






Python + IA

Python + IA

-  3/11: LLMs
- ↶ 3/13: Vector embeddings
-  3/18: RAG
-  3/19: Modelos de Vision
-  3/25: Salidas estructuradas
-  3/27: Calidad y Seguridad

Regístrate @ aka.ms/PythonIA/series





Python + IA



Modelos de Lenguaje Grande

Gwyneth Peña-Siguenza

Python Cloud Advocate

madebygps.com/about

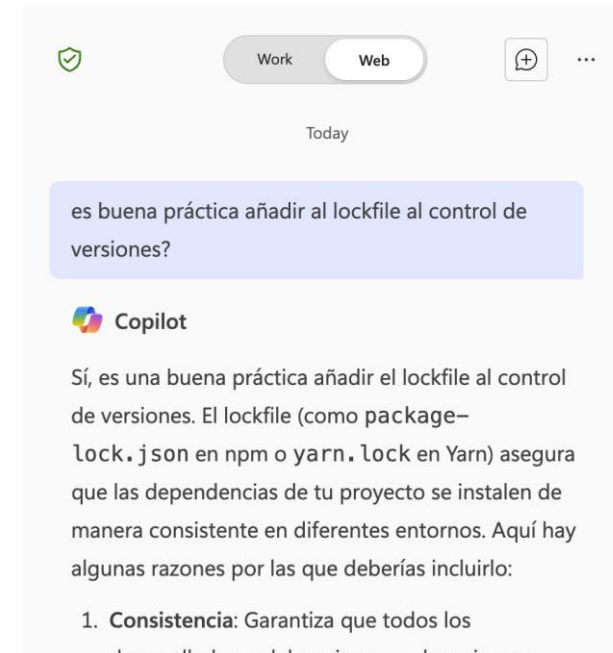
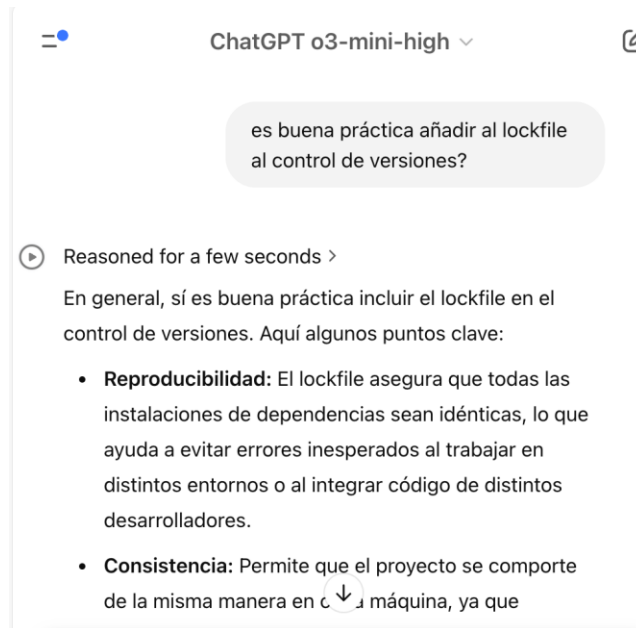
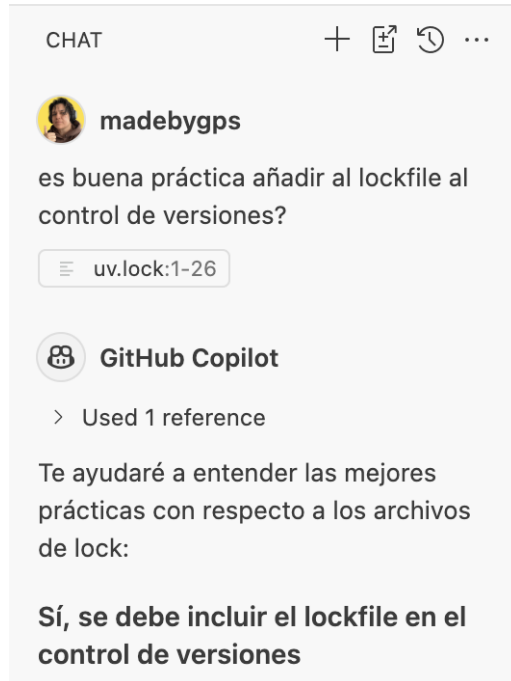
@madebygps

Hoy cubriremos...

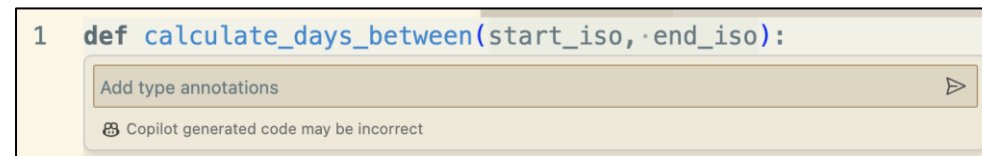
- Cómo funcionan los LLM
- Opciones de modelo
- Uso de los LLM desde Python
- Mejora de los resultados del LLM
- Librerías LLM
- Creación de aplicaciones basadas en LLM

Cómo funcionan los LLM

De seguro que has usado un LLM...



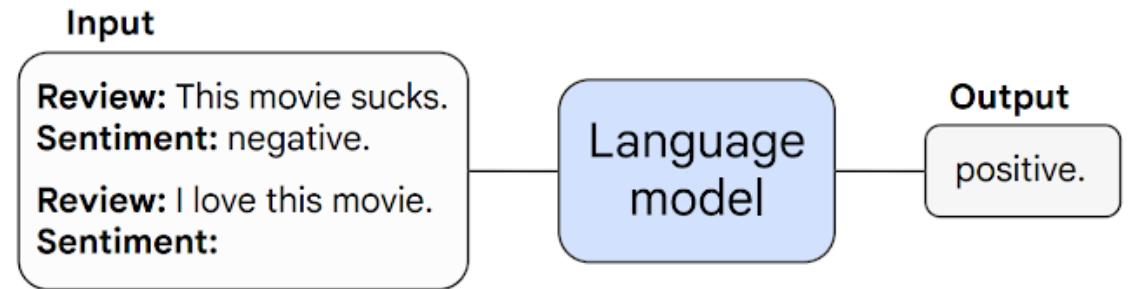
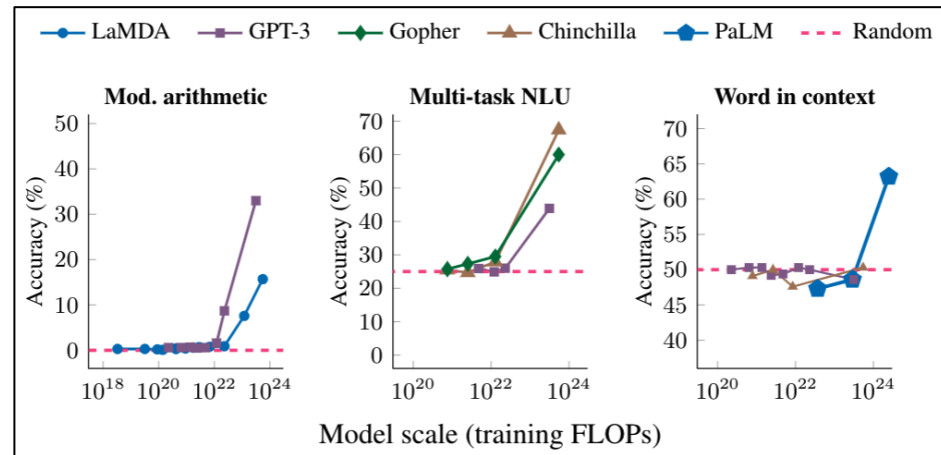
```
def calculate_days_between(start_iso, end_iso):
1 def calculate_days_between(start_iso, end_iso):
    from datetime import datetime
    start = datetime.strptime(start_iso, "%Y-%m-%d")
    end = datetime.strptime(end_iso, "%Y-%m-%d")
    return (end - start).days
```



