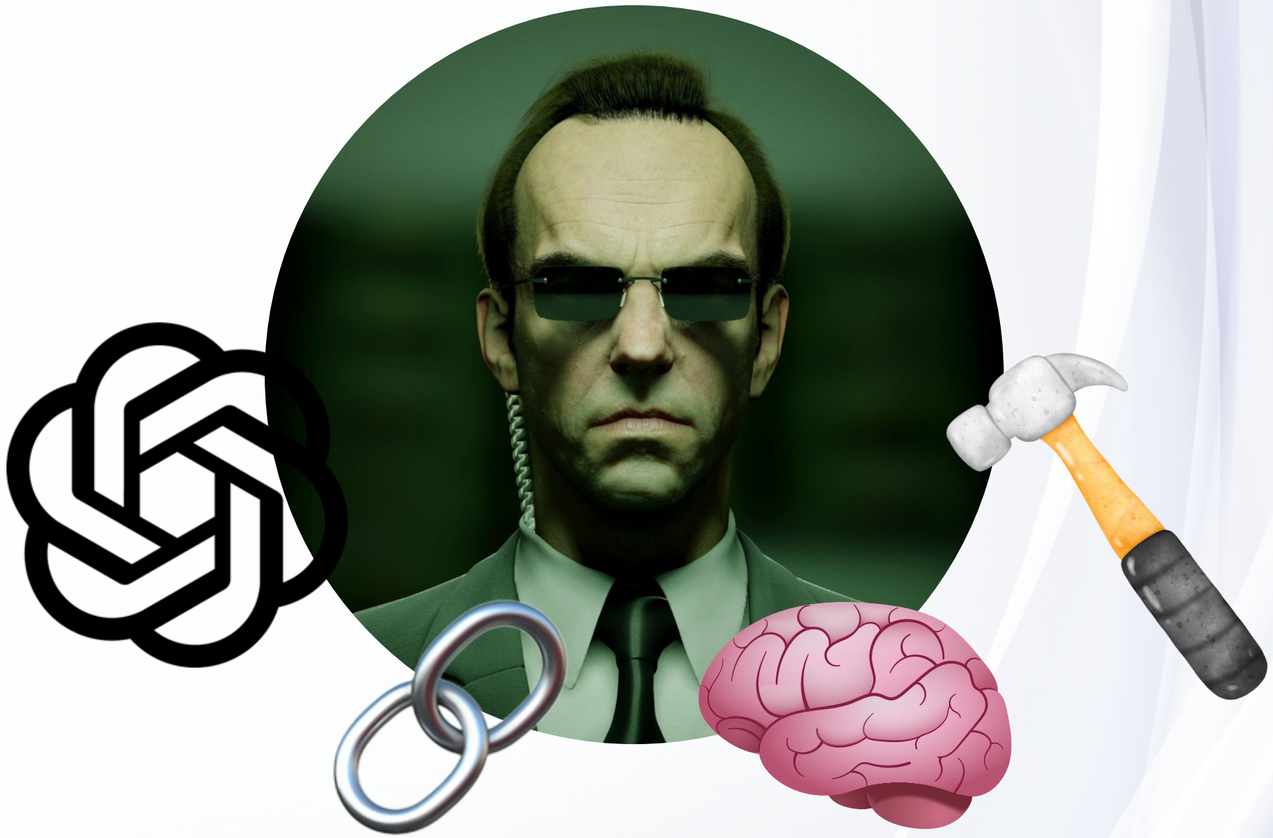
OPENAI Y LANGCHAIN

OpenAI

wingsoft

OPENAI Y LANGCHAIN

LangChain



wingsoft

OPENAI Y LANGCHAIN

LLM

```
pip install langchain

from langchain.llms import OpenAI
llm = OpenAI(temperature=0.3)

prompt = 'Hola cómo estás?'
print(llm(prompt))
```

