Microsoft

LIVE

Python + IA

Microsoft

LIVE

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

[Prototipando Agentes de IA con GitHub Models]

Microsoft

Python + IA

🛠️ Llamada a funciones y salidas estructuradas

Gwyneth Peña-Siguenza

Python Cloud Advocate

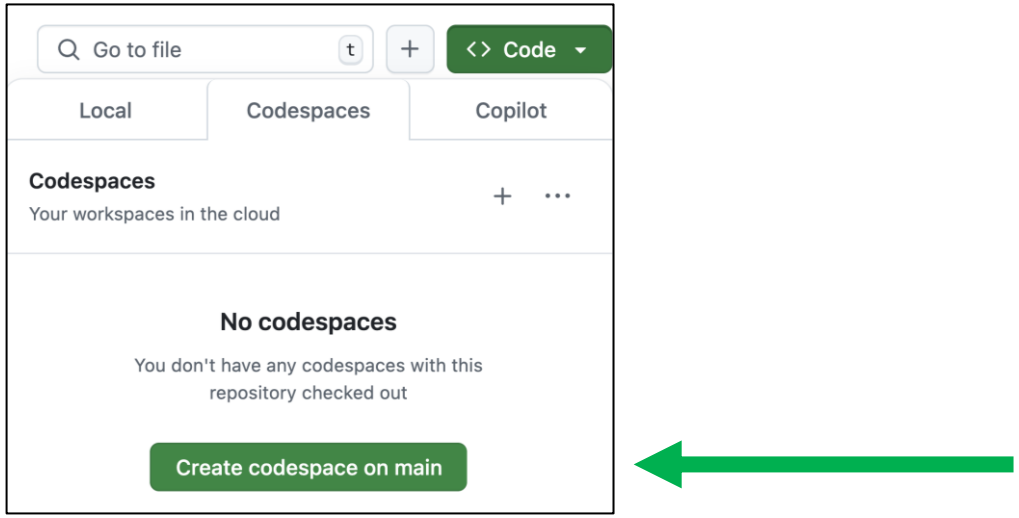
linkedin.com/in/madebygps

Agenda

- Function calling
- Salidas Estructuradas
- Escenarios populares

¿Quieres seguir los pasos?

1. Abre este repositorio de GitHub: https://github.com/pamelafox/python-openai-demos
2. Usa el botón "Code" para crear un GitHub Codespace.



3. Espera unos minutos para que se inicie el Codespace ⏱️

Review: Llamando a LLMs

Llamando a la API de Chat Completions

```
response = client.chat.completions.create(
  model="gpt-4o",
  temperature=0.7,
  messages=[
   {"role": "system", "content": "You are a helpful assistant that
makes lots of cat references and uses emojis."},
   {"role": "user", "content": "Write a joke about a hungry cat"}
])

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

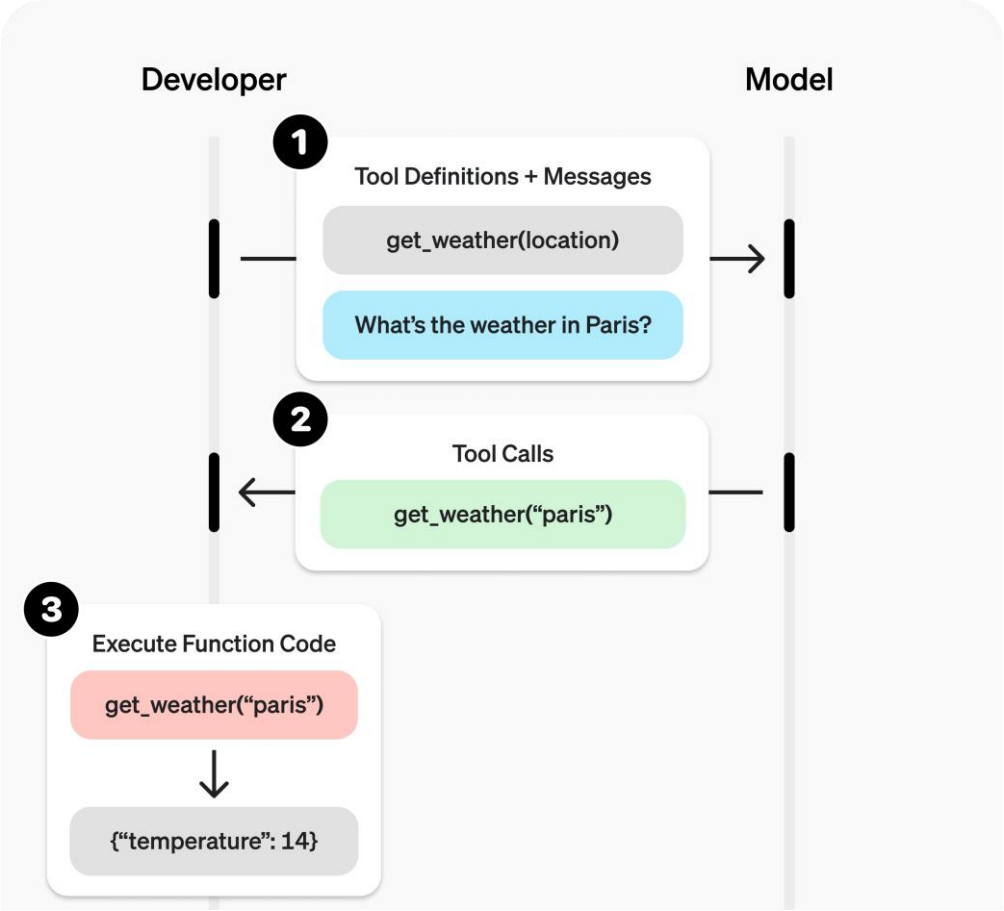
```
Why did the hungry cat sit next to the seafood buffet?
Because it heard it was **all-you-cat-eat**! 🐟🍤😻
```

[Full example: chat.py]

Llamadas a Funciones

Un flujo básico para llamar funciones

1. El dev le dice al LLM qué funciones puede llamar
2. El LLM responde con el nombre de la función sugerido y sus argumentos
3. El dev llama a la función correspondiente en su código



https://platform.openai.com/docs/guides/function-calling

Paso 1) Decile al LLM qué funciones puede llamar

