

Python + IA

Python + IA



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

Ponte al día @ aka.ms/pythonia/grabaciones





Python + IA



Retrieval Augmented Generation

Gwyneth Peña-Siguenza

Python Cloud Advocate

[Aka.ms/madebygps](https://aka.ms/madebygps)

Hoy cubriremos...

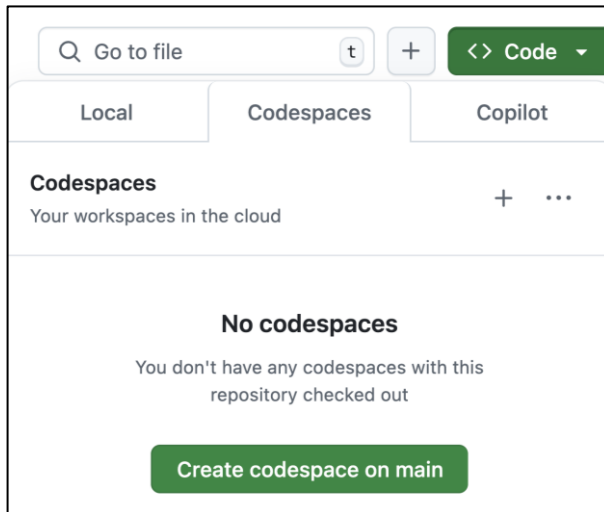
- Generación Aumentada por Recuperación
- Flujos de RAG simples y avanzados
- RAG en bases de datos
- RAG en documentos
- Más formas de construir RAG

¿Quieres seguir paso a paso?


1. Abre este repositorio de GitHub:

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

2. Usa el botón "Code" para crear un GitHub Codespace:




3. Wait a few minutes for Codespace to start up 




¿Por qué RAG?

Las limitaciones de los LLMs

 You

Write a model class using latest version of Flask SQLAlchemy

 Assistant

Sure! Here's an example of a model class using the latest version of Flask SQLAlchemy:

```
from flask_sqlalchemy import SQLAlchemy


db = SQLAlchemy()

class User(db.Model):
    __tablename__ = 'users'


    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True)
    email = db.Column(db.String(120), unique=True)
```

Conocimiento público
desactualizado

Sin conocimiento
interno

 You

Do my company perks cover underwater activities?

 Assistant

To determine whether your company perks cover underwater activities, you should consult your employee benefits package or reach out to your human resources department. They will be able to provide you with detailed information regarding the specific perks and benefits offered by your company, including any coverage for underwater activities.

Integrando conocimiento de dominio



Fine tuning

Aprender nuevas habilidades
(permanentemente)



Alto costo, tiempo



Generación Aumentada por Recuperación

Aprender nueva info
(temporalmente)

RAG en práctica

Teams Copilot (RAG en tus chats)

when does raghack start?

The RAGHack is scheduled to start on **September 3rd** and will run until **September 16th** ¹ ² ³.

If you have any other questions or need further details, feel free to ask!

1 J Re: RAGHack 2024: Speak... 2 A RE: RAGHack kick-off lives... +1

Copy

3 of 30 responses AI-generated content may be incorrect

Bing Copilot (RAG en la web)

You

what is RAG? 1-sentence TLDR

Copilot

Retrieval-Augmented Generation (RAG) is a technique that enhances the accuracy of generative AI models by fetching relevant information from external sources before generating responses ¹ ².

Learn more ¹ [aws.amazon.com](#) ² [geeksforgeeks.org](#) ³ [en.wikipedia.org](#) ⁴ [blogs.nvidia.com](#)

2 of 30 responses AI-generated content may be incorrect

GitHub Copilot (RAG en VS Code workspace)

pamelafox

@workspace are there any notebooks that do image search?

Workspace

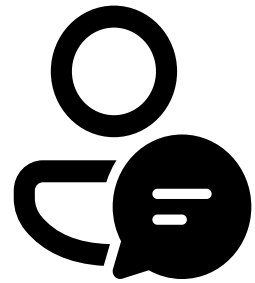
> Used 7 references

Yes, the `image_search.ipynb` notebook appears to be related to image search based on its name. You can open it to check the implementation details.

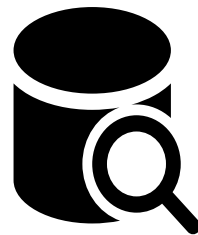
RAG 101

RAG: Retrieval Augmented Generation

¿Qué tan rápido es el Prius V?

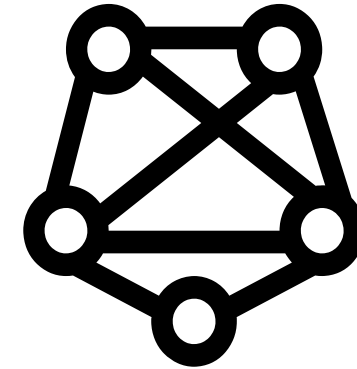


User
Question



Search

vehicle	year	msrp	acceleration
---	---	---	---
Prius (1st Gen)	1997	24509.74	7.46
Prius (2nd Gen)	2000	26832.25	7.97
Prius (3rd Gen)	2009	24641.18	9.6
Prius V	2011	27272.28	9.51
Prius C	2012	19006.62	9.35
Prius PHV	2012	32095.61	8.82
Prius C	2013	19080.0	8.7
Prius	2013	24200.0	10.2
Prius Plug-in	2013	32000.0	9.17



Language Model

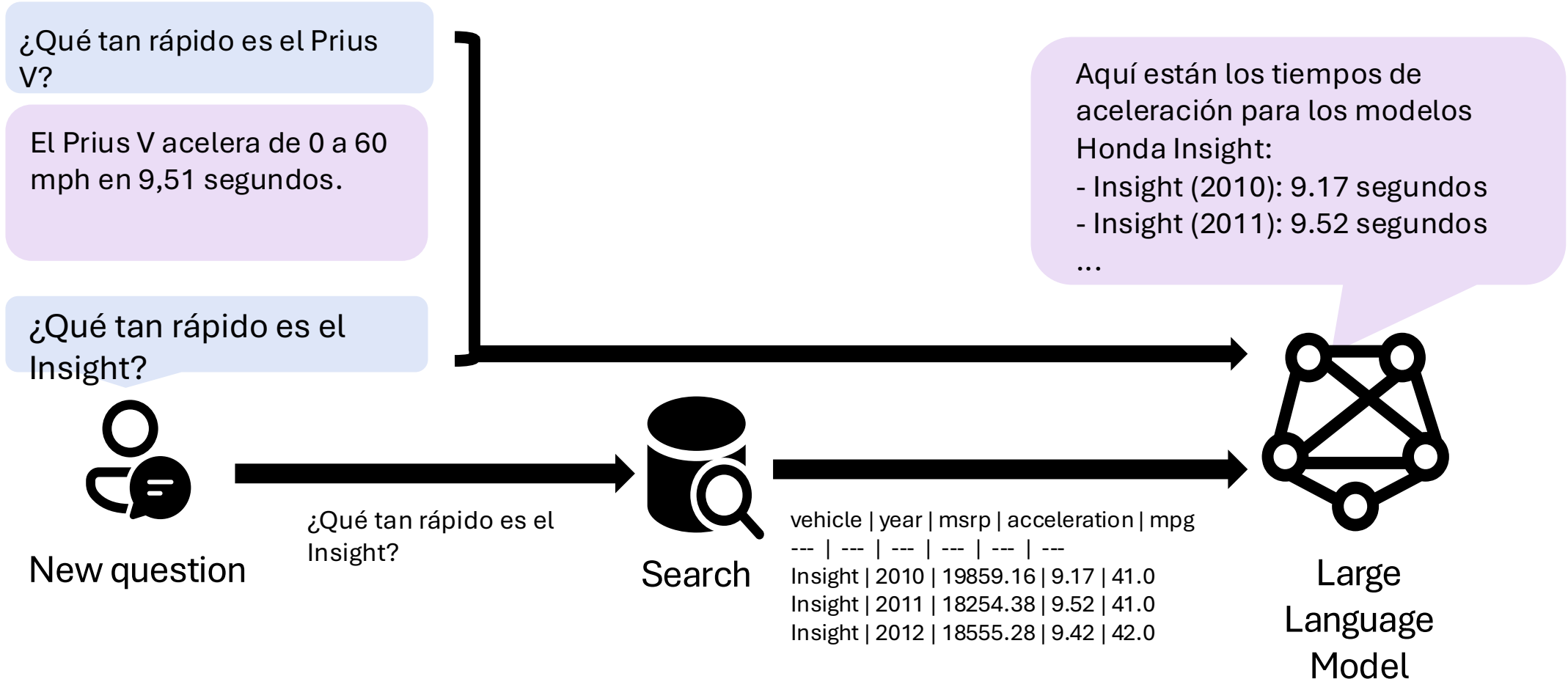
El Prius V acelera de 0 a 60 mph en 9,51 segundos.