ChatGPT, GitHub Copilot, Bing Copilot y muchos otros productos **funcionan** con LLM.

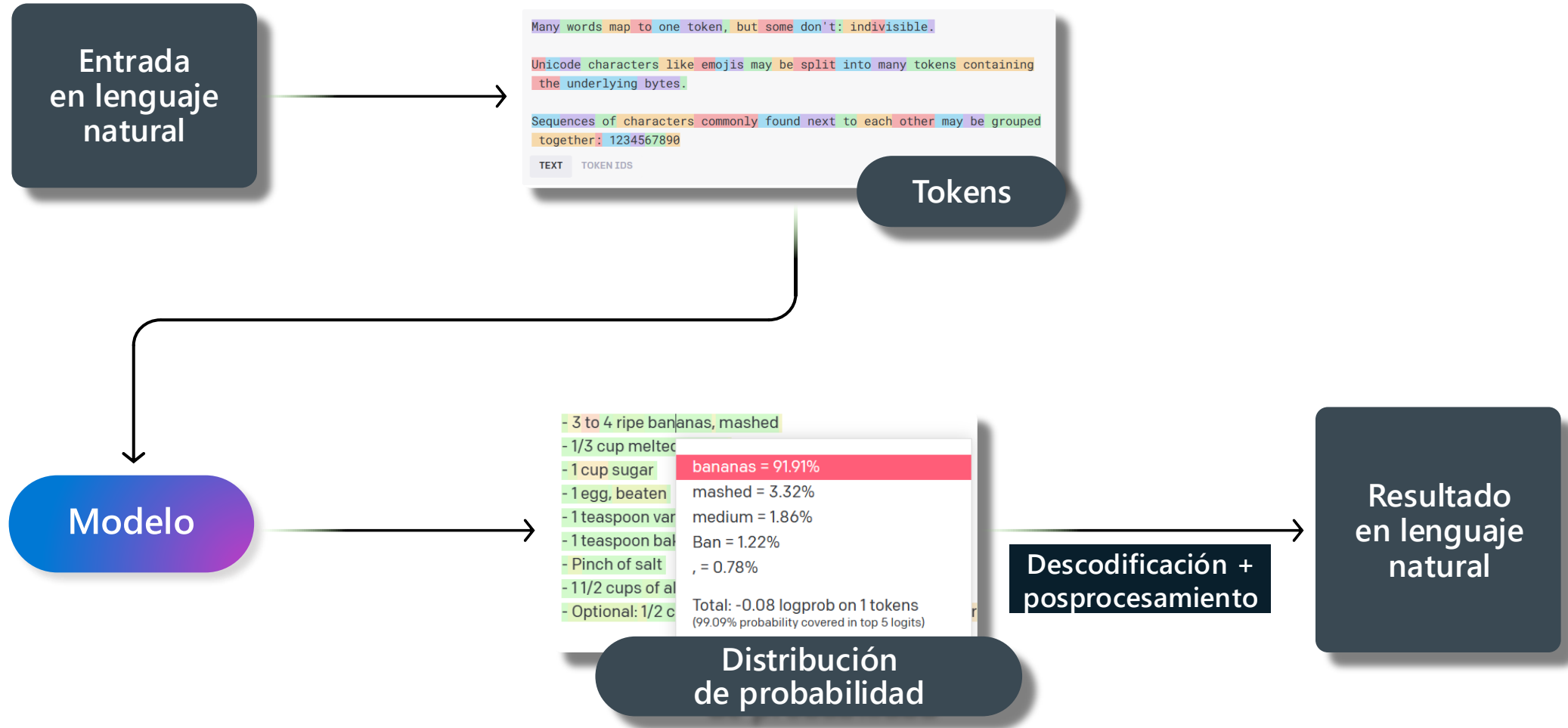
LLM: Modelos de Lenguaje Grande

Un LLM es un modelo de machine learning tan grande que logra comprender y generar lenguaje de uso cotidiana



[Characterizing Emergent Phenomena in LLMs](#)

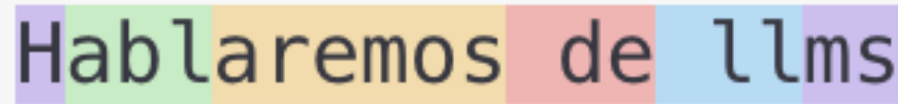
Así funcionan los modelos lingüísticos



Preprocesamiento de la entrada: Tokenizaton

Input: “Hablaaremos de llms”

Tokens:



Token IDs:

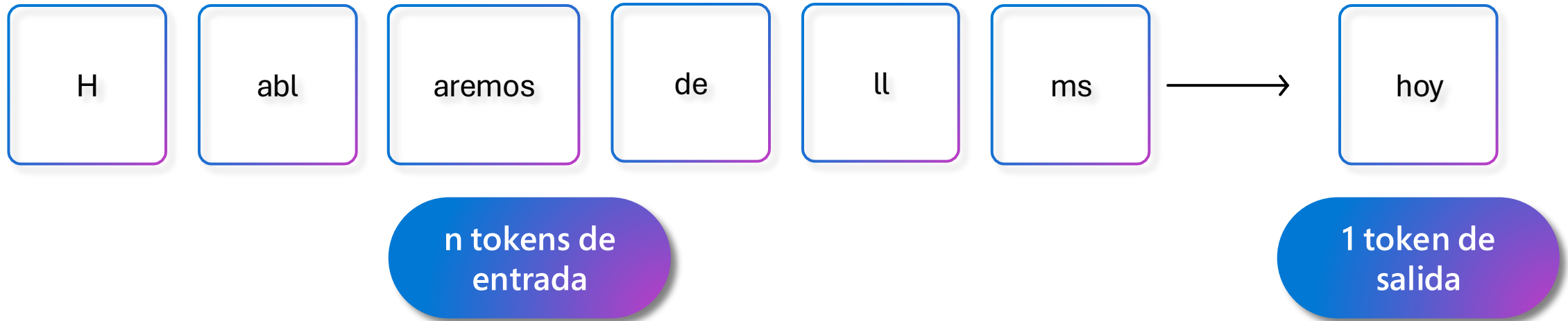
39 18122 32127

334

11475 1782

platform.openai.com/tokenizer

Entradas y salidas del modelo



El contexto se expande

Hablaremos de los llms hoy

Hablaremos de los llms hoy en

Hablaremos de los llms hoy en la

Hablaremos de los llms hoy en la clase

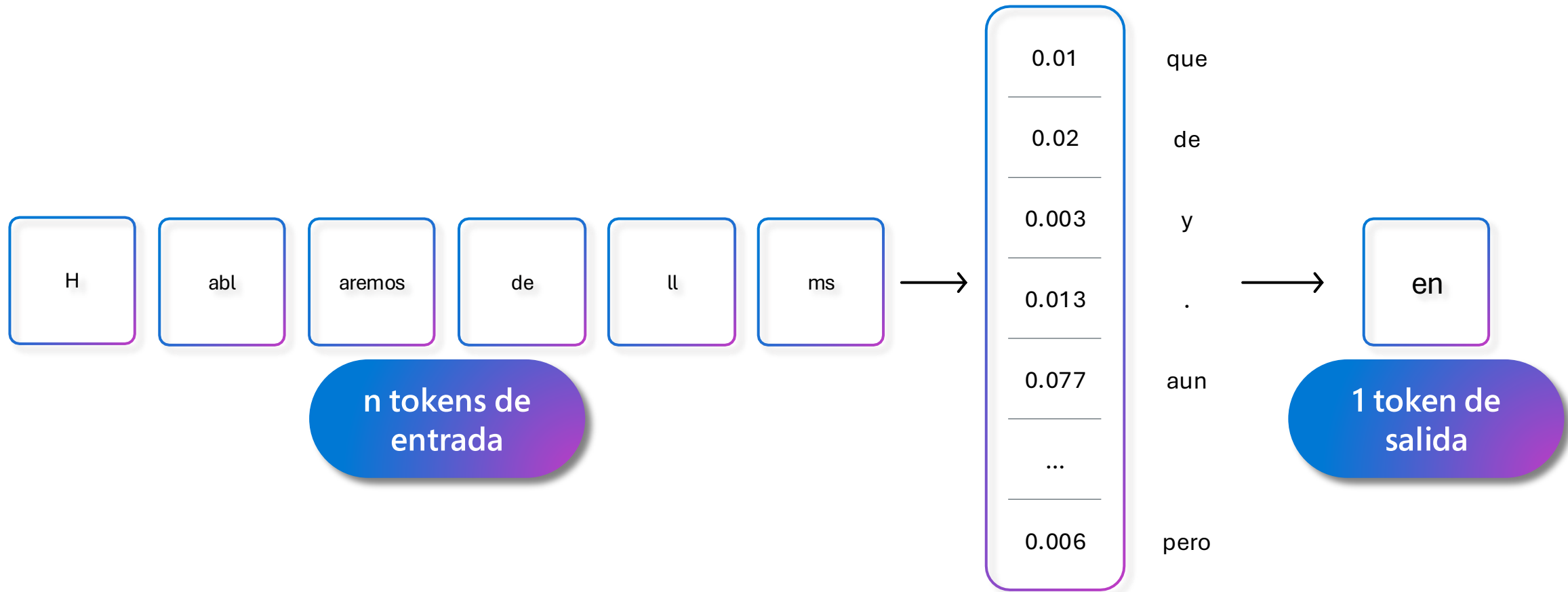
Hablaremos de los llms hoy en la clase de

Hablaremos de los llms hoy en la clase de programación

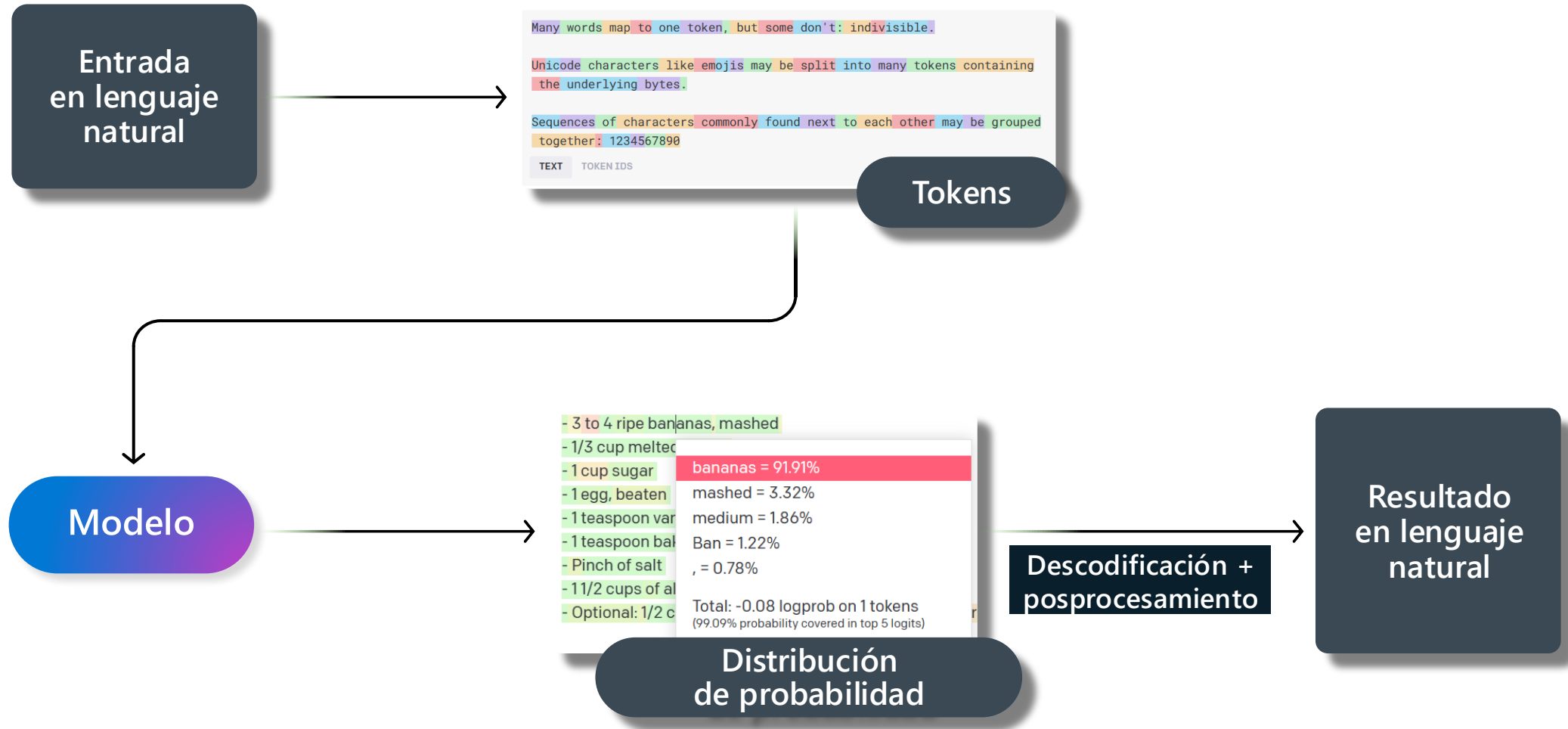
Hablaremos de los llms hoy en la clase de programación .