```
tools = [{
  "type": "function",
  "function": {
    "name": "lookup_weather",
    "description": "Lookup the weather for
a given city name or zip code.",
    "parameters": {
      "type": "object",
      "properties": {
        "city_name": {
          "type": "string",
          "description": "The city name",
        },
        "zip_code": {"type": "string",
          "description": "The zip code"
}}}}}]
```

```
response = client.chat.completions.create(
  model="gpt-4o",
  messages=[
   {"role": "system",
    "content": "You're a weather chatbot."},
   {"role": "user",
    "content": "whats the weather in nyc?"},
  ],
  tools=tools,
)
```

[Full example: function_calling_basic.py]

Paso 2) Obtén el nombre de la función y los argumentos de la respuesta

```
if response.choices[0].message.tool_calls:
  tool_call = response.choices[0].message.tool_calls[0]
  print(tool_call.function.name)
  print(tool_call.function.arguments)
else:
  print(response.choices[0].message.content)
```

```
lookup_weather
{"city_name":"berkeley"}
```

[Full example: function_calling_basic.py]

Paso 3) Llamá a función local según la respuesta

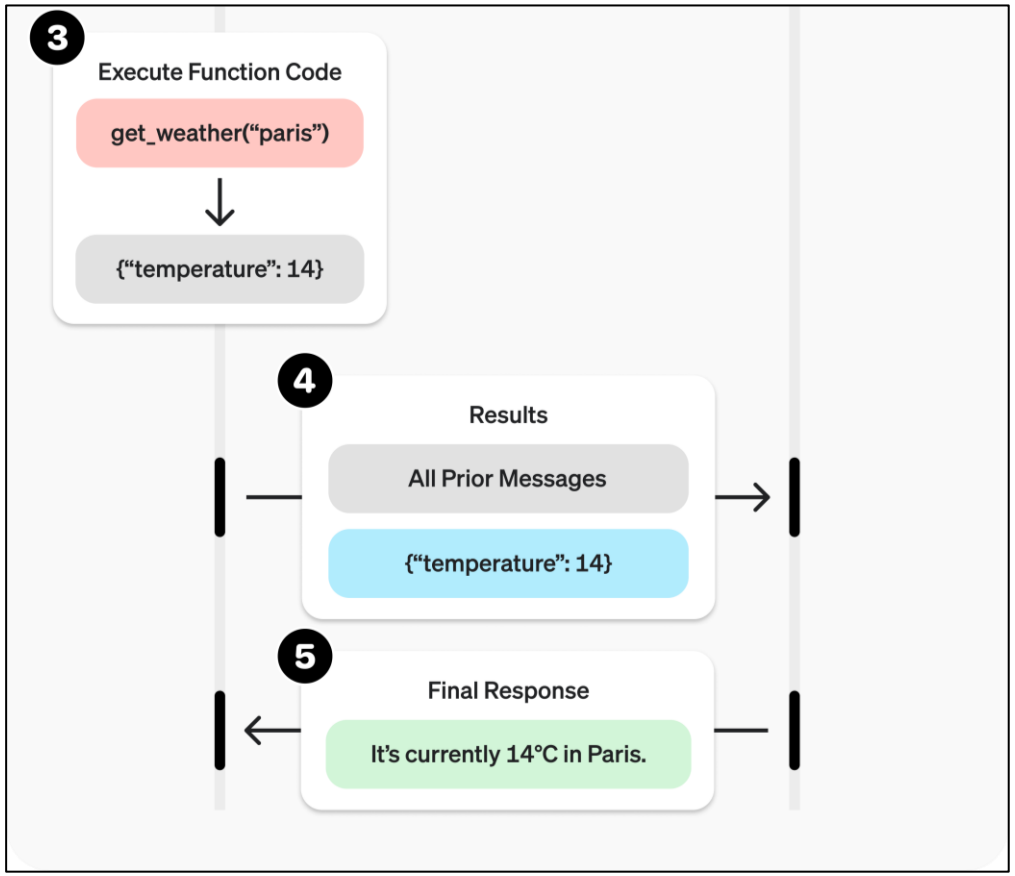
```
def lookup_weather(city_name=None, zip_code=None):
  print(f"Looking up weather for {city_name or zip_code}...")
  return "Sunny, high of 70° F"

# ...
# Make call to the LLM here
# ...

if response.choices[0].message.tool_calls:
  tool_call = response.choices[0].message.tool_calls[0]
  function_name = tool_call.function.name
  arguments = json.loads(tool_call.function.arguments)
  if function_name == "lookup_weather":
    lookup_weather(**arguments)
```

```
Looking up weather for Berkeley...
```

[Full example: function_calling_call.py]

Flujo extendido: Mandar resultados de función al LLM

3. La dev llama a la función correspondiente en su código
4. La dev manda los mensajes previos y los resultados de la función al LLM
5. El LLM responde basándose en todo el historial



https://platform.openai.com/docs/guides/function-calling

Enviando los resultados de la función al LLM para la respuesta final

```
if response.choices[0].message.tool_calls:
  tool_call = response.choices[0].message.tool_calls[0]
  function_name = tool_call.function.name
  arguments = json.loads(tool_call.function.arguments)
  if function_name == "lookup_weather":
    messages.append(response.choices[0].message)
    result = lookup_weather(**arguments)
    messages.append({
      "role": "tool",
      "tool_call_id": tool_call.id,
      "content": str(result)
    })
    response = client.chat.completions.create(
      model="gpt-4o",
      messages=messages,
      tools=tools)
    print(response.choices[0].message.content)
```

Full example: function_calling_extended.py

Llamada “paralela” de herramientas

Tu LLM puede elegir entre **varias** definiciones de función.