RAG con OpenAI Python SDK

```
user_query = "How fast is the Prius V?"
retrieved_content = "vehicle | year | msrp | acceleration | mpg | class
--- | --- | --- | --- | --- | ---
Prius (1st Gen) | 1997 | 24509.74 | 7.46 | 41.26 | Compact
Prius (2nd Gen) | 2000 | 26832.25 | 7.97 | 45.23 | Compact..."

response = openai.chat.completions.create(
    messages = [
        {
            "role": "system",
            "content": "You must answer questions according to sources provided."
        },
        {
            "role": "user",
            "content": user_query + "\n Sources: \n" + retrieved_content
        }
    ]
])
```


RAG con soporte para multiturn



RAG con soporte multiturn (Código)

```
messages = [{"role": "system", "content": SYSTEM_MESSAGE}]

while True:
    question = input("\nYour question: ")
    matches = search(question)

    messages.append({"role": "user", "content": f"{question}\nSources: {matches}"})
    response = client.chat.completions.create(
        model=MODEL_NAME,
        temperature=0.3,
        messages=messages
    )

    bot_response = response.choices[0].message.content
    messages.append({"role": "assistant", "content": bot_response})
```

[rag_multiturn.py](#)

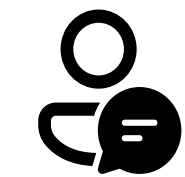
RAG con multiturn + query reescritura (rewriting)

¿Qué tan rápido es el Prius V?

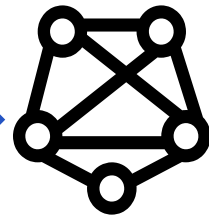
El Prius V acelera de 0 a 60 mph en 9,51 segundos.

Y el insight?

El Insight 2011 tiene un tiempo de aceleración de 9,52 segundos.



User
Question



Large
Language
Model

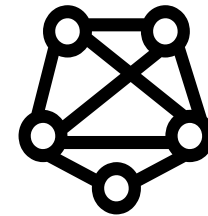
insight speed



Search



vehicle	year	msrp	acceleration	mpg
---	---	---	---	---
Insight	2010	19859.16	9.17	41.0
Insight	2011	18254.38	9.52	41.0
Insight	2012	18555.28	9.42	42.0



Large
Language
Model

RAG con multiturn + query rewriting (Code)

```
messages = [{"role": "system", "content": SYSTEM_MESSAGE}]

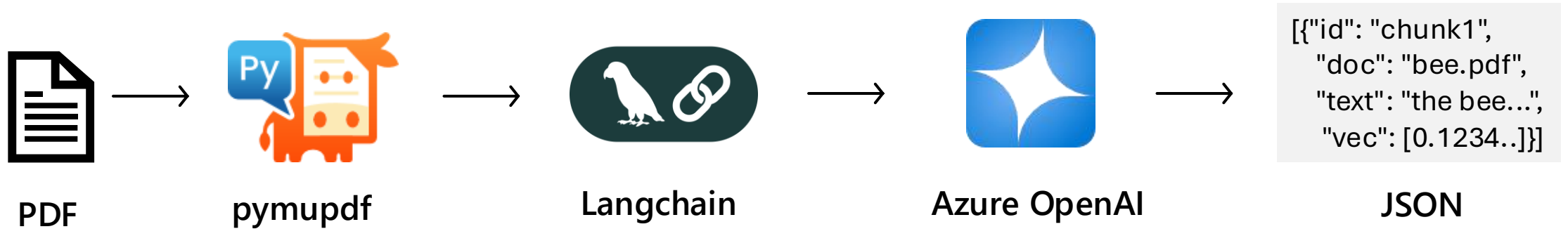
while True:
    question = input("\nYour question: ")
    matches = search(question)

    messages.append({"role": "user", "content": f"{question}\nSources: {matches}"})
    response = client.chat.completions.create(
        model=MODEL_NAME,
        temperature=0.3,
        messages=messages
    )

    bot_response = response.choices[0].message.content
    messages.append({"role": "assistant", "content": bot_response})
```

[rag_queryrewrite.py](#)

Ingesta de documentos RAG



Extraer texto de PDF

Otras opciones para este paso:
Azure Document Intelligence,
cargadores de documentos de
Langchain,
servicios OCR, Unstructured,
etc.

Dividir datos en fragmentos

Dividir el texto en base a los
límites de oraciones y
longitudes de tokens.
También podrías usar divisores
"semánticos" y tus propios
divisores personalizados.

Vectorizar fragmentos

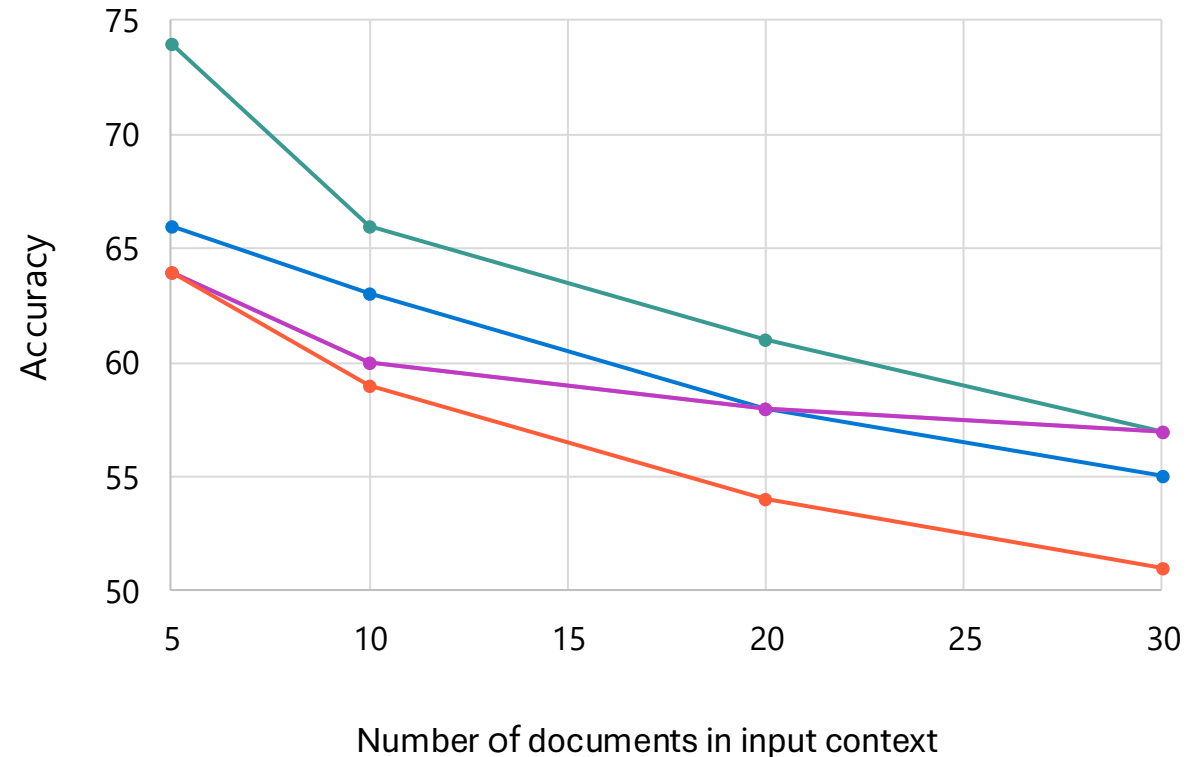
Calcular embeddings utilizando
el modelo de embeddings que
elijas.