Hablaremos de los llms hoy en la clase de programación.

Selección de token



Simplemente, los LLMs predicen el siguiente token



Opciones de modelos

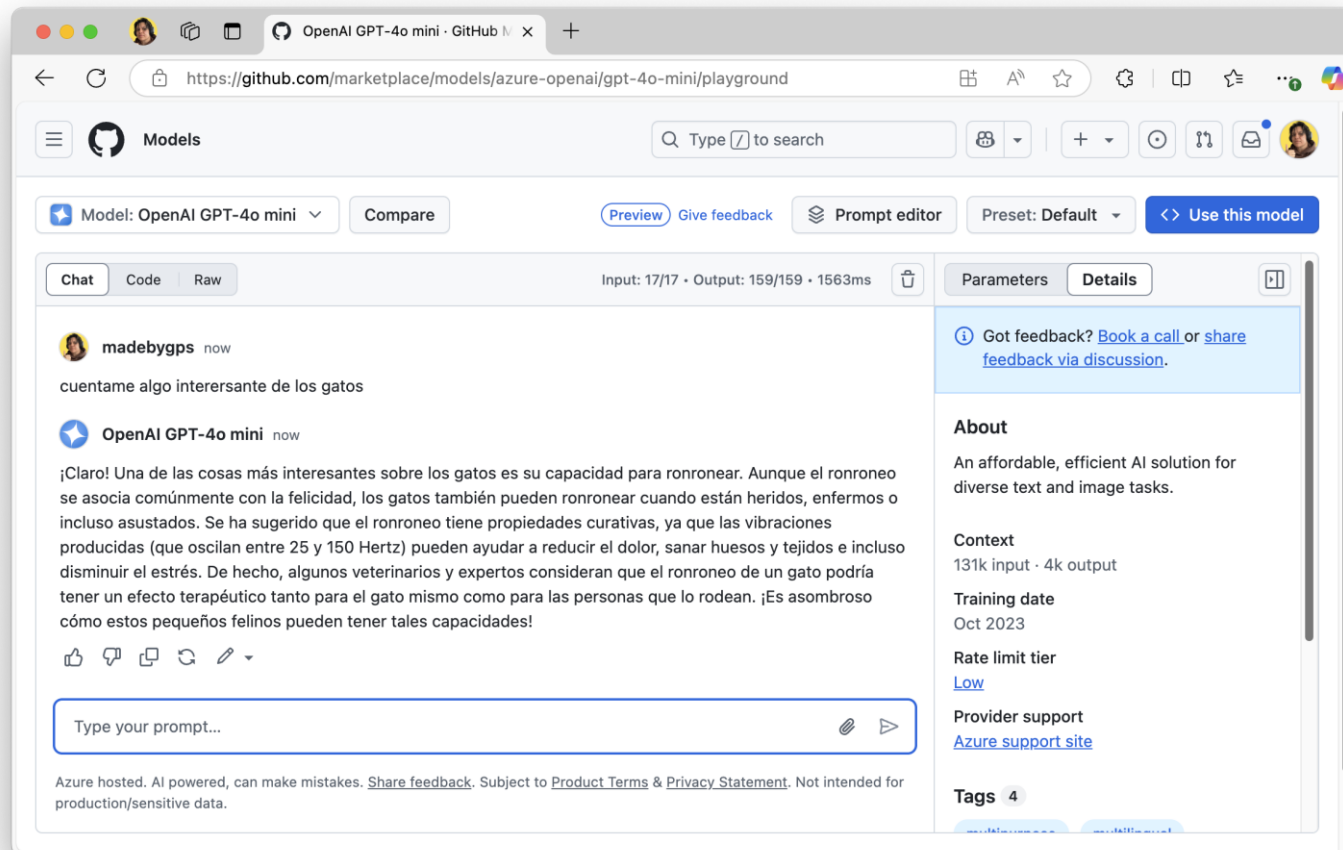
Modelos gestionados

Los LLMs gestionados se pueden acceder a través de una API, desde una empresa que administra el modelo y la infraestructura para usted.

| Host | Models |
|------------------------|----------------------------------------------------------------------------|
| OpenAI.com | GPT-3.5 GPT-4 GPT-4o |
| Azure AI | OpenAI GPT models, Meta models, Cohere, Mistral, DeepSeek,... |
| GitHub Models (GRATIS) | Azure AI models |
| Google | Gemini 1, 1.5 |
| Anthropic | Claude 3 family |

Demo: GitHub Models

Cualquier usuario de GitHub puede usar los modelos y los playgrounds:



<http://github.com/marketplace/models>

Modelos locales

Los modelos locales tienen open weights y pueden funcionar en máquinas personales.

Los SLM = «Small Language Models» son modelos más pequeños, < 100 B parámetros.

Local model runners:

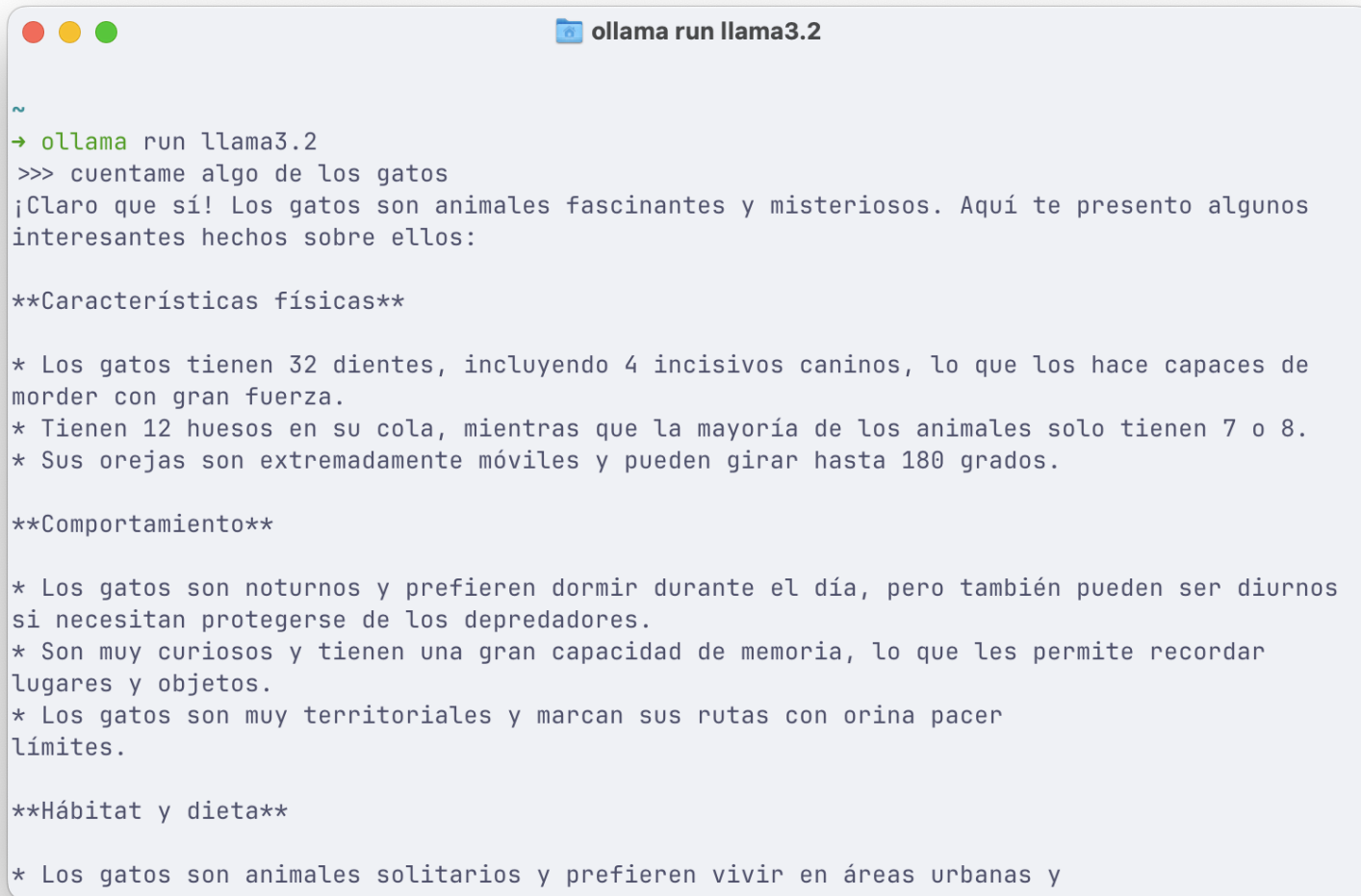
- Ollama
- Llamafire
- LM Studio

SLMs populares:

- Mistral
- Llama 3 series
- Phi4 series
- DeepSeek-R1
- Llava
- Gemma

Demo: Ollama

[Ollama](#) es una herramienta para ejecutar LLM locales en tu computadora.



```
ollama run llama3.2

~
→ ollama run llama3.2
>>> cuentame algo de los gatos
¡Claro que sí! Los gatos son animales fascinantes y misteriosos. Aquí te presento algunos interesantes hechos sobre ellos:

**Características físicas**

* Los gatos tienen 32 dientes, incluyendo 4 incisivos caninos, lo que los hace capaces de morder con gran fuerza.
* Tienen 12 huesos en su cola, mientras que la mayoría de los animales solo tienen 7 o 8.
* Sus orejas son extremadamente móviles y pueden girar hasta 180 grados.

**Comportamiento**

* Los gatos son nocturnos y prefieren dormir durante el día, pero también pueden ser diurnos si necesitan protegerse de los depredadores.
* Son muy curiosos y tienen una gran capacidad de memoria, lo que les permite recordar lugares y objetos.
* Los gatos son muy territoriales y marcan sus rutas con orina pacer límites.

**Hábitat y dieta**

* Los gatos son animales solitarios y prefieren vivir en áreas urbanas y
```

Usando LLMs desde Python

Librerías de Python para modelos de Azure AI

Estas librerías soportan directamente modelos gestionados por Azure/GitHub:

| Librería | Modelos compatibles |
|--------------------|------------------------------------------------------------------|
| openai | OpenAI.com models, Azure OpenAI models, OpenAI-compatible models |
| azure-ai-inference | Azure OpenAI, GitHub Models, Azure AI models |
| azure-ai-agents | Azure AI Agents service models |

Más adelante hablaremos de librerías wrappers/orquestadores como Langchain.

Librería OpenAI

La librería Python oficial de openai está disponible en PyPI:

```
pip install openai
```

Incluye soporte para **chat completions**, **embeddings**, y más.

openai librería es compatible con modelos gestionados en varios lugares:

- [Openai.com account](#)
- [Azure OpenAI account](#)
- [GitHub models](#)
- Local con OpenAI-compatible API ([Ollama/llamafire](#))

OpenAI demos repo

Usaremos este repo hoy o demos:

<https://github.com/pamelafox/python-openai-demos/spanish>

Usa estos enlaces para abrirlo con el host de OpenAI que prefieras:

- aka.ms/python-openai-github (GRATIS)
- aka.ms/python-openai-openai
- aka.ms/python-openai-azure
- aka.ms/python-openai-ollama

Autenticación API para hosts OpenAI

Para openai.com OpenAI, configura tu API key:

```
client = openai.OpenAI(api_key=os.environ["OPENAI_KEY"])
```

Para Azure OpenAI (keyless auth), usa Azure default credentials:

```
token_provider = azure.identity.get_bearer_token_provider(
    DefaultAzureCredential(), "https://cognitiveservices.azure.com/.default"
)
client = openai.AzureOpenAI(
    api_version=os.environ["AZURE_OPENAI_VERSION"],
    azure_endpoint=os.environ["AZURE_OPENAI_ENDPOINT"],
    azure_ad_token_provider=token_provider,
)
```

<https://learn.microsoft.com/azure/developer/ai/keyless-connections>

Autenticación API para OpenAI-like APIs

Para Ollama, configura tu base URL a localhost y key a cualquier valor:

```
client = openai.OpenAI(  
    base_url="http://localhost:11434/v1",  
    api_key="nokeyneeded",  
)
```

Para GitHub models, configura tu base URL a GitHub models host y configura key a PAT (personal access token):

```
client = openai.OpenAI(  
    base_url="https://models.inference.ai.azure.com",  
    api_key=os.environ["GITHUB_TOKEN"])
```

👉 En GitHub Codespaces, GITHUB_TOKEN siempre almacenará tu PAT. Si estás ejecutando localmente, crea un nuevo PAT.