OPENAI Y LANGCHAIN

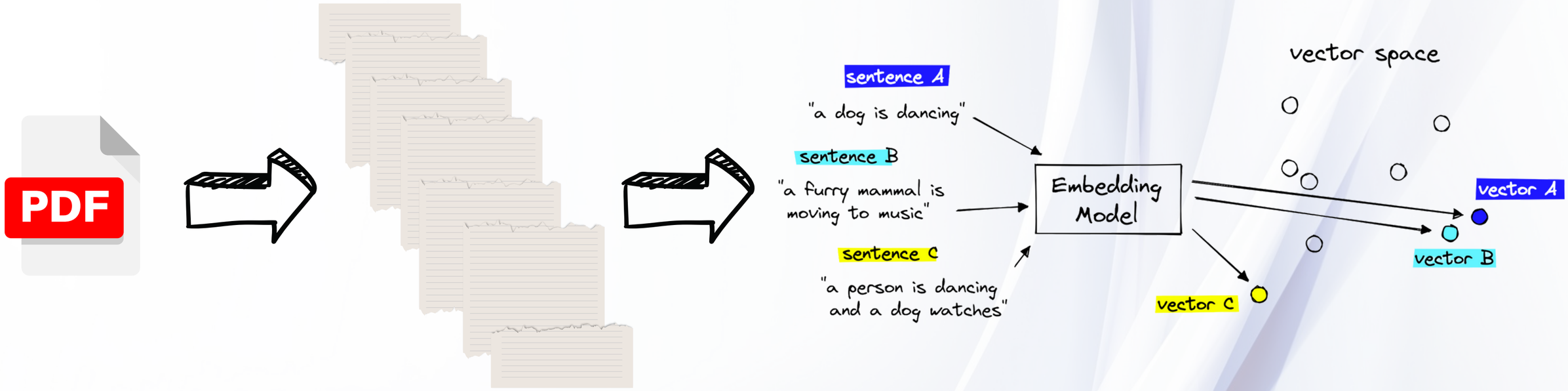
Chat Models

```
from langchain.chat_models import ChatOpenAI
from langchain.schema import SystemMessage, HumanMessage

prompt = 'Hola cómo estás?'
chat = ChatOpenAI(model='gpt-3.5-turbo',temperature=0.3)
messages = [
    SystemMessage(content='You are a helpful assistant'),
    HumanMessage(content=prompt)
]
response = chat(messages)
print(response)
```