Almacenar fragmentos

Aquí es donde normalmente
usarías un servicio de
búsqueda como Azure AI
Search o una base de datos
como PostgreSQL.

¿Por qué necesitamos dividir los documentos?

- 1 Los LLMs tienen ventanas de contexto limitadas (4K – 128K).
- 2 Cuando un LLM recibe demasiada información, puede distraerse fácilmente con detalles irrelevantes.
- 3 Cuantos más tokens envíes, mayor será el costo y más lenta la respuesta.



Source: Lost in the Middle: How Language Models Use Long Contexts, Liu et al. arXiv:2307.03172

Tamaño óptimo de fragmento de documento

¿Qué tan grandes deberían ser los fragmentos?

# de tokens por fragments	Recall@50
512	42.4
1024	37.5
4096	36.4
8191	34.9

Source: <https://aka.ms/ragrelevance>

Un **token** es la unidad de medida para la entrada/salida de un modelo de lenguaje. Aproximadamente 1 token por palabra en inglés, y proporciones más altas en otros idiomas.

Más sobre las proporciones de tokens: <https://aka.ms/genai-cjk>

¿Dónde se deben dividir los fragmentos?

Estrategia de límites de fragmentación	Recall@50
Dividir en el límite de tokens	40.9
Conservar los límites de las oraciones	42.4
10% fragmentos superpuestos	43.1
25% fragmentos superpuestos	43.9

Source: <https://aka.ms/ragrelevance>

Un algoritmo de fragmentación también debería considerar las tablas y evitar dividir las cuando sea posible.

RAG document ingestion (Code)

```
filenames = ["data/California_carpenter_bee.pdf", "data/Centris_pallida.pdf"] all_chunks = []
for filename in filenames:
    md_text = pymupdf4llm.to_markdown(filename)

    text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
        model_name="gpt-4o", chunk_size=500, chunk_overlap=0)
    texts = text_splitter.create_documents([md_text])
    file_chunks = [{"id": f"{filename}-{i + 1}", "text": text.page_content}
                    for i, text in enumerate(texts)]

    for file_chunk in file_chunks:
        file_chunk["embedding"] = (client.embeddings.create(
            model="text-embedding-3-small", input=file_chunk["text"])
            .data[0].embedding)

    all_chunks.extend(file_chunks)
```

[rag_documents_ingestion.py](#)

Simple RAG flow on documents (Code)

```
user_question = "where do digger bees live?"

docs = index.search(user_question)
context = "\n".join([f"{doc['id']}: {doc['text']}" for doc in docs[0:5]])

SYSTEM_MESSAGE = """
You must use the data set to answer the questions,
you should not provide any info that is not in the provided sources.
Cite the sources you used to answer the question inside square brackets.
The sources are in the format: <id>: <text>.
"""

response = client.chat.completions.create(
    model="gpt-4o",
    temperature=0.3,
    messages=[
        {"role": "system", "content": SYSTEM_MESSAGE},
        {"role": "user", "content": f"{user_question}\nSources: {context}"}])
```

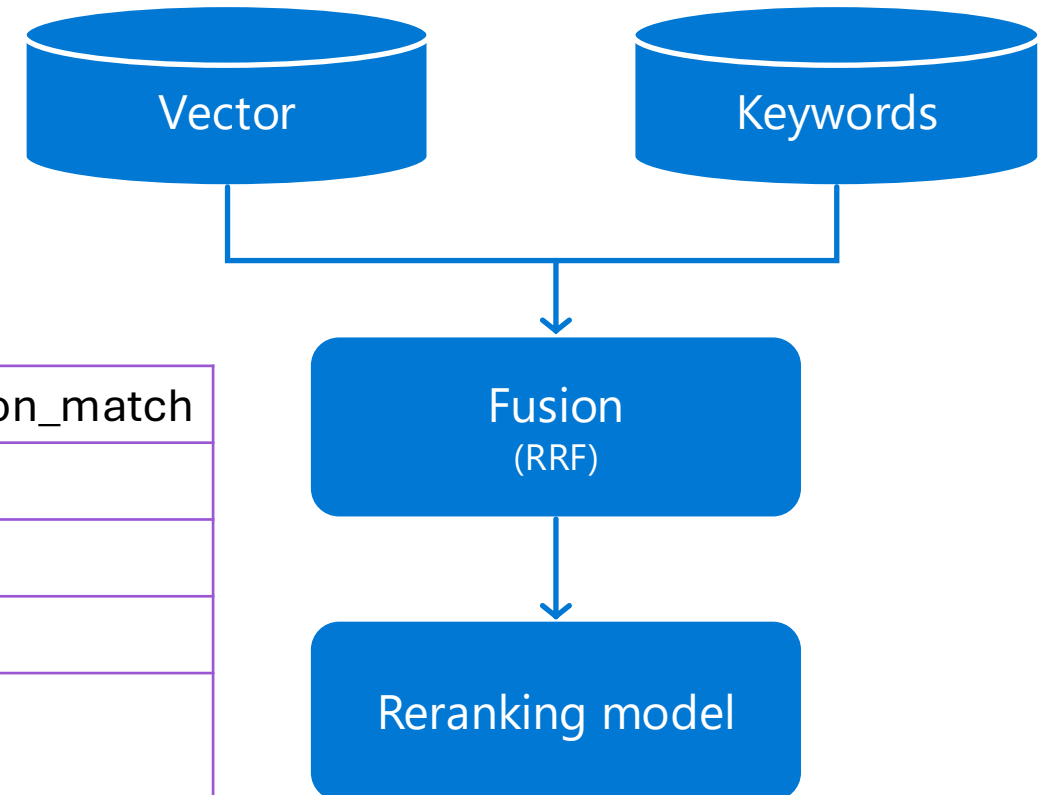
[rag_documents_flow.py](#)

RAG con recuperación híbrida

La búsqueda **completa** funciona mejor:

- Recuperación híbrida (palabras clave + vectores) > solo vector o solo palabras clave
- Hybrid + Reranking > Hybrid