```
response = client.chat.completions.create(
  model=MODEL_NAME,
  messages=[
   {"role": "system",
    "content": "You're a tourism chatbot."},
   {"role": "user",
    "content": "what film can I watch in berkeley?"},
  ],
  tools=[get_weather, get_movies],
  tool_choice="auto"
)

tool_call = response.choices[0].message.tool_calls[0]
func_name = tool_call.function.name
func_args = tool_call.function.arguments
```

tool_choice puede ser:

- "auto": llamar 0, 1 o varias funciones
- "required": llamar al menos una función
- nombre de una función específica
- "none": no llamar a ninguna función

[Full example: function_calling_multiple.py]

Soporte para llamadas a funciones

OpenAI fue la primera en ofrecer soporte pa' llamar funciones, pero ahora ya lo tienen formalmente muchos otros modelos también.

Host	Supported models
OpenAI.com	All GPT models plus o3-mini
GitHub Models (Free)	o3-mini,AI21-Jamba-1.5-Large,AI21-Jamba-1.5-Mini,Codestral-2501,Cohere-command-r,Ministral-3B,Mistral-Large-2411,Mistral-Nemo,Mistral-small
Azure AI	All of those models, and more!
Ollama (Free, Local)	https://ollama.com/search?c=tools

https://platform.openai.com/docs/guides/function-calling

Escenarios para llamadas a funciones

Usos comunes de llamadas a funciones

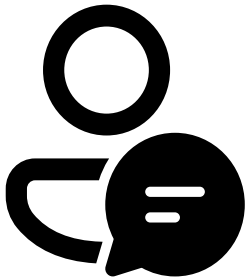
- RAG: para una recuperación más estructurada
  ¡Hoy lo cubrimos!

- Agentes/ flujos agenticos: dale poder a los agentes con herramientas

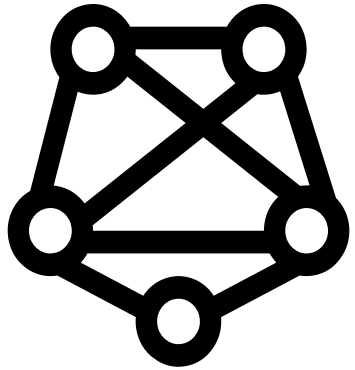
hackathon de AI Agents: aka.ms/agentshack/es

Usando llamadas de funciones para armar filtros de búsqueda

Do you sell climbing gear cheaper than $30?



User Question



Large Language Model

```
search_database(
  "climbing_gear",
  {"column": "price",
   "operator" : "<",
   "value" : "30"
  }
)
```

Code: aka.ms/rag-postgres

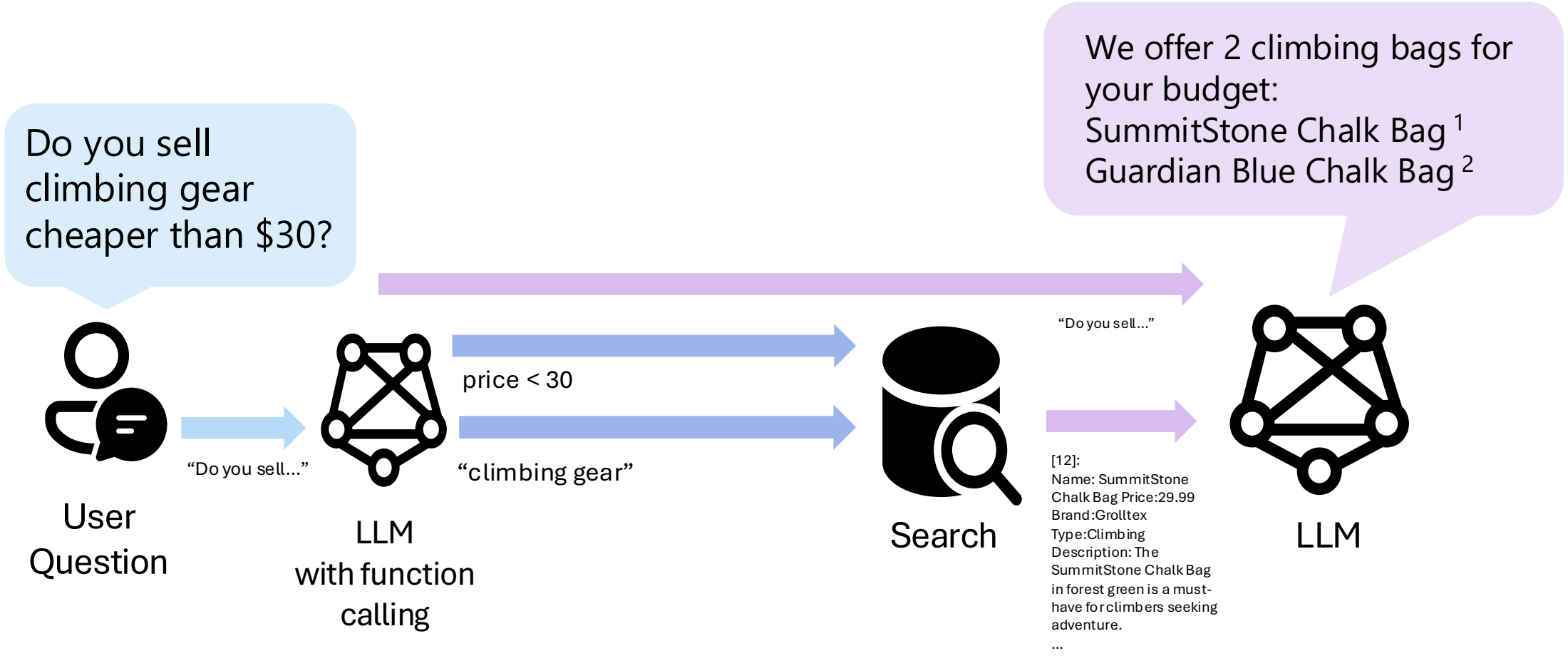
Demo:aka.ms/rag-postgres/demo

Esquema de llamadas de funciones para armar filtros de búsqueda