Llamada a Chat Completion API

```
response = client.chat.completions.create(  
    model="gpt-4o",  
    temperature=0.7,  
    max_tokens=30,  
    n=1,  
    messages=[  
        {"role": "system", "content": "Eres un asistente útil que hace  
        muchas referencias a gatos y usa emojis."},  
        {"role": "user", "content": "Escribe un haiku sobre un gato  
        hambriento que quiere atún"}  
    ]  
)  
  
print(response.choices[0].message.content)
```

[Ejemplo completo: chat.py](#)

Transmite la respuesta

```
completion = client.chat.completions.create(  
    model=MODEL_NAME,  
    stream=True,  
    messages=[  
        {"role": "system", "content": "Eres un asistente útil que hace  
        muchas referencias a gatos y usa emojis."},  
        {"role": "user", "content": "Escribe un haiku sobre un gato  
        hambriento que quiere atún"}  
    ]  
)  
  
for event in completion:  
    print(event.choices[0].delta.content, end="", flush=True)
```

[Ejemplo completo: chat_stream.py](#)

LLMs: Pros y Cons

Pros:

- Creativo 😊
- Genial con los patrones
- Bueno con sintaxis (natural y de programación)

Cons:

- Creativo 😞
- Inventa cosas (sin saberlo)
- Ventana de contexto limitada (4K-128K)
- Más tokens = más \$, más tiempo

Como mejorar output del LLM

Maneras de mejorar LLM output

- **Prompt engineering:** Solicitar un tono y un formato específico
- **Few-shot examples:** Demostrar el formato de salida deseado
- **Llamadas encadenadas:** Haz que el LLM reflexione, reduzca la velocidad, lo descomponga
 - 😊 cubriremos hoy
- **Recuperación-Aumentada Generación (RAG):** Proveer contexto al LLM just-in-time
 - 🔍 cubriremos 3/18
- **Llamada a funciones y salidas estructuradas**
 - 🛠 cubriremos 3/25
- **Fine tuning:** Enseñar a LLM nueva información alterando permanentemente los weights

Prompt engineering

El primer mensaje enviado al modelo se llama «system message» o «system prompt». Utilízalo para establecer orientación general y reglas de formato.

```
response = client.chat.completions.create(  
    model=MODEL_NAME,  
    temperature=0.7,  
    messages=[  
        {"role": "system", "content": "Responde como Yoda"},  
        {"role": "user", "content": "¿Que es un LLM?"},  
    ]  
)
```

[Ejemplo completo: prompt_engineering.py](#)

Few-shot ejemplos

Otra forma de guiar a un modelo de lenguaje es proporcionar "few shots", es decir, una secuencia de ejemplos de preguntas y respuestas que demuestran cómo debería responder.

```
response = client.chat.completions.create(model=MODEL_NAME,  
    messages=[  
        {"role": "system", "content": "Eres un tutor que da pistas, no respuestas."},  
        {"role": "user", "content": "¿Cuál es la capital de Francia?"},  
        {"role": "assistant", "content": "¿Puedes pensar en la ciudad conocida por la Torre  
Eiffel?"},  
        {"role": "user", "content": "¿Cuál es la raíz cuadrada de 144?"},  
        {"role": "assistant", "content": "¿Qué número multiplicado por sí mismo es igual a  
144?"},  
        {"role": "user", "content": "¿Cuál es el número atómico del oxígeno?"},  
        {"role": "assistant", "content": "¿Cuántos protones tiene un átomo de oxígeno?"},  
        {"role": "user", "content": "¿Cuál es el planeta más grande del sistema solar?"},  
    ])
```

[Ejemplo completo: few_shot_examples.py](#)

Chained calls

```
response = client.chat.completions.create(model=MODEL_NAME,
    messages=[{"role": "user",
        "content": "Explica cómo funcionan los LLM en un solo párrafo."}])
explanation = response.choices[0].message.content

response = client.chat.completions.create(model=MODEL_NAME,
    messages=[{"role": "user",
        "content": f"Eres un editor. Revisa la siguiente explicación y proporciona
comentarios detallados sobre claridad, coherencia y cautivación. \n Explicación: \n
{explanation}"}])
feedback = response.choices[0].message.content

response = client.chat.completions.create(model=MODEL_NAME,
    messages=[{"role": "user",
        "content": f"Revisa el artículo usando los comentarios siguientes pero mantenlo a
un solo párrafo. \n Explicación:\n {explanation} \n\n Comentarios:\n{feedback}"}])
final_article = response.choices[0].message.content
```

[Ejemplo completo: chained_calls.py](#)

Librerías populares LLM

LLM librerías

Muchas librerías Python ofrecen una capa de abstracción para trabajar con LLMs:

- [Langchain](#): Orquestación
- [Llamaindex](#): Orquestación para RAG y Agentes
- [PydanticAI](#): Orquestación para RAG y Agentes
- [Semantic Kernel](#): Orquestación
- [Autogen](#): Orquestación para flujos de agentes
- [Litellm](#): Wrapper para varios hosts
- ...¡y muchos más!

Langchain

Conectando a GitHub models a través de Langchain:

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(
    model_name=os.environ["GITHUB_MODEL"],
    openai_api_base="https://models.inference.ai.azure.com",
    openai_api_key=os.environ["GITHUB_TOKEN"]
)
prompt = ChatPromptTemplate.from_messages(
    [("system", "Eres un asistente útil que hace muchas referencias a gatos."),
     ("user", "{input}")]
)
chain = prompt | llm
response = chain.invoke(
    {"input": "escribe un haiku sobre un gato hambriento que quiere atún"})
```

[Ejemplo completo: chat_langchain.py](#)

Llamaindex

Conectando a GitHub models a través de Llamaindex:

```
from llama_index.llms.openai_like import OpenAILike

llm = OpenAILike(
    model=os.environ["GITHUB_MODEL"],
    api_base="https://models.inference.ai.azure.com",
    api_key=os.environ["GITHUB_TOKEN"],
    is_chat_model=True,
)
chat_msgs = [
    ChatMessage(role=MessageRole.SYSTEM,
        content="Eres un asistente útil que hace muchas referencias a gatos."),
    ChatMessage(role=MessageRole.USER,
        content="escribe un haiku sobre un gato hambriento que quiere atún"),
]
response = llm.chat(chat_msgs)
```

[Ejemplo completo: chat_llamaindex.py](#)

Pydantic AI

Conectando a GitHub models a través de Pydantic AI:

```
from pydantic_ai.models.openai import OpenAIModel

llm = OpenAIModel(
    model="gpt-4o",
    base_url="https://models.inference.ai.azure.com",
    api_key=os.environ["GITHUB_TOKEN"]
)

agent = Agent(model, system_prompt="Eres un asistente útil que hace muchas referencias a gatos.")