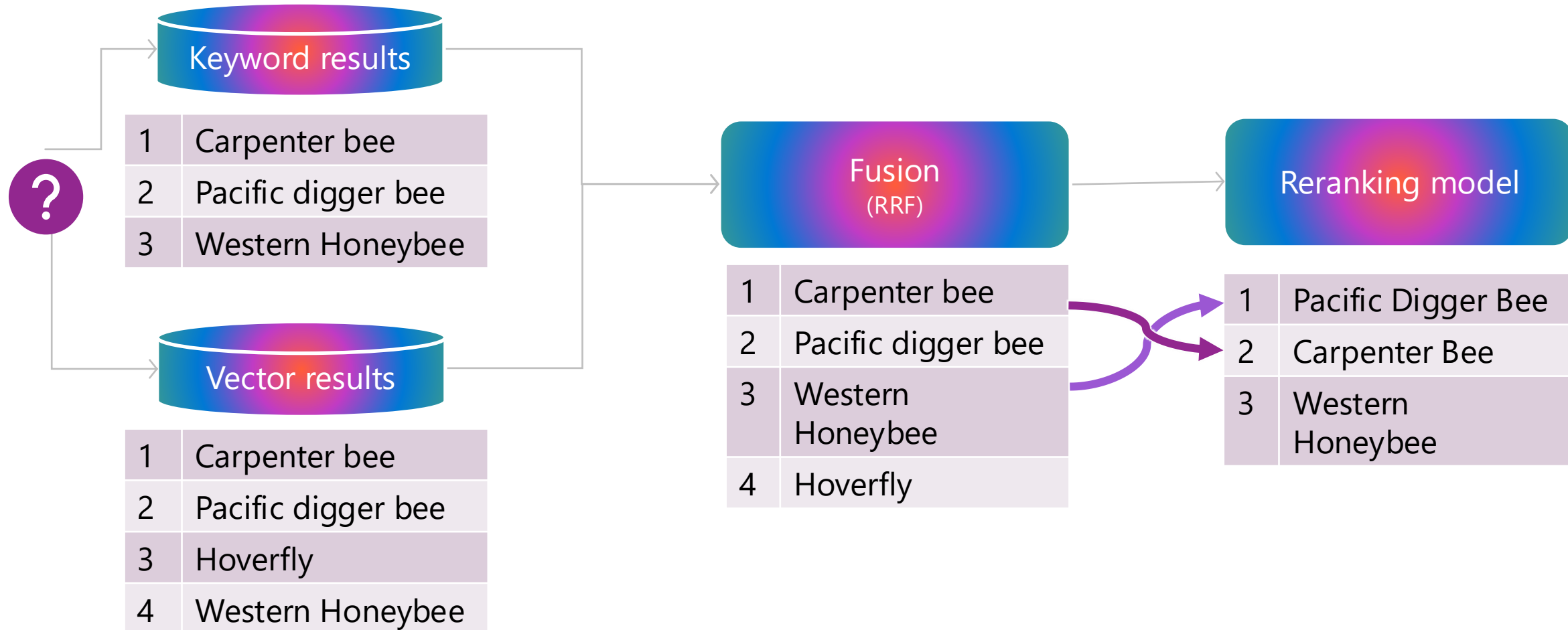
search mode	groundedness	relevance	citation_match
vector only	2.79	1.81	0.02
text only	4.87	4.74	0.89
hybrid	3.26	2.15	0.11
hybrid with ranker	4.89	4.78	0.92



<https://blog.pamelafox.org/2024/06/vector-search-is-not-enough.html>

Flujo de recuperación híbrida

Question: "cute gray fuzzy bee"



RAG con hybrid retrieval (Code)

```
def full_text_search(query, limit):

def vector_search(query, limit):

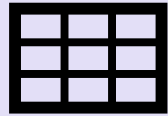
def reciprocal_rank_fusion(text_results, vector_results, alpha=0.5):

def rerank(query, retrieved_documents):
    encoder = CrossEncoder("cross-encoder/ms-marco-MiniLM-L-6-v2")
    scores = encoder.predict([[query, doc["text"]] for doc in retrieved_documents])
    return [v for _, v in sorted(zip(scores, retrieved_documents), reverse=True)]

def hybrid_search(query, limit):
    text_results = full_text_search(query, limit * 2)
    vector_results = vector_search(query, limit * 2)
    combined_results = reciprocal_rank_fusion(text_results, vector_results)
    combined_results = rerank(query, combined_results)
    return combined_results[:limit]
```

[rag_documents_hybrid.py](#)

Tipos de fuentes de datos RAG



Database rows (Datos estructurados)

Necesitas una forma de **vectorizar** las columnas objetivo con un modelo de embeddings.

Necesitas una forma de **buscar** entre las filas vectorizadas.



Documents (Datos no estructurados)

PDFs, docx, pptx, md, html, images

Necesitas un proceso de **ingestión** para **extraer**, **dividir**, **vectorizar** y **almacenar** fragmentos de documentos.

Necesitas una forma de **buscar** entre los fragmentos vectorizados.

RAG con PostgreSQL

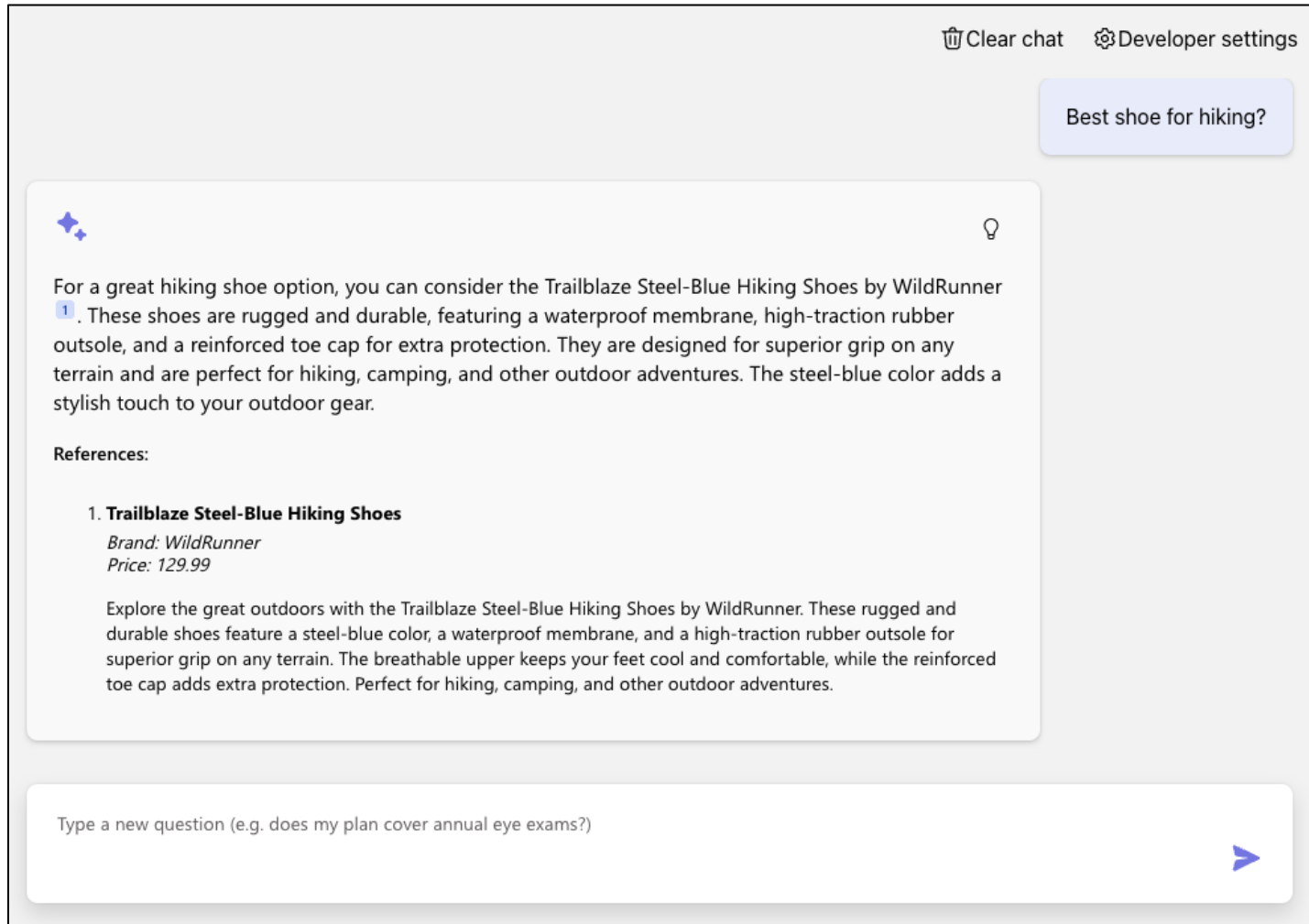
RAG con PostgreSQL in Python

```
question = "algun zapato barato para senderismo?"

cur.execute("SELECT ... ")
results = cur.fetchall()
for result in results:
    formatted_results += f"## {result[1]}\n\n{result[2]}\n"

response = openai.chat.completions.create(
    messages = [
        {
            "role": "system",
            "content": "Debes utilizar las fuentes de datos para responder las preguntas"
        },
        {
            "role": "user",
            "content": question + "\n Fuentes: \n" + formatted_content
        }
    ])
```