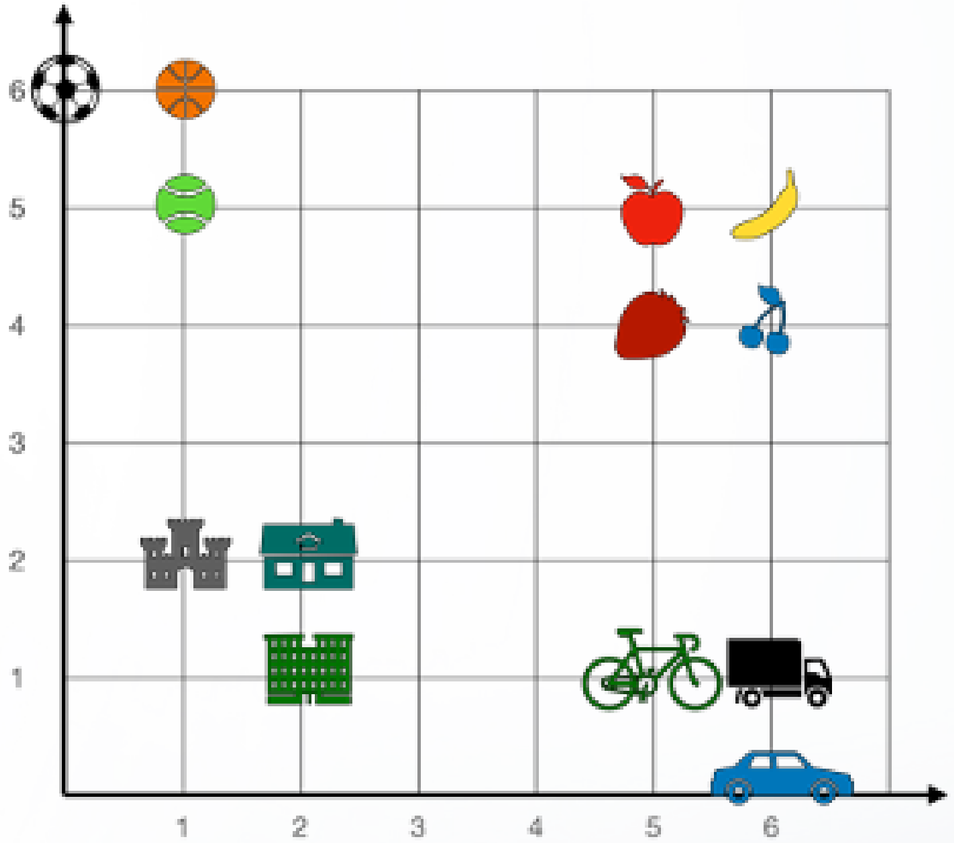
OPENAI Y LANGCHAIN

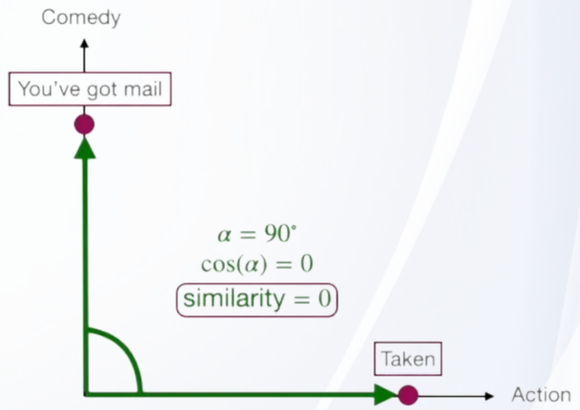
Chunks



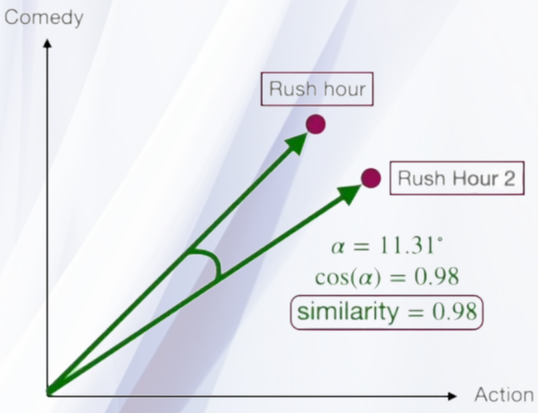
OPENAI Y LANGCHAIN

Embeddings





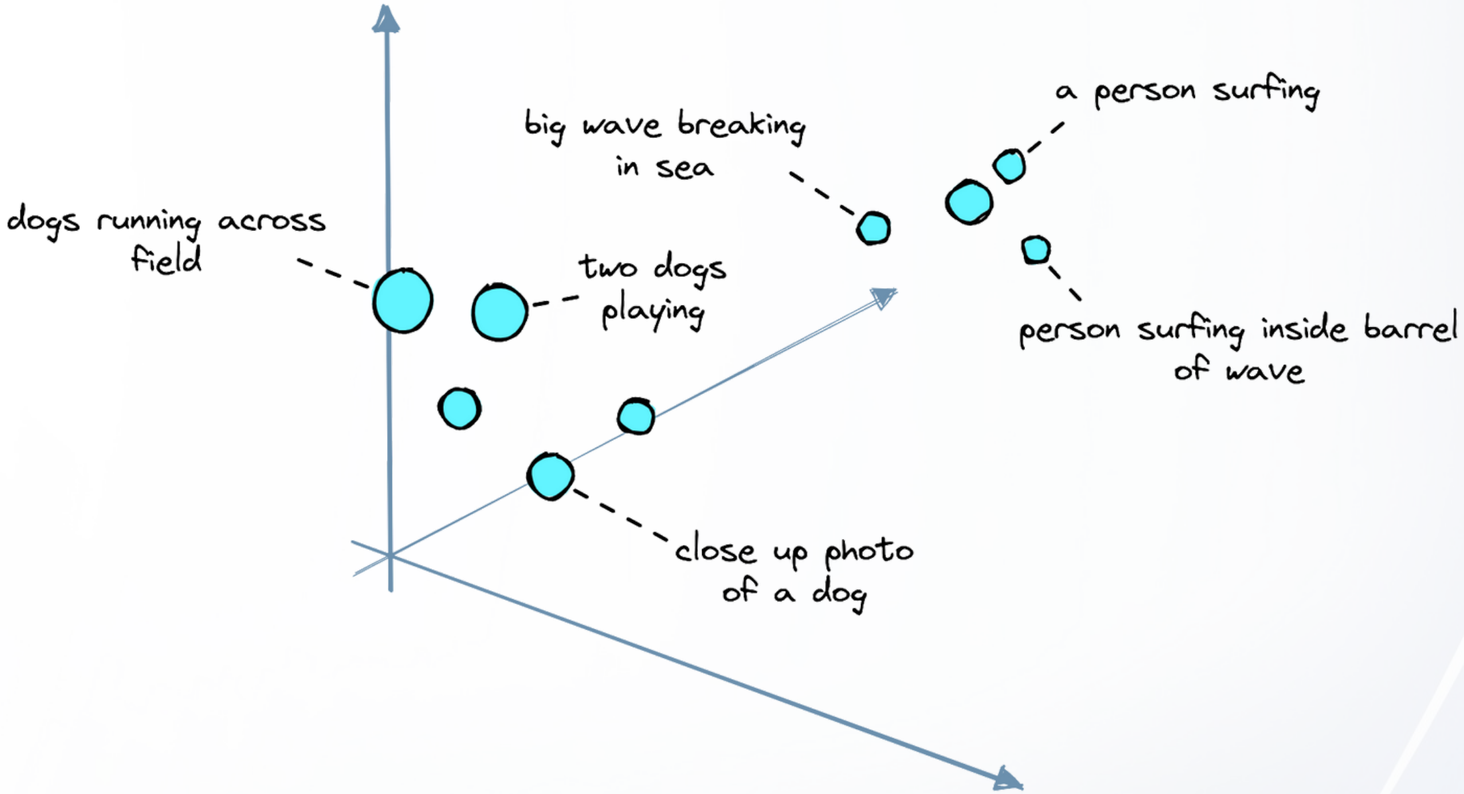
Small similarity



Large similarity

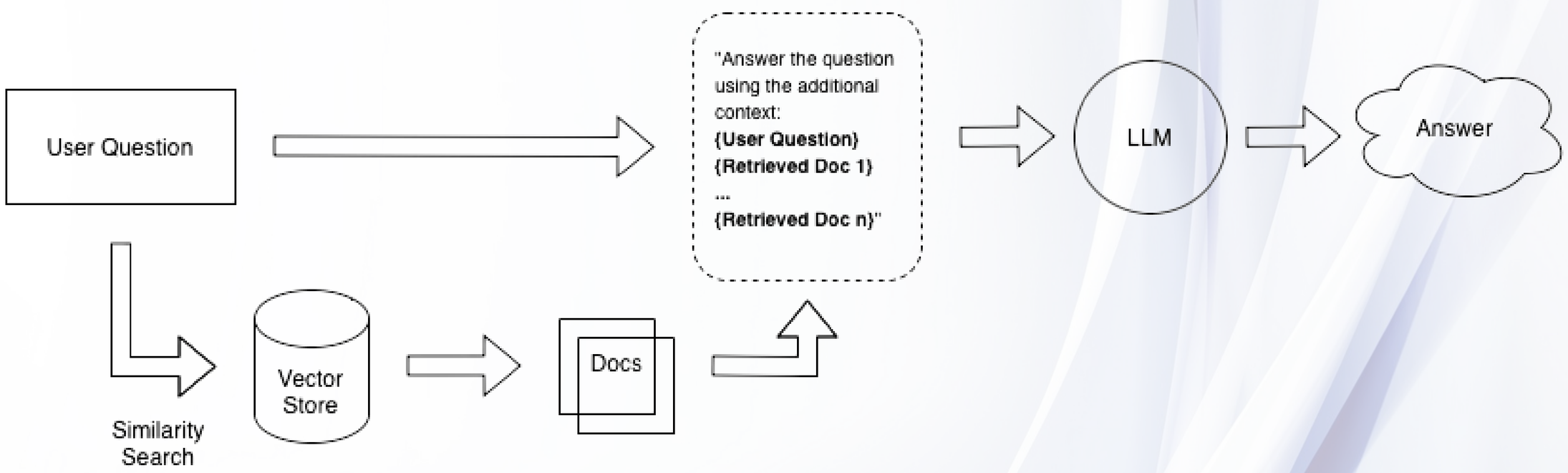
OPENAI Y LANGCHAIN

Vector Store



OPENAI Y LANGCHAIN

Retrievals



OPENAI Y LANGCHAIN

Retrievals

OPENAI Y LANGCHAIN

Retrievals

```python
def load_document(file):