result = agent.run_sync("escribe un haiku sobre un gato hambriento que quiere atún")
```

[Ejemplo completo: chat_pydantica.py](#)

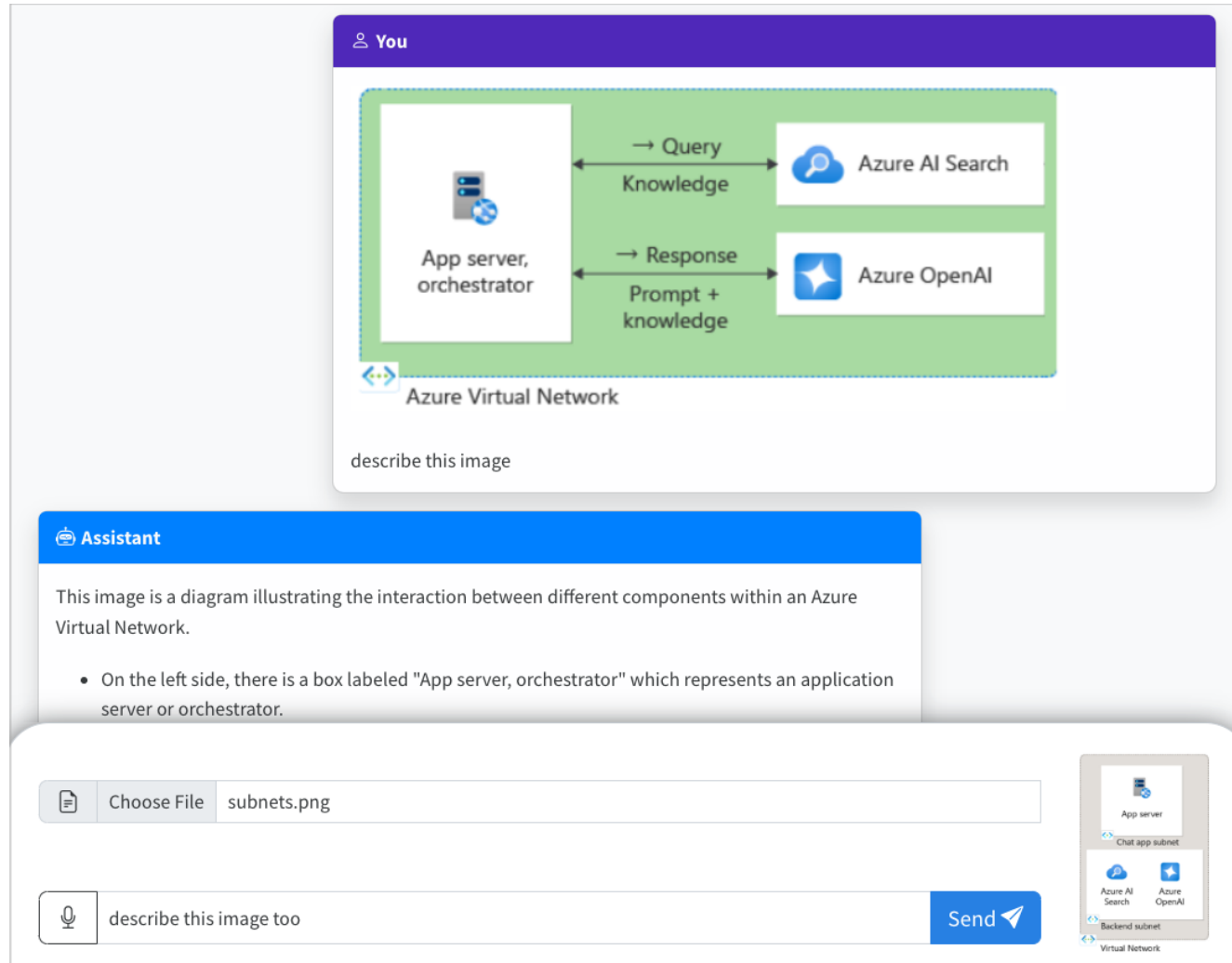
Cómo elegir una librería LLM

Considera:

- ¿Es compatible con todos los LLM/host que necesitas?
- ¿Tiene las funciones que necesitas (streaming, herramientas, etc.)?
- ¿Es fácil de usar y debuggear?
- ¿Está activamente mantenido?
- ¿Merecen la pena las ventajas?

Desarrollando aplicaciones impulsadas por LLMs

Chat + vision app



Supported model hosts:

- Azure OpenAI
- GitHub Models

Features:

- Streaming
- Speech I/O
- Image upload*

Code:

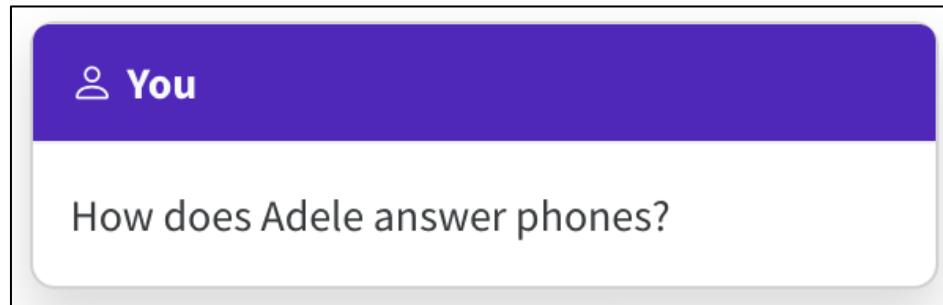
aka.ms/chat-vision-app

Profundizaremos en los modelos de visión en la sesión del 3/19.

App architecture

Frontend

(HTML, JavaScript)



User question

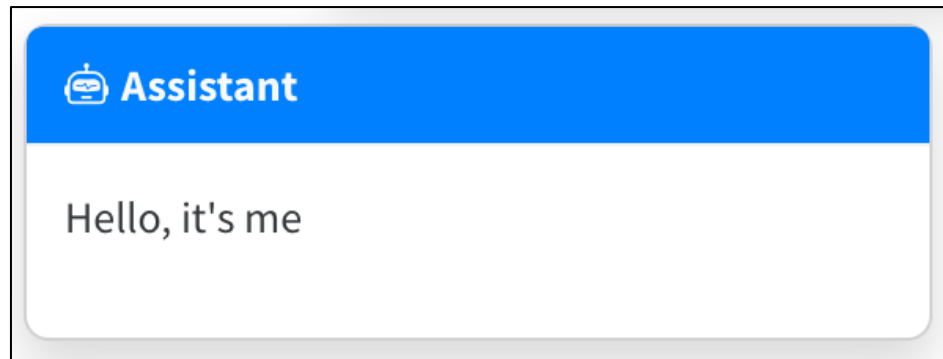
Python backend

(Quart, Uvicorn)

```
@bp.post("/chat/stream")  
async def chat_handler()
```



Streamed response



```
Transfer-Encoding: Chunked  
{ "content": "He"  
{ "content": "llo"  
{ "content": "It's"  
{ "content": "me"
```

Streaming respuestas en Quart

```
@bp.post("/chat/stream")
async def chat_handler():
    request_json = await request.get_json()