Open source template: RAG con PostgreSQL



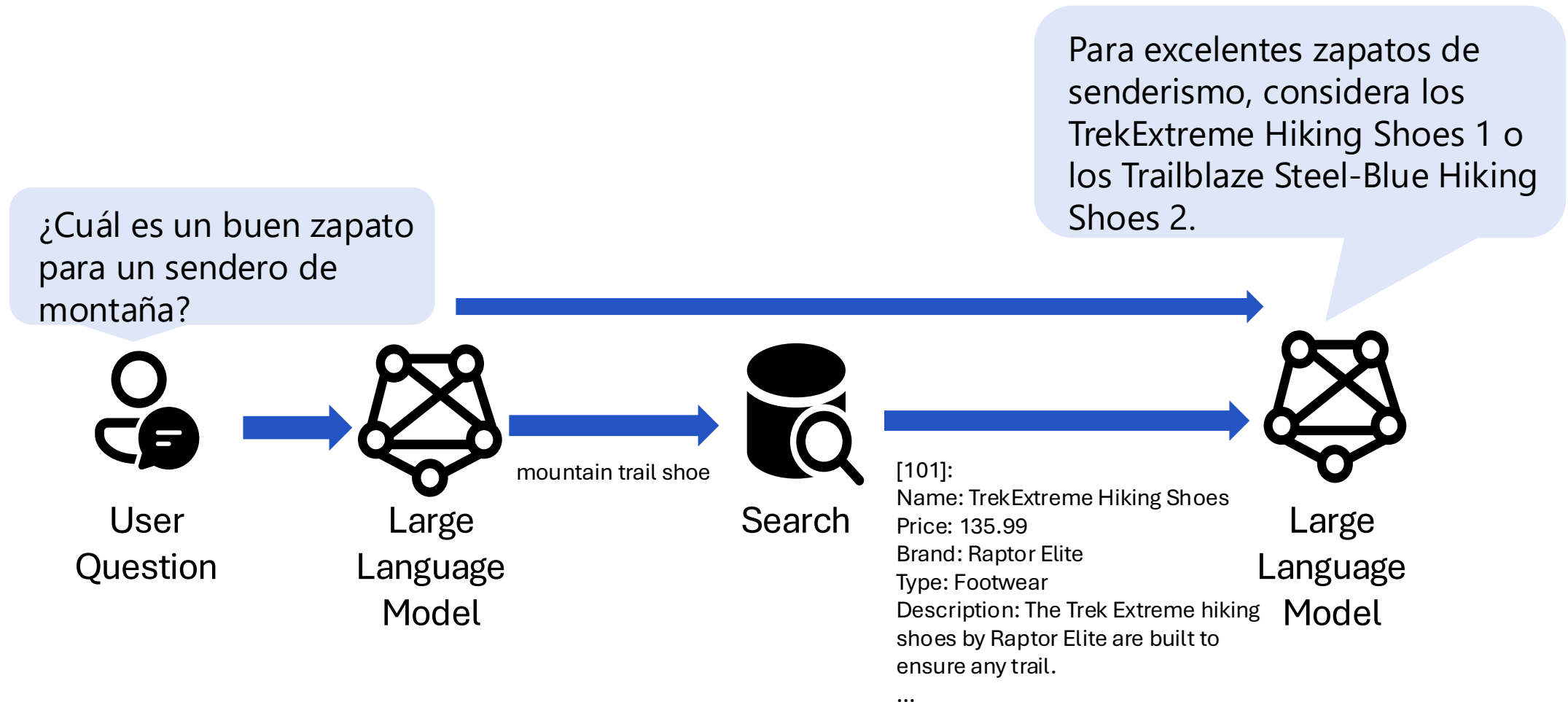
Azure OpenAI +
Azure PostgreSQL Flexible Server +
Azure Container Apps

Supports simple and advanced RAG

Code:
aka.ms/rag-postgres

Demo:
aka.ms/rag-postgres/demo

RAG con postgres: flow con query rewriting



RAG con PostgreSQL: App architettura

Typescript frontend
(React, FluentUI)

chat.tsx
makeApiRequest()

api.ts
chatApi()

Python backend
(FastAPI, Uvicorn)

app.py
chat()

advanced_rag.py
run()

get_search_query()
compute_text_embedding()
search()
get_messages_from_history()
chat.completions.create()

RAG con PostgreSQL: Resumen de código

File:	Controls:
src/backend/fastapi_app/postgres_models.py	Table schema with embedding columns, indexes
src/backend/fastapi_app/postgres_searcher.py	SQL query to perform vector+keyword+RR
src/backend/fastapi_app/rag_simple.py	Simple RAG flow
src/backend/fastapi_app/rag_advanced.py	Advanced RAG flow with query rewriting
src/backend/fastapi_app/prompts/	All the prompts used by flows
src/frontend/index.html	title, metadata, script tag
src/frontend/src/pages/chat/Chat.tsx	“Chat” tab for asking questions