    import os
    name, extension = os.path.splitext(file)

    if extension == '.pdf':
        from langchain.document_loaders import PyPDFLoader
        print(f'Loading {file}')
        loader = PyPDFLoader(file)
    elif extension == '.docx':
        from langchain.document_loaders import Docx2txtLoader
        print(f'Loading {file}')
        loader = Docx2txtLoader(file)
    elif extension == '.txt':
        from langchain.document_loaders import TextLoader
        loader = TextLoader(file)
    else:
        print('El formato del archivo no es válido')
        return None

    data = loader.load()
    return data
```

OPENAI Y LANGCHAIN

Retrievals

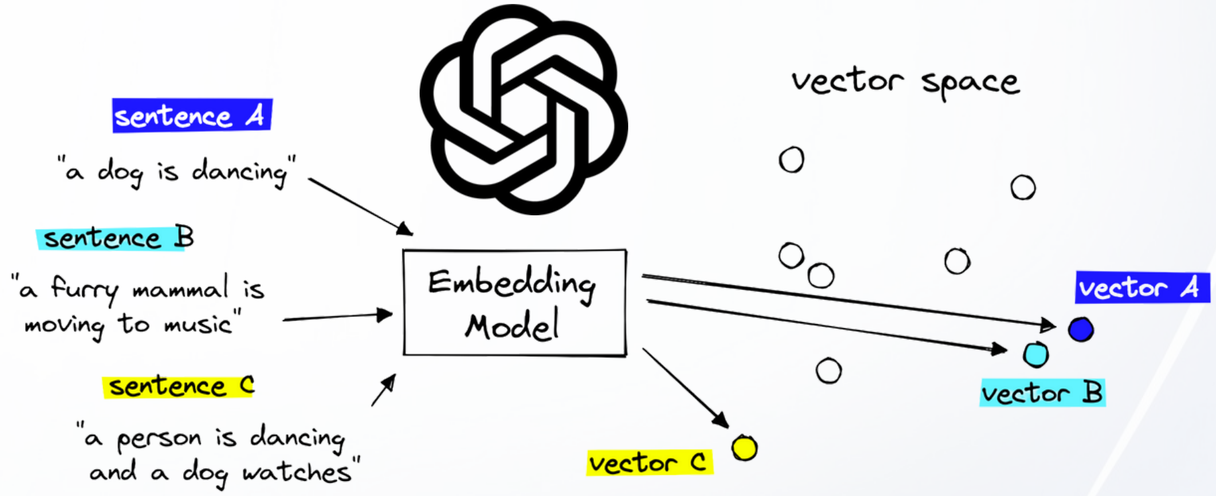
```python
def chunk_data(data, chunk_size=500, chunk_overlap=0):
    from langchain.text_splitter import RecursiveCharacterTextSplitter
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)
    chunks = text_splitter.split_documents(data)
    return chunks
```