    @stream_with_context
    async def response_stream():

        chat_coroutine = bp.openai_client.chat.completions.create(
            model=os.environ["OPENAI_MODEL"],
            messages=request_json["messages"],
            stream=True,
        )
        async for event in await chat_coroutine:
            event_dict = event.model_dump()
            yield json.dumps(event_dict["choices"][0], ensure_ascii=False) + "\n"

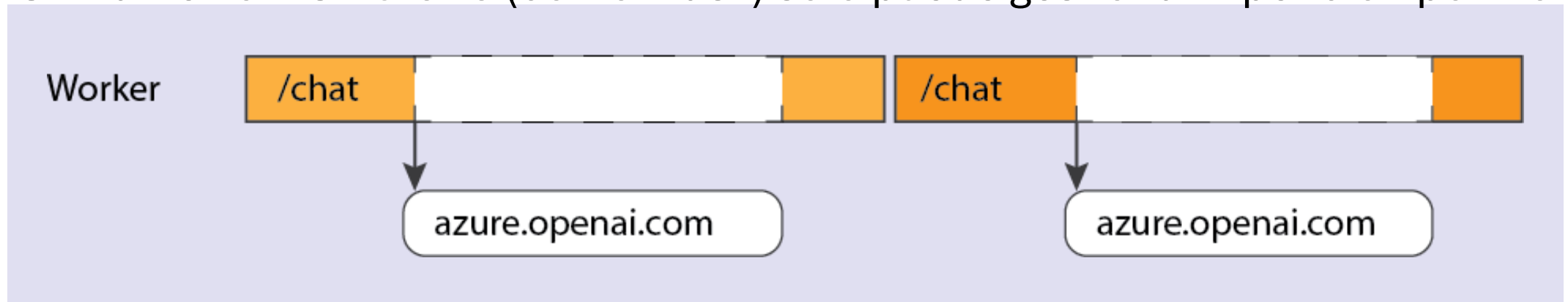
    return Response(response_stream())
```

blog.pamelafox.org/2023/09/best-practices-for-openai-chat-apps_16.html

P: ¿Por qué el backend utiliza Quart?

Muchos desarrolladores empiezan con Flask, uno de los framework web más popular.

Un framework **síncrono** (como Flask) solo puede gestionar 1 petición por worker:

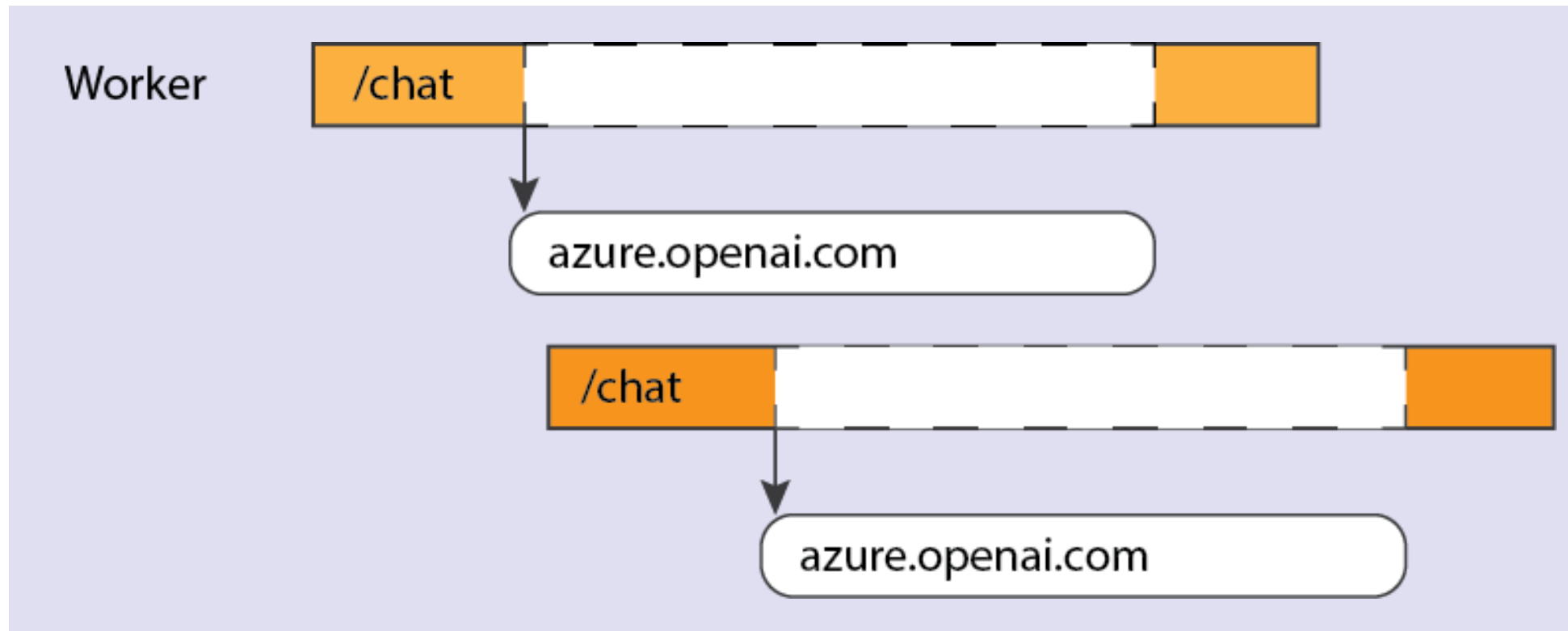


<https://blog.pamelafox.org/2023/09/best-practices-for-openai-chat-apps.html>

R: ¿Por qué el backend utiliza Quart?

Quart es la versión **asíncrona** de Flask.

Un framework **asíncrono** puede gestionar nuevas peticiones mientras espera la I/O:



Async frameworks para Python

Hay varias opciones populares:

| Framework | Ejemplo apps para Azure AI |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Quart | aka.ms/azai/chat aka.ms/chat-vision-app aka.ms/ragchat |
| FastAPI | aka.ms/azai/fastapi aka.ms/rag-postgres |
| Aiohttp | aka.ms/voicerag/repo |
| Django con async | |

<https://blog.pamelafox.org/2024/07/should-you-use-quart-or-fastapi-for-ai.html>

Próximos pasos

¡Únete a los próximos streams!
→

Ven a las horas de oficina los
Lunes en Discord:

aka.ms/pythonia/ho

Obtén más recursos de Python
AI

aka.ms/thefsource/Python_AI



3/11: LLMs



3/13: Vector embeddings



3/18: RAG



3/19: Models de Vision



3/25: Salidas estructuradas



3/27: Calidad y Seguridad

Regístrate @ aka.ms/PythonIA/series

The background is a light purple gradient. It is decorated with stylized, darker purple clouds along the top and bottom edges. In the corners, there are four-pointed starburst shapes in a darker shade of purple with yellow highlights.

Gracias