RAG con Azure AI Search

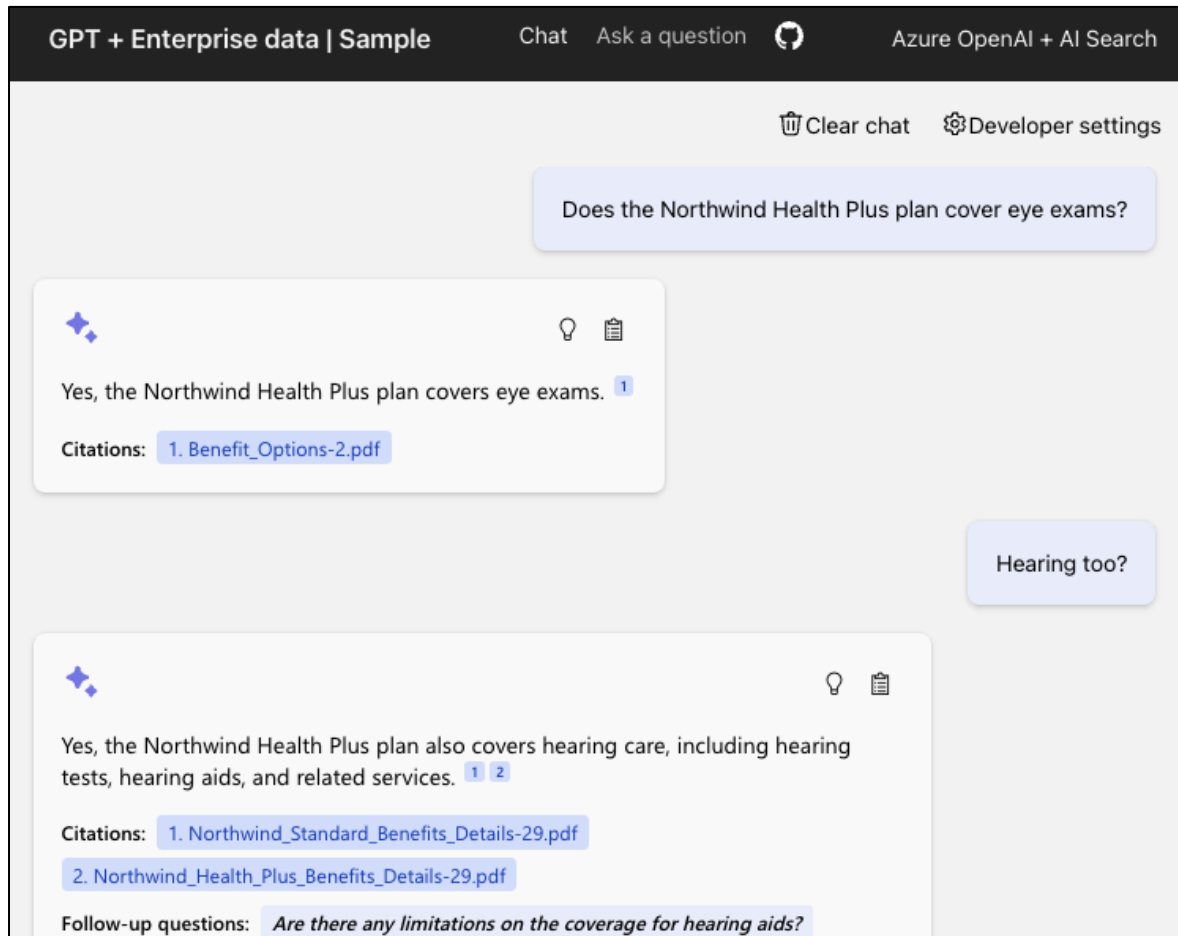
RAG con documents y Azure AI Search

```
user_question = "What does a product manager do?"

user_question_vector = get_embedding(user_question)
r = search_client.search(user_question,
    vector_queries=[VectorizedQuery(vector=user_question_vector, fields="vector")],
    sources = "\n\n".join([f"[{doc['sourcepage']}]: {doc['content']}\n" for doc in r])
response = openai_client.chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "system",
         "content": "Answer ONLY with the facts from sources below. Cite sources with brackets."},
        {"role": "user",
         "content": user_question + "\nSources: " + sources}])
```

[rag-with-azure-ai-search-notebooks: rag.ipynb](#)

Open source template: RAG con AI Search



Azure OpenAI +
Azure AI Search +
Azure App Service/Container Apps

Supports simple and advanced flows
(Ask tab vs. Chat tab)

Code:
aka.ms/ragchat

Demo:
aka.ms/ragchat/demo

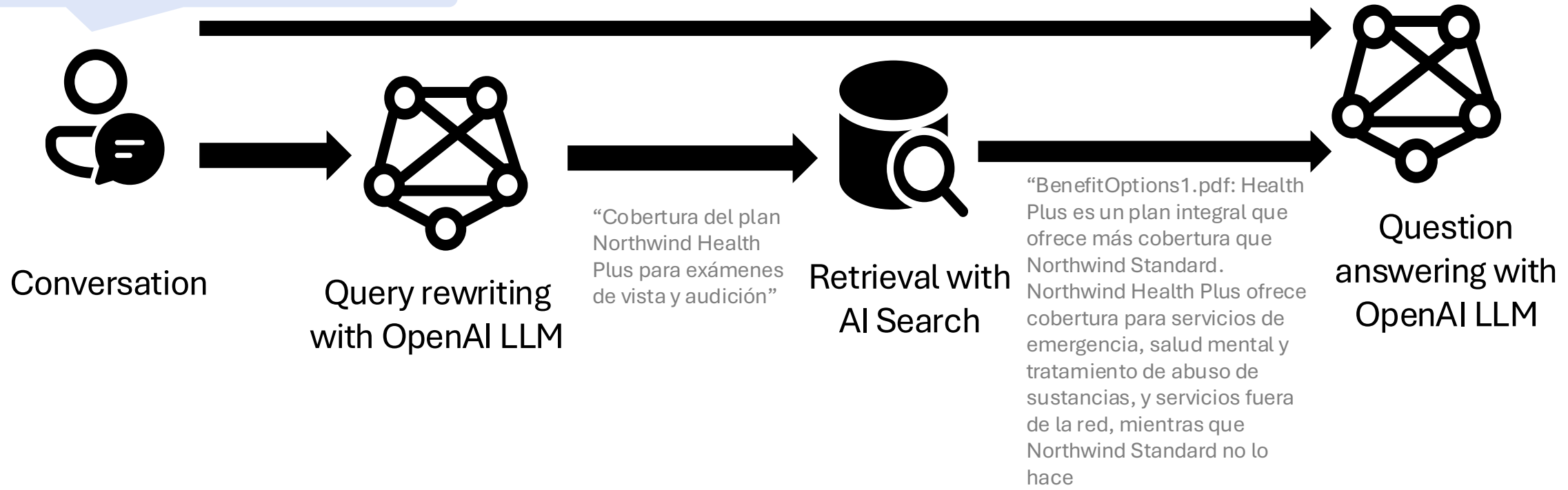
RAG con AI Search: flow con query rewriting

¿El plan Northwind Health Plus cubre exámenes de vista?

Sí, el plan Northwind Health Plus cubre exámenes de vista. 1

¿Y audición?

Sí, el plan Northwind Health Plus también cubre exámenes auditivos. 1



RAG con AI Search: Ingesta de datos

El proceso de ingestión se maneja mediante un script en Python:



Azure Blob
Storage



Azure Document
Intelligence



Python



Azure
OpenAI



Azure
AI Search

Subir documentos

Una versión en línea de cada documento es necesaria para citas clicables.

Extraer datos de los documentos

Soporta PDF, HTML, docx, pptx, xlsx, imágenes y, además, puede realizar OCR cuando sea necesario. Parsers locales también disponibles para PDF, HTML, JSON, txt.

Dividir datos en fragmentos

Dividir el texto basándose en límites de oraciones y longitudes de tokens. También se podrían usar divisores de Langchain aquí..

Vectorizar fragmentos

Calcular embeddings utilizando el modelo de embedding de OpenAI que elijas.