OPENAI Y LANGCHAIN

Retrievals

```python
def create_embeddings(chunks):
    from langchain.embeddings.openai import OpenAIEmbeddings
    embeddings = OpenAIEmbeddings()

    from langchain.vectorstores import Chroma
    vector_store = Chroma.from_documents(chunks, embeddings)
    return vector_store
```



OPENAI Y LANGCHAIN

Retrievals

```python
def ask_and_get_answer(vector_store, q, k=3):
    from langchain.chains import RetrievalQA
    from langchain.chat_models import ChatOpenAI

    llm = ChatOpenAI(model='gpt-3.5-turbo', temperature=0.2)
    retriever = vector_store.as_retriever(search_type='similarity', search_kwargs={'k': k})
    chain = RetrievalQA.from_chain_type(llm=llm, chain_type="stuff", retriever=retriever)

    answer = chain.run(q)
    return answer
```

OPENAI Y LANGCHAIN

Memoria

With conversational memory

I'm interested in integrating LLMs with external knowledge.

LLMs are great at generating human-like text. Yet, integrating external knowledge can enhance their capabilities even more.

What are the different possible methods for doing this?

You could use pre-existing knowledge graphs, allow LLMs access to tools like APIs, or retrieval augmentation with vector DBs!

Conversation History

Interesting! What was it I wanted to know about again?

You were interested in integrating LLMs with external knowledge.

Without conversational memory

(No conversation history is stored)

Conversation History

Interesting! What was it I wanted to know about again?

Sorry I have no idea what you're talking about!