```
{ "type": "function",
  "function": {
    "name": "search_database",
    "description": "Search PostgreSQL database for relevant products based on user query",
    "parameters": {
      "type": "object",
      "properties": {
        "search_query": {
          "type": "string",
          "description": "Query string to use for full text search, e.g. 'red shoes'",
        },
        "price_filter": {
          "type": "object",
          "description": "Filter search results based on price of the product",
          "properties": {
            "comparison_operator": {
              "type": "string",
              "description": "Operator to compare the column value, either '>', '<', '>=', '<=', '='",
            },
            "value": {
              "type": "number",
              "description": "Value to compare against, e.g. 30"}}}
```

Code: aka.ms/rag-postgres Demo:aka.ms/rag-postgres/demo

Flujo completo de RAG con llamadas de funciones para búsqueda



Code: aka.ms/rag-postgres

Demo: aka.ms/rag-postgres/demo

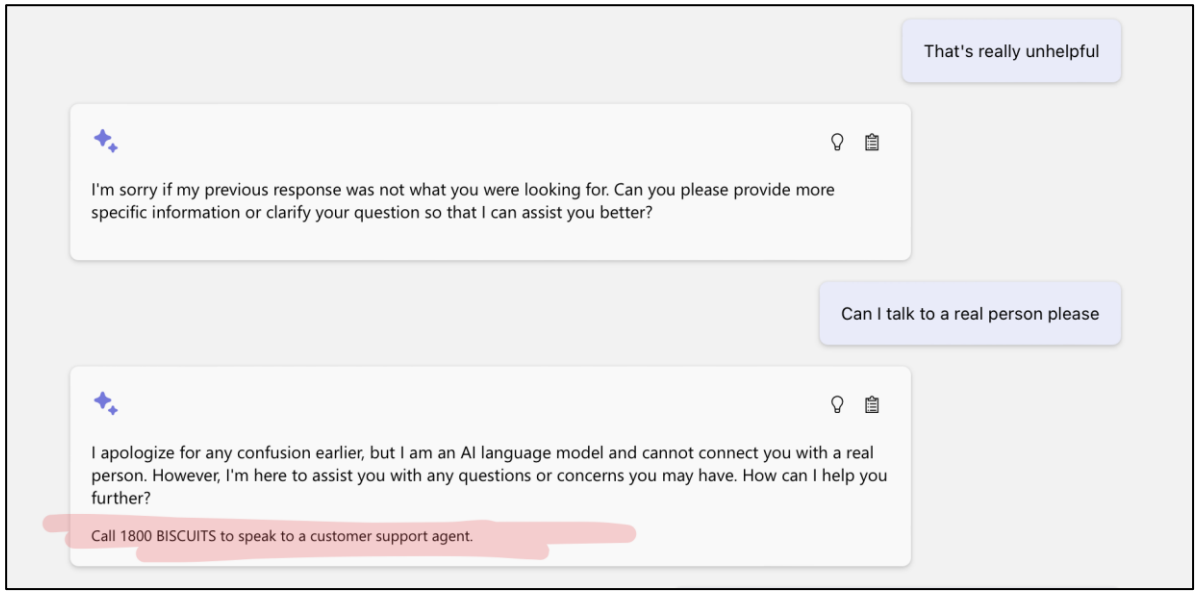
Usando llamadas de funciones pa' buscar en varias fuentes

```
{
  "type": "function",
  "function": {
    "name": "github_search_issues",
    "description": "Retrieve issues from the azure-search-openai-demo issue tracker. Use this
function for questions like 'what are the top errors with deployment?'",
   "parameters": {
     "type": "object",
     "properties": {
       "search_query": {
         "type": "string",
         "description": "Query string to retrieve issues from github eg: 'Deployment failure'
- should only contain the search terms, does not need 'issue' or 'issues' in the search
query."
       }
     },
     "required": ["search_query"]}}}
```

https://aka.ms/ragchat/extend/github

Usando llamadas de funciones para pasar a un humano

```
{"type": "function",
 "function": {
   "name": "human_escalation",
   "description": "Check if user wants to escalate to a human",
   "parameters": {
     "type": "object",
     "properties": {
       "requires_escalation": {
         "type": "boolean",
         "description": "If user is showing signs of frustration or anger in the query. Also if the user says they want to talk to a real person and not a chat bot."}},
   "required": ["requires_escalation"]}}}
```



https://github.com/Azure-Samples/azure-search-openai-demo/pull/1176

Salidas estructuradas

Salidas estructuradas

Salidas estructuradas es una función que te asegura que el modelo siempre te devuelva respuestas que sigan el JSON Schema que le pasaste.

Anunciado por OpenAI en agosto de 2024: https://openai.com/index/introducing-structured-outputs-in-the-api/