Indexing

- Índice de documentos
- Índice de fragmentos
- Ambos

RAG con AI Search: app arquitectura

Typescript frontend

/app/frontend

(React, FluentUI)

chat.tsx
makeApiRequest()

api.ts
chatApi()

Python backend

/app/backend

(Quart, Uvicorn)

app.py
chat()

chatreadretrieveread.py
run()

get_search_query()
search()
chat.completions.create()

RAG with AI Search: Resumen de código

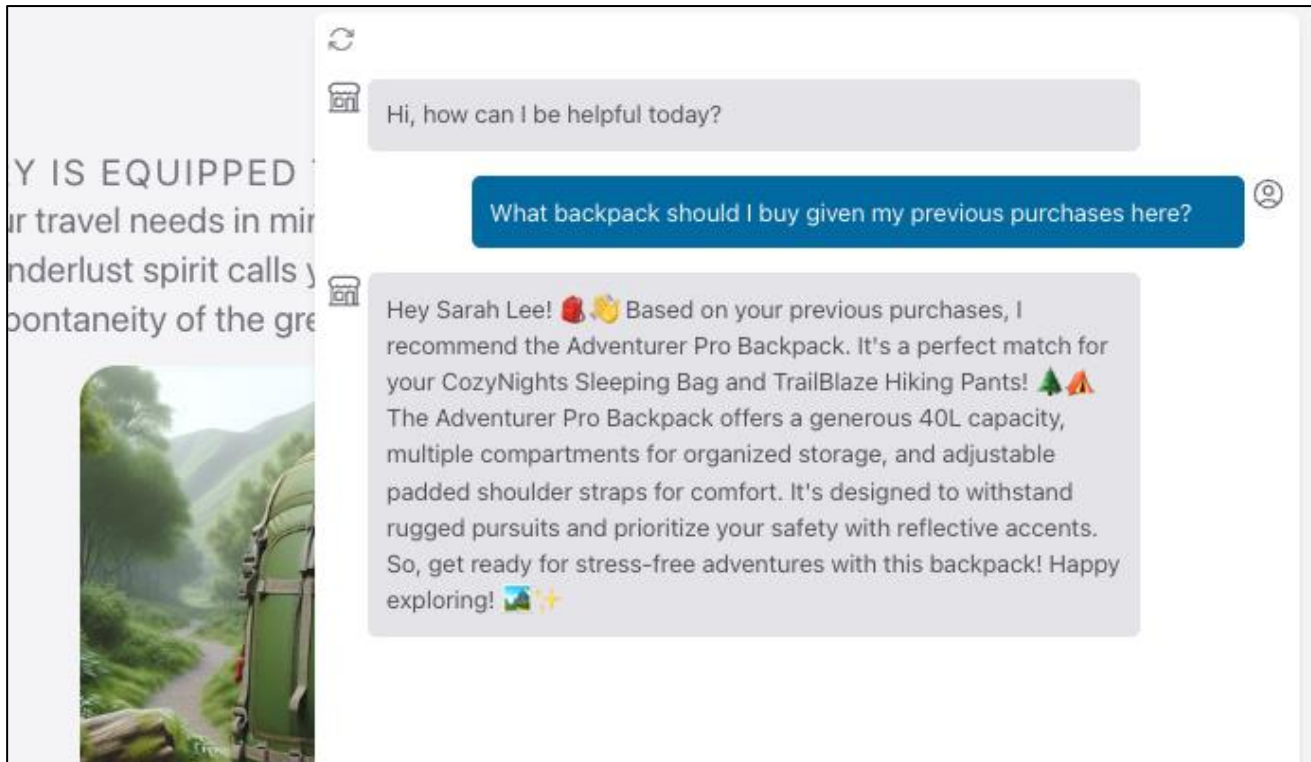
File:	Controls:
app/backend/app.py	app routes, app configuration
app/backend/approaches/chatreadretrieveread.py	“Chat” tab, RAG prompt and flow
app/backend/approaches/retrievethenread.py	“Ask” tab, RAG prompt and flow
app/backend/approaches/prompts/	All the prompts used by flows.
app/frontend/index.html	title, metadata, script tag
app/frontend/src/pages/chat/Chat.tsx	“Chat” tab and default settings
app/frontend/src/pages/ask/Ask.tsx	“Ask” tab and default settings

Formas de construir aplicaciones RAG

Componentes de una app RAG

Componente	Ejemplo
Ingesta: Herramientas para procesar datos en un formato que pueda ser indexado y procesado por un LLM	Azure: Document Intelligence Local: PyMuPDF, BeautifulSoup
Recuperador: Una base de conocimientos que pueda recuperar eficientemente fuentes que coincidan con una consulta del usuario (Idealmente soporta tanto búsqueda vectorial como búsqueda de texto completo)	Azure: Azure AI Search, Azure CosmosDB, Local: PostgreSQL, Qdrant, Pinecone
LLM: Un modelo que pueda responder preguntas basándose en la consulta y en las fuentes proporcionadas, y que pueda incluir citas	OpenAI: GPT 3.5, GPT 4, GPT-4o Azure AI Studio: Meta Llama3, Mistral, Cohere R+ Anthropic: Claude 3.5 Google: Gemini 1.5
Orquestador (opcional): Una forma de organizar las llamadas al recuperador y al LLM	Community: Langchain, Llamaindex Microsoft: Semantic Kernel, Autogen
Features	Chat history, Feedback buttons, Text-to-speech, User login, File upload, Access control, etc.

