← Back to blog

AI LLMs ChatGPT

Jul 12, 2023 10 min read

How to use LangChain with OpenAI, Pinecone, and Apify

Use LangChain, Pinecone, and Apify to extend the capabilities of OpenAI's ChatGPT and answer questions based on data after 2021 or from any dataset.

Posted by

 **Usama Jamil**

CONTENT

OpenAI's ChatGPT has rapidly evolved into a very powerful and versatile tool. People have been using it to generate text, answer questions, translate languages, and much more. Trained on data up to September 2021, a big ChatGPT limitation is its inability to answer questions based on more recent information. So, how can we use OpenAI to create a system that can answer questions from current data? Several tools, including LangChain, Pinecone, and Apify, offer the ability to extend and enhance the capabilities of OpenAI's AI models.

1. **LangChain**: a framework designed for the development of applications that leverage language models. It allows us to integrate large language models like OpenAI and Hugging Face with different applications.
2. **Pinecone**: an external tool that allows us to save the embeddings online and extract them whenever we need.

train LLMs on real-time data and develop applications.

In this tutorial, we will extract data using one of Apify's many pre-built web scraping tools, Airbnb Scraper. We'll scrape Airbnb data from **New York City** and feed it to the LLM to make a system that will answer questions from that data.

> **Note**: You can find the complete notebook on this Google Colab link.

How to set up LangChain and Apify

To start setting up LangChain and Apify, you'll need to create a new directory and a Python file. You can do this by opening your terminal or command line and entering the following commands:

```
mkdir LangChain
cd LangChain
touch main.ipynb
```

Let's install the packages. Copy the command below, paste it into your terminal, and press Enter. These packages will provide the tools and libraries we need to develop our AI web scraping application.

Bash Copy

```
pip3 install langchain==0.0.189 pinecone-client openai tiktoken nest_asyncio apify-client chro
```

This should install all the dependencies in your system. To confirm that everything is installed properly, you can enter the following command in your terminal.

Bash Copy

```
pip freeze | egrep '(langchain|pinecone-client|openai|tiktoken|nest_asyncio|apify-client|chrom
```

This should include all the installed dependencies with their versions. If you spot any missing dependencies, you may need to re-run the installation command for that specific package.

Now, we're ready to write our code once we're done with the installation.

What is LangChain?

Find out how LangChain overcomes the limits of ChatGPT

• Apify BlogTheo Vasilis

What is LangChain?

Set up the environment variables

Python Copy

```
import os
# Add your OPENAI API key below
os.environ["OPENAI_API_KEY"] = ""
# Add your APIFY API key below
os.environ["APIFY_API_TOKEN"] = ""
```

How to use Airbnb Scraper to extract data

Now, we will use the Apify API to scrape data from Airbnb listings and then save that data into a structured format (as `Document` objects). `ApifyWrapper` from `langchain.utilities` allows us to interact with Apify, `Document` from `langchain.document_loaders.base` is used to structure the scraped data, and `json` is a standard Python library for working with JSON data.