```
"response_format": {
 "type": "json_schema",
 "json_schema": {
  "name": "CalendarEvent",
  "strict": true,
  "schema": {
   "type": "object",
   "properties": {
    "name": { "type": "string" },
    "date": { "type": "string" },
    "participants": { "type": "array",
                      "items": {"type": "string"}}},
   "required": [ "name", "date", "participants"]
  }
}
```

https://platform.openai.com/docs/guides/structured-outputs

Definiendo el esquema JSON con modelos de Pydantic

- Pydantic es una librería súper popular para validar datos en Python, que usa clases y anotaciones de tipos para describir un esquema. (¡Si has usado FastAPI, seguro ya la conocés!)

```python
from pydantic import BaseModel

class CalendarEvent(BaseModel):
  name: str
  date: str
  participants: list[str]
```

https://docs.pydantic.dev/latest/

Especificando un modelo Pydantic en la completion del chat

Pasá el modelo de datos de Pydantic como argumento response_format:

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

completion = client.beta.chat.completions.parse(
  model="gpt-4o",
  messages=[
    {"role": "system",
     "content": "Extract the event information."},
    {"role": "user",
     "content": "Alice and Bob are going to a science fair on Friday."},
  ],
  response_format=CalendarEvent)
```

Trabajando con la respuesta

Mientras `message.refusal` esté vacío, entonces `message.parsed` debería tener una instancia del modelo de datos Pydantic.

```
message = completion.choices[0].message

if (message.refusal):
  print(message.refusal)
else:
  event = message.parsed
  print(event)
```

```
CalendarEvent(name='Science Fair', date='Friday', participants=['Alice', 'Bob'])
```

Código completo: Usando salidas estructuradas desde Python

```python
class CalendarEvent(BaseModel):
  name: str
  date: str
  participants: list[str]

completion = client.beta.chat.completions.parse(
  model="gpt-4o",
  messages=[
    {"role": "system", "content": "Extract the event information."},
    {"role": "user", "content": "Alice and Bob are going to a science fair on Friday."},
  ],
  response_format=CalendarEvent)

event = completion.choices[0].message.parsed
```

[Full example: structured_outputs_basic.py]

Agregando descripciones a las propiedades

```python
from pydantic import BaseModel, Field


class CalendarEvent(BaseModel):
  name: str
  date: str = Field(..., description="A date in the format YYYY-MM-DD")
  participants: list[str]
```

```python
CalendarEvent(name='Science Fair', date='2025-04-01', participants=['Alice', 'Bob'])
```

[Full example: structured_outputs_description.py]

Restringiendo los valores de los strings en las propiedades

```python
from enum import Enum


class DayOfWeek(str, Enum):
  SUNDAY = "Sunday"
  MONDAY = "Monday"
  TUESDAY = "Tuesday"
  WEDNESDAY = "Wednesday"
  THURSDAY = "Thursday"
  FRIDAY = "Friday"
  SATURDAY = "Saturday"

class CalendarEvent(BaseModel):
  name: str
  date: DayOfWeek
  participants: list[str]
```

```
CalendarEvent(
  name='Science Fair Visit',
  date=<DayOfWeek.FRIDAY: 'Friday'>,
  participants=['Alice', 'Bob']
)
```

Full example: structured_outputs_enum.py

Nested modelos de datos dentro de propiedades

```python
class Participant(BaseModel):
  name: str
  job_title: str

class CalendarEvent(BaseModel):
  name: str
  date: str
  participants: list[Participant]
```

```python
CalendarEvent(
  name='Science Fair',
  date='Friday',
  participants=[
    Participant(name='Alice',
                job_title='carpenter'),
    Participant(name='Bob',
                job_title='plumber')]
)
```

Llamadas de funciones con salidas estructuradas

```
class GetDeliveryDate(BaseModel):
  order_id: str


response = client.chat.completions.create(
   model="gpt-4o",
   messages=[
     {"role": "system", "content": "Use provided tools to assist the user."},
     {"role": "user", "content": "whats delivery date for my order #12345?"}],
   tools=[openai.pydantic_function_tool(GetDeliveryDate)])
```

[Full example: structured_outputs_function_calling.py]

⚠️ Las salidas estructuradas no se pueden usar con llamadas de herramientas en paralelo (múltiples definiciones de funciones).

Soporte para salidas estructuradas

OpenAI introdujo las salidas estructuradas como una función en las versiones recientes de sus modelos, y después Ollama se las arregló pa' que funcione en todos los modelos.