Open source template: Azure AI Foundry



Retriever: Azure AI Search

LLM: OpenAI

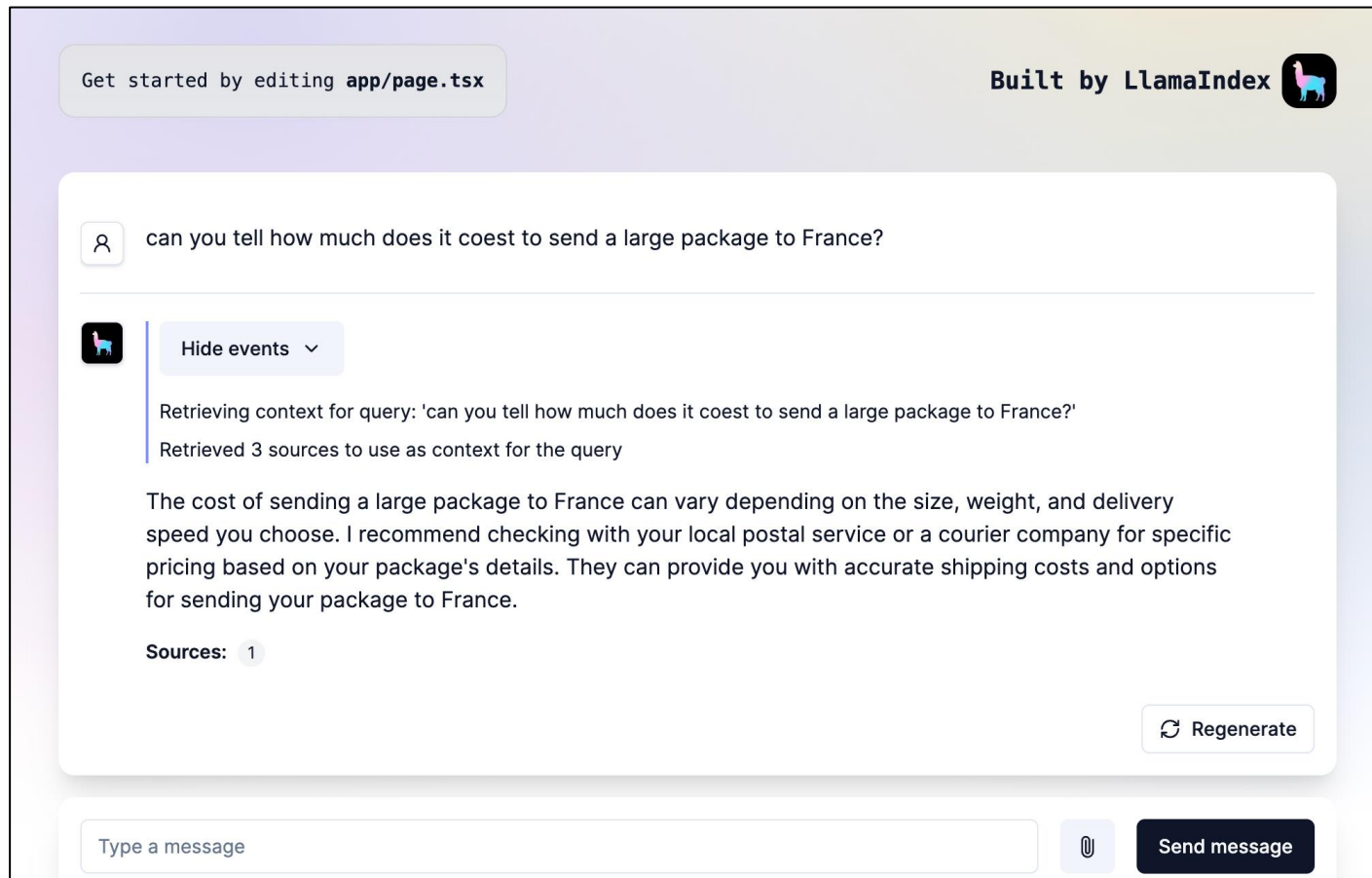
Orchestrator: Prompty

Features:

CosmosDB user info lookup

<https://github.com/Azure-Samples/contoso-chat>

Open source template: RAG con Llamaindex



Retriever: In-Memory

LLM: OpenAI

Orchestrator: Llamaindex

<https://github.com/Azure-Samples/llama-index-python>

Notebook Ejemplos de Cosmos DB

```
# Perform a vector search on the Cosmos DB container
def vector_search(container, vectors, similarity_score=0.02, num_results=5):
    # Execute the query
    results = container.query_items(
        query= '''
            SELECT TOP @num_results c.overview, VectorDistance(c.vector, @embedding) as SimilarityScore
            FROM c
            WHERE VectorDistance(c.vector,@embedding) > @similarity_score
            ORDER BY VectorDistance(c.vector,@embedding)
            ''',
        parameters=[
            {"name": "@embedding", "value": vectors},
            {"name": "@num_results", "value": num_results},
            {"name": "@similarity_score", "value": similarity_score}
        ],
        enable_cross_partition_query=True, populate_query_metrics=True)
    results = list(results)
    # Extract the necessary information from the results
    formatted_results = []
    for result in results:
        score = result.pop('SimilarityScore')
        formatted_result = {
            'SimilarityScore': score,
            'document': result
        }
        formatted_results.append(formatted_result)
```

Retriever: Cosmos DB

LLM: OpenAI

Orchestrator: Varies

aka.ms/cosmosdb-rag-samples

Mas maneras de RAG

GraphRAG

<https://www.microsoft.com/research/project/graphrag/>

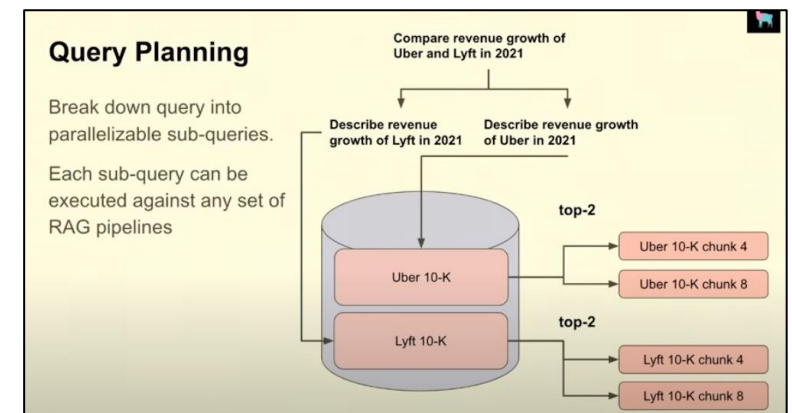
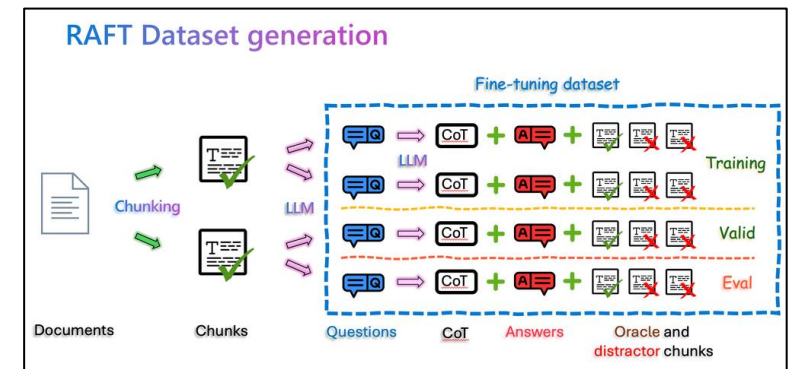
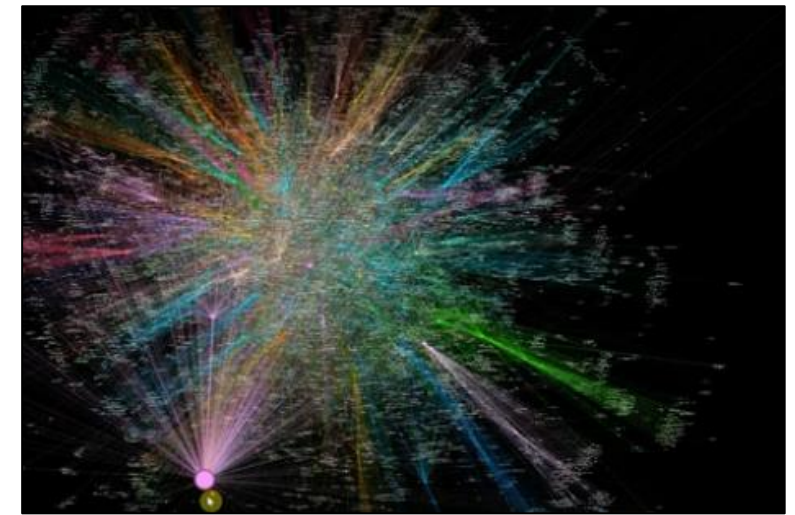
RAFT (RAG + FineTuning)

<https://github.com/ShishirPatil/gorilla/tree/main/raft>

<https://github.com/Azure-Samples/raft-distillation-recipe>

Agentic RAG

<https://www.youtube.com/live/aQ4yQXeB1Ss>



Charlas acerca de RAG

RAGHack (August 2024): 25+ streams about building RAG on Azure
aka.ms/raghack/streams

RAG Deep Dive (January 2025): 11 streams about azure-search-openai-demo
aka.ms/ragdeepdive/watch

RAG Time (March 2025): Advanced topics on RAG with Azure AI Search
aka.ms/rag-time

Building RAG from Scratch with GitHub Models
aka.ms/rag-vs-code-github-models

Próximos pasos

¡próximos streams! →

horas de oficina los Lunes en
Discord:

aka.ms/pythonia/ho

más recursos de Python AI:

aka.ms/thesource/Python_AI

Agents hack

Aka.ms/agentshack



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

Ponte al día @ aka.ms/pythonia/grabaciones

The background is a light purple gradient. It features stylized, darker purple clouds along the top and bottom edges. In the corners, there are decorative starburst or sparkles in shades of purple and yellow.

Gracias