Python Copy

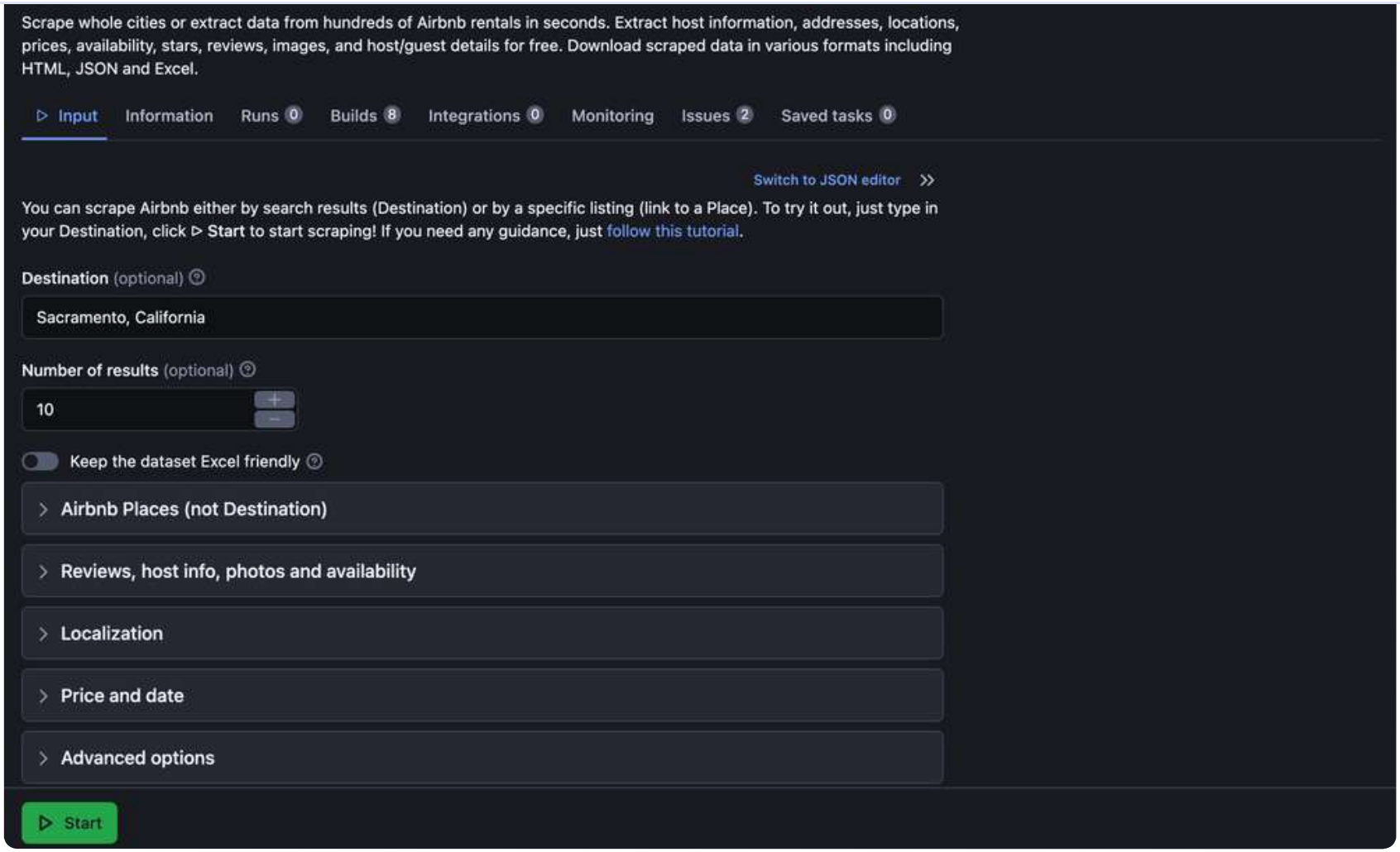
```
from langchain.utilities import ApifyWrapper
from langchain.document_loaders.base import Document
import json
```

Python Copy

```
apify = ApifyWrapper()
```

Now, we need to scrape data through Airbnb Scraper. There are two ways to use this scraper:

1. Apify Console: You can go to Apify Console and open Airbnb Scraper. From there, you can choose the data you want to select according to your needs. If you have no preferences, you can just enter the city and press **Start.**



Airbnb Scraper console page

Next, we run the Actor, which is basically our Airbnb scraper, and wait for it to finish the execution. It will fetch the results in a LangChain document loader.

Python Copy

```
loader = []
loader = apify.call_actor(
    actor_id = "dtrungtin/airbnb-scraper",
    run_input = {
        "currency": "USD",
        "maxListings":500,
        "locationQuery": "New York, ",
        "proxyConfiguration": {"useApifyProxy": True},
        "maxConcurrency": 50,
        "limitPoints": 100,
        "timeoutMs": 300000,
    },
    dataset_mapping_function = lambda item: Document(
        page_content = json.dumps({
            'name': item['name'],
            'room_type': item['roomType'],
            'city': item['city'],
            'stars': item['stars'],
            'address': item['address'],
            'number_of_guests': item['numberOfGuests'],
```

```
            'currency': item['pricing']['rate']['currency'],
        },
        'url': item['url'],
        'host': {
            'first_name': item['primaryHost']['firstName'],
            'is_superhost': item['primaryHost']['isSuperHost'],
            'about': item['primaryHost']['about'],
            'member_since': item['primaryHost']['memberSince'],
            'languages': item['primaryHost']['languages'],
            'badges': item['primaryHost']['badges']
        },
        'reviews': [
            {
                'author': {
                    'first_name': review['author']['firstName'],
                },
                'comments': review['comments'],
                'created_at': review['createdAt'],
                'rating': review['rating'],
                'localized_date': review['localizedDate']
            }
            for review in item['reviews']
        ]
    }),
metadata = {"source": item["url"]}
```

While the Actor is running, especially if it's scraping a large dataset, you can monitor its progress by going to Apify Console.