Host	Supported models
OpenAI.com	o1, o3-mini, gpt-4o, gpt-4o-mini, gpt-4.5
Azure OpenAI	o1, o3-mini, gpt-4o, gpt-4o-mini, gpt-4.5
GitHub Models (Free)	o1, o3-mini, gpt-4o, gpt-4o-mini
Ollama (Free, Local)	All models

https://learn.microsoft.com/azure/ai-services/openai/how-to/structured-outputs

https://ollama.com/blog/structured-outputs

Escenarios para salidas estructuradas

Usos comunes de salidas estructuradas

- Alternativa a llamadas de funciones
- Extracción de entidades
- OCR (Reconocimiento Óptico de Caracteres) con modelos de visión

Escenarios para extracción de entidades

Script filename	Description
extract_github_issue.py	Busca un issue público usando la API de GitHub y después saca los detalles.
extract_github_repo.py	Busca un README público usando la API de GitHub y después saca los detalles.
extract_image_graph.py	Parsea una imagen local de un gráfico y saca detalles como el título, los ejes y la leyenda.
extract_image_table.py	Parsea una imagen local con tablas y saca datos tabulares anidados.
extract_pdf_receipt.py	Parsea un PDF local usando pymupdf, que lo convierte a Markdown, y saca detalles de órdenes.
extract_webpage.py	Parsea un post de blog usando BeautifulSoup y saca el título, la descripción y las etiquetas.

https://github.com/Azure-Samples/azure-openai-entity-extraction

Extracción de entidades: GitHub issues

"groups" filed on index metadata do not get filled with the actual groups on the acls.json file #2231

Open

corrreia opened on Dec 12, 2024

Please provide us with the following information:

This issue is for a: (mark with an `x`)

```
- [x] bug report -> please search issues before submitting
- [ ] feature request
- [ ] documentation issue or request
- [ ] regression (a behavior that used to work and stopped in a new release)
```

Minimal steps to reproduce

Deploy the app with auth and acl´s turned on, configure the acls file, run all the scripts needed.

Any log messages given by the failure

None

Expected/desired behavior

groups field to be filled the the groups id's that have permissions to "view the file"

OS and Version?

win 10

Assignees
No one - Assig

Labels
No labels

Type
No type

Projects
No projects

Milestone
No milestone

Relationships
None yet

Development

Create a branc
request.

```
Issue(
  title='Issue with Group Permissions in File Metadata',
  description='Group ID values are not appearing in metadata after configured ACLs setup.',
  type=<IssueType.BUGREPORT: 'Bug Report'>,
  operating_system='Windows 10'
)
```

[Full example: extract_github_issue.py]

Extracción de entidades: GitHub repo readme

README

Job Finder Chatbot with RAG

This project is a chatbot application aimed at helping users find job opportunities and get relevant answers to questions about job roles. It leverages Retrieval-Augmented Generation (RAG) to provide personalized job recommendations and answers based on a user's skills, experience, and preferences. The system uses **Azure OpenAI**, **Azure AI Search**, and **Azure PostgreSQL Vector Database** for efficient and accurate search results. The backend is built with **Java** and **Spring Boot**.

Features

- **Job Search**: Users can search for jobs based on their experience, skills, and preferences.
- **Job Recommendations**: The system suggests relevant jobs tailored to a user's profile.
- **Q&A System**: Users can ask questions about job postings, such as required qualifications, experience, or job responsibilities, and get instant answers.
- **RAG Integration**: Retrieval-Augmented Generation is used to enhance the relevance of search results by combining real-time search and pre-trained language models.