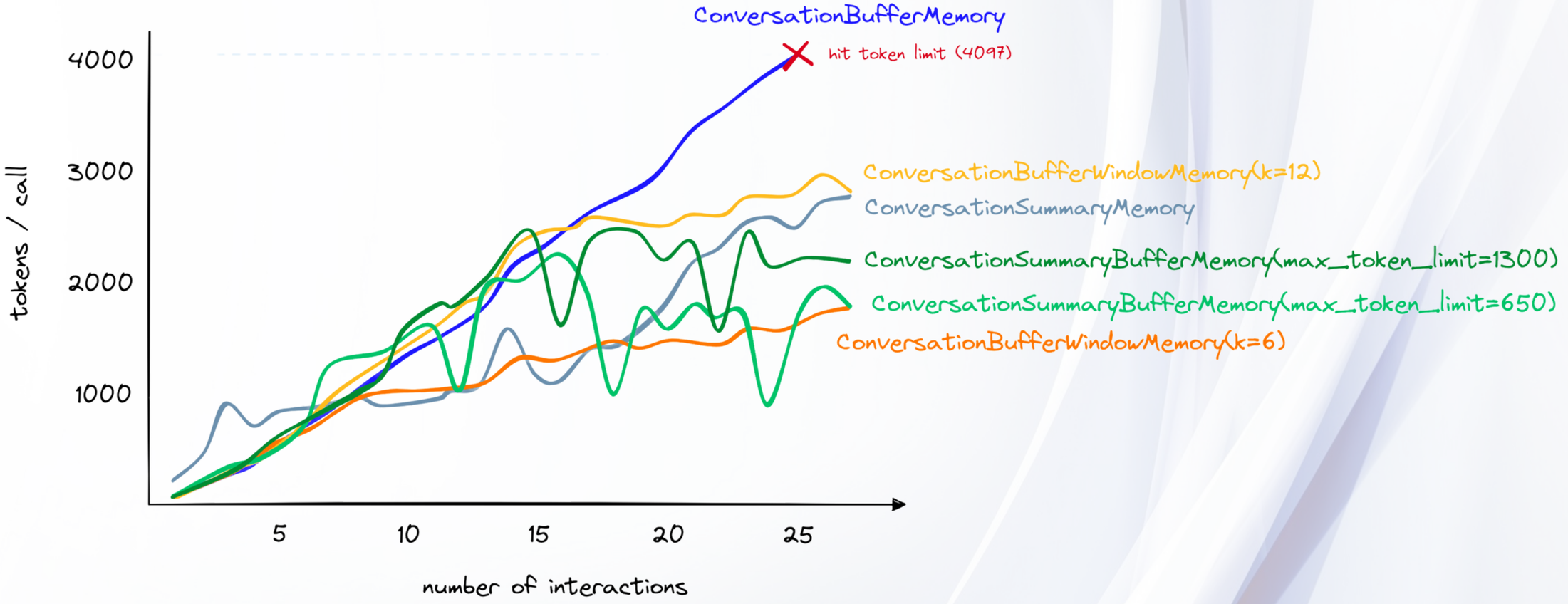
OPENAI Y LANGCHAIN

Memoria

- ConversationBufferMemory (All)
- ConversationSummaryMemory (Summary)
- ConversationBufferWindowMemory (k)
- ConversationSummaryBufferMemory (MaxTokens)
- Vector store-backed memory*

OPENAI Y LANGCHAIN

Memoria



OPENAI Y LANGCHAIN

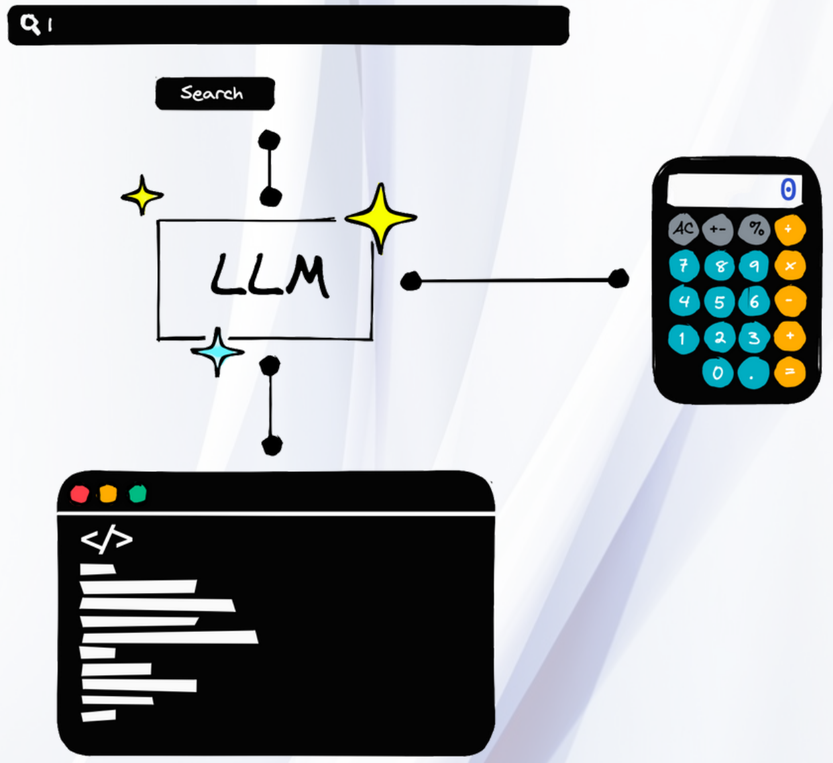
Memoria

- ConversationBufferMemory
- ConversationBufferWindowMemory (k)