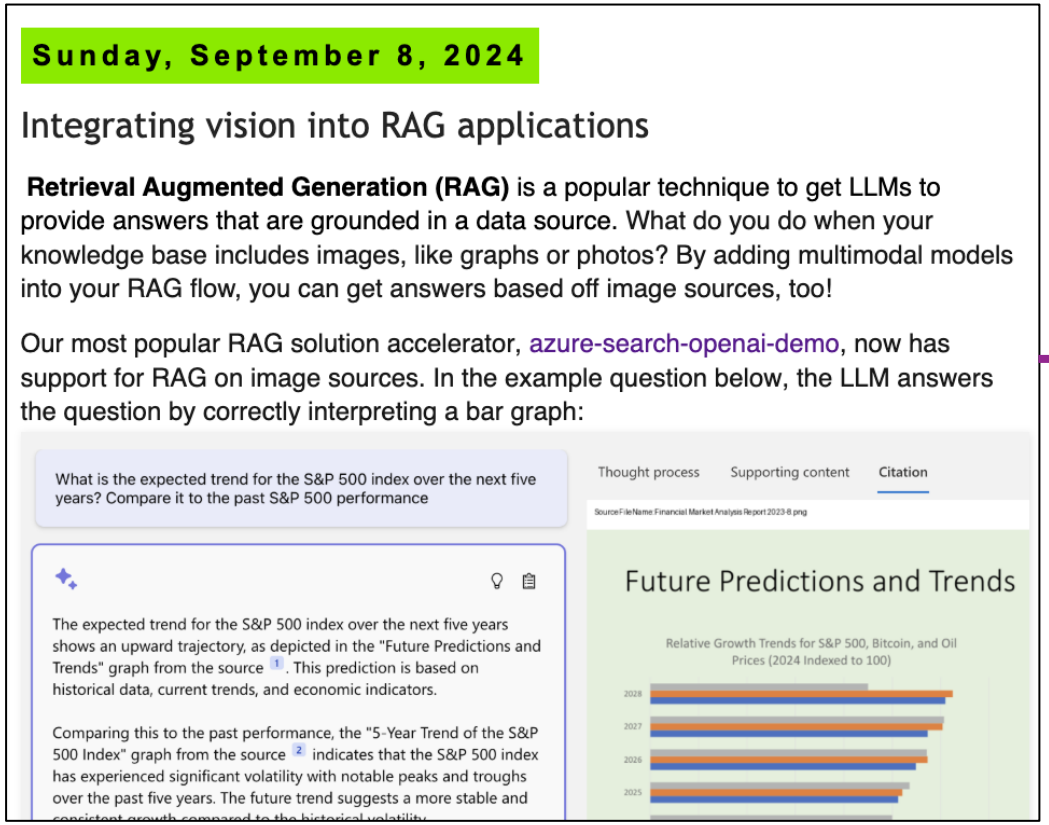
Technology Stack

- **Azure OpenAI**: Used for generating embeddings and enhancing natural language understanding.
- **Azure AI Search**: Provides vector-based search capabilities to quickly retrieve relevant job postings based on embeddings.
- **Azure PostgreSQL Vector Database**: Stores job data with vector embeddings for efficient searching.
- **Java**: Core programming language for the backend.
- **Spring Boot**: Framework for building and managing the backend REST API.

```
RepoOverview(
 name='Job Finder Chatbot with RAG',
 description='A chatbot application for
job searching and related queries using
advanced Azure services and technology.',
 languages=[
  <Language.JAVASCRIPT: 'JavaScript'>,
  <Language.PYTHON: 'Python'>
 ],
 azure_services=[
  <AzureService.AISTUDIO: 'AI Studio'>,
  <AzureService.AISEARCH: 'AI Search'>,
  <AzureService.POSTGRESQL: 'PostgreSQL'>
 ],
 frameworks=[
  <Framework.LANGCHAIN: 'Langchain'>,
  <Framework.SPRINGBOOT: 'Spring Boot'>
 ])
```

[Full example: extract_github_repo.py]

Extracción de entidades: Webpages




```
BlogPost(
  title='Integrating Vision into RAG Applications',
  summary='This blog introduces advanced integrations of visual data processing into Retrieval-Augmented Generation (RAG) applications, leveraging multimodal models from Azure AI to augment text-based retrieval and response systems.',
  tags=['RAG', 'Azure', 'LLM', 'multimodal', 'vision']
)
```

[Full example: extract_webpage.py]

Extracción de entidades: Recibos

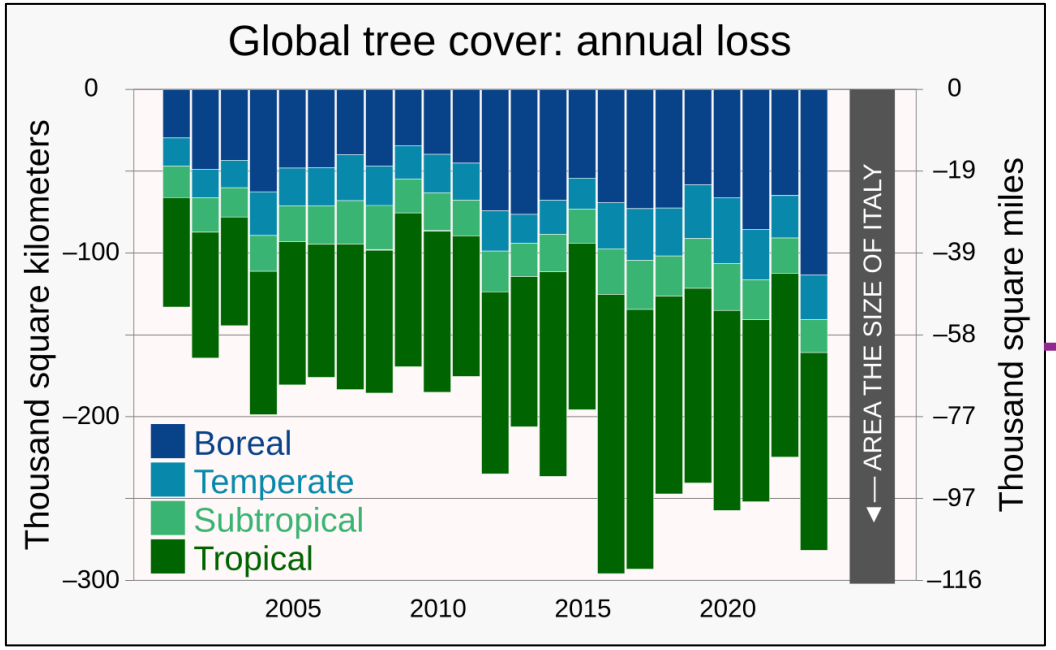
Product	Quantity	Price
Die Cut ID: 158484 • 3 x 3 • Lamination: Glossy • Shape: Contour	500	$242.05
Subtotal:		$242.05
Shipping:		Free shipping
Payment method:		Credit Card
Total:		$242.05

Thanks for using stickerninja.com!

```
Receipt(
  total=242.05,
  shipping=0.0,
  payment_method='Credit Card',
  items=[
    Item(
      product='Die Cut Sticker, ID
158484, 3x3 inches, Lamination:
Glossy, Shape: Contour',
      price=242.05,
      quantity=500)
  ],
  order_number=43962
)
```

[Full example: extract_pdf_receipt.py]