OPENAI Y LANGCHAIN

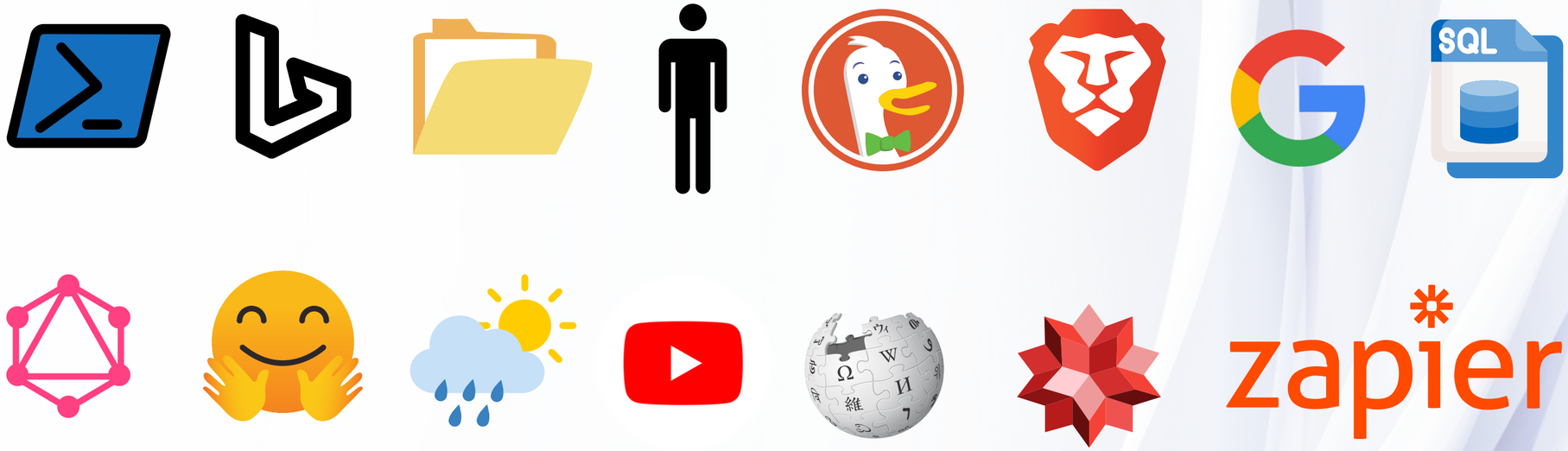
Tools

USER	What is the answer to 4.1^2.1?
ASSISTANT	The answer to 4.1^2.1 is approximately 17.817.



OPENAI Y LANGCHAIN

Tools



https://integrations.langchain.com/tools

OPENAI Y LANGCHAIN

Tools

```python
from langchain.chains import LLMMathChain
from langchain.agents import Tool

llm_math = LLMMathChain(llm=llm)

math_tool = Tool(
    name='Calculator',
    func=llm_math.run,
    description='Useful for when you need to answer questions about math.'
)

tools = [math_tool]
```

OPENAI Y LANGCHAIN

Tipos de Agentes

- Zero-shot ReAct
- Structured input ReAct
- OpenAI Functions
- Conversational
- Self-ask with search
- ReAct document store

OPENAI Y LANGCHAIN

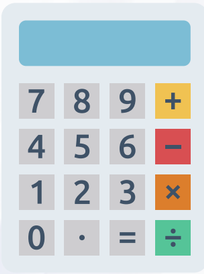
Agentes

```python
from langchain.agents import initialize_agent

zero_shot_agent = initialize_agent(
    agent="zero-shot-react-description",
    tools=tools,
    llm=llm,
    verbose=True,
    max_iterations=3
)
```

OPENAI Y LANGCHAIN

Agents



AI BOOTCAMP

Gustavo Inostroza Ruiz
Head of Data Science

EL CLUB DE LA IA

wingsoft

Microsoft

Ocular