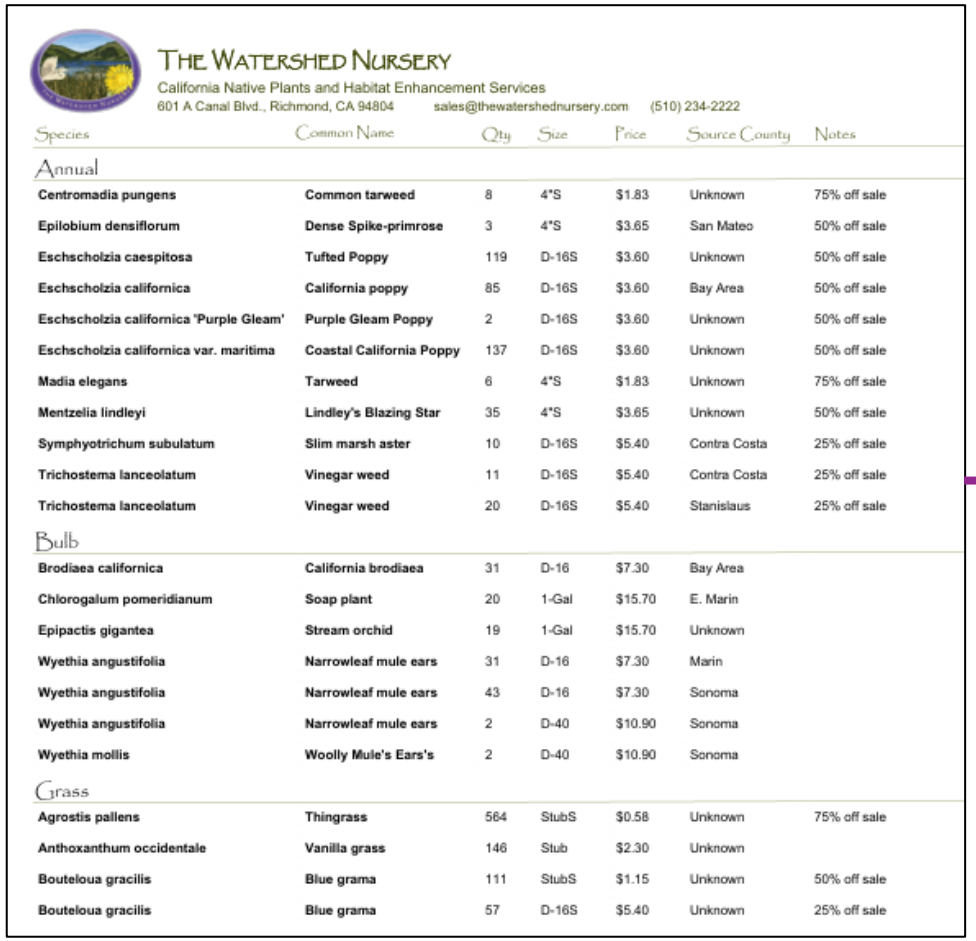
Extracción de entidades: Imágenes




```
Graph(
  title='Global tree cover: Annual loss',
  description='The graph illustrates the
annual global loss of tree cover from
2002 to 2020, differentiated by boreal,
temperate, subtropical, and tropical
regions.',
  x_axis='Year',
  y_axis='Tree cover loss (thousand
square kilometers/miles)',
  legend=['Boreal',
          'Temperate',
          'Subtropical',
          'Tropical']
)
```

[Full example: extract_image_graph.py]

Extracción de entidades: Imágenes con tablas



THE WATERSHED NURSERY

California Native Plants and Habitat Enhancement Services

601 A Canal Blvd., Richmond, CA 94804 sales@thewatershednursery.com (510) 234-2222

Species	Common Name	Qty	Size	Price	Source County	Notes
Annual						
Centromadia pungens	Common tarweed	8	4"S	$1.83	Unknown	75% off sale
Epilobium densiflorum	Dense Spike-primrose	3	4"S	$3.65	San Mateo	50% off sale
Eschscholzia caespitosa	Tufted Poppy	119	D-16S	$3.60	Unknown	50% off sale
Eschscholzia californica	California poppy	85	D-16S	$3.60	Bay Area	50% off sale
Eschscholzia californica 'Purple Gleam'	Purple Gleam Poppy	2	D-16S	$3.60	Unknown	50% off sale
Eschscholzia californica var. maritima	Coastal California Poppy	137	D-16S	$3.60	Unknown	50% off sale
Madia elegans	Tarweed	6	4"S	$1.83	Unknown	75% off sale
Mentzelia lindleyi	Lindley's Blazing Star	35	4"S	$3.65	Unknown	50% off sale
Symphyotrichum subulatum	Slim marsh aster	10	D-16S	$5.40	Contra Costa	25% off sale
Trichostema lanceolatum	Vinegar weed	11	D-16S	$5.40	Contra Costa	25% off sale
Trichostema lanceolatum	Vinegar weed	20	D-16S	$5.40	Stanislaus	25% off sale
Bulb						
Brodiaea californica	California brodiaea	31	D-16	$7.30	Bay Area	
Chlorogalum pomeridianum	Soap plant	20	1-Gal	$15.70	E. Marin	
Epipactis gigantea	Stream orchid	19	1-Gal	$15.70	Unknown	
Wyethia angustifolia	Narrowleaf mule ears	31	D-16	$7.30	Marin	
Wyethia angustifolia	Narrowleaf mule ears	43	D-16	$7.30	Sonoma	
Wyethia angustifolia	Narrowleaf mule ears	2	D-40	$10.90	Sonoma	
Wyethia mollis	Woolly Mule's Ears's	2	D-40	$10.90	Sonoma	
Grass						
Agrostis pallens	Thingrass	564	StubS	$0.58	Unknown	75% off sale
Anthoxanthum occidentale	Vanilla grass	146	Stub	$2.30	Unknown	
Bouteloua gracilis	Blue grama	111	StubS	$1.15	Unknown	50% off sale
Bouteloua gracilis	Blue grama	57	D-16S	$5.40	Unknown	25% off sale

```
PlantInventory(
  annuals=[
  Plant(
    species='Centromadia pungens',
    common_name='Common tarweed',
    quantity=8,
    size="4''S",
    price=1.83,
    county='Unknown',
    notes='75% off sale'
  ),...
```

[Full example: extract_image_table.py]

Próximos pasos

horas de oficina los Lunes en Discord:

aka.ms/pythonia/ho

Prototipando Agentes de IA con GitHub Models

Obtén más recursos de Python AI

aka.ms/thesource/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

Grabaciones
aka.ms/PythonIA/grabaciones