> **Note:** The Actor call function can take some time as it loads the data from the Airbnb website.

Once the execution is completed, all the data will also be saved on the Apify platform. You can learn to load the extracted data in this notebook, but we will not load data from Apify in this tutorial. In that notebook, you'll also find an explanation of the `dataset_mapping_function`, which is used to map fields from the Apify dataset records to LangChain `Document` fields.

What is generative AI?

• Apify BlogTheo Vasilis

Learn more about how generative AI works and what it means for developers

Load and process data with LangChain and Pinecone

Next, let's explore how to use LangChain for loading and processing the data we've scraped, and look at how Pinecone contributes to this process.

Get the data from the document

Now, load the data from the `page_content` property of the `Document` object. This property contains the JSON representation of the Airbnb listing. The `load()` method will parse the JSON string and create a dictionary that represents the Airbnb listing.

Python Copy

```
loader = loader.load()
```

Split the data into chunks

Due to memory limitations and computational efficiency, language models, including those used by LangChain, can only process a certain amount of text at a time. By splitting the data into smaller chunks, we can feed these chunks into the language model one at a time, which can help to improve the performance of the LLM.

Python Copy

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 1200,
    chunk_overlap  = 200
)
docs_chunks = text_splitter.split_documents(loader)
print(docs_chunks)
```

After executing this code, we will see documents with a chunk size of `1200` characters, and an overlap of `200` to maintain the semantics.

Create and save the embeddings to Pinecone

Initialize Pinecone by providing values to the `api_key` and `environment` arguments.

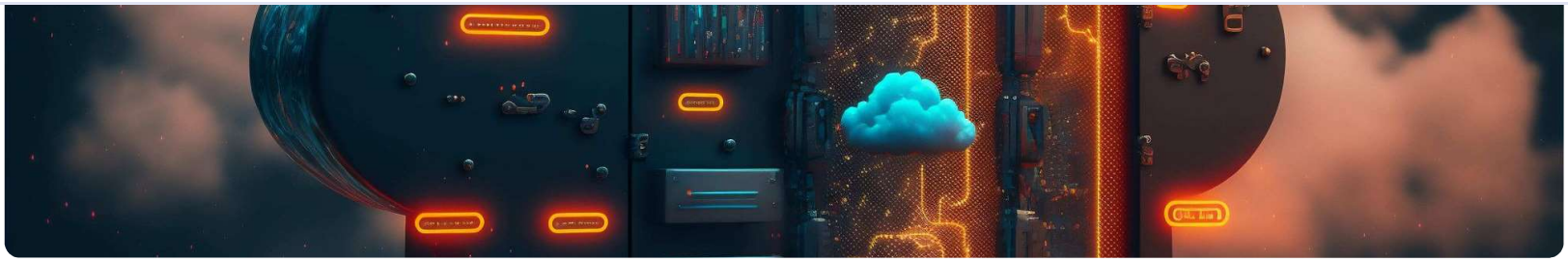
```
import pinecone
pinecone.init(
    api_key = "",
    environment = ""
)
```

Once Pinecone has been initialized, we can set up an index, which is like a database for storing and querying our embeddings. We provide a name for our index and specify the type of embeddings we're using.

Python Copy

```
from langchain.vectorstores import Pinecone
from langchain.embeddings.openai import OpenAIEmbeddings
index_name = "index_name"
embeddings = OpenAIEmbeddings()
#create a new index
docsearch = Pinecone.from_documents(docs_chunks, embeddings, index_name=index_name)
```

A new index with the name `index_name` will be created, and all the embeddings will be stored. There are other types of embeddings, but we will use `OpenAIEmbeddings()` .



What is Pinecone and why use it for LLMs?

If you're a dev working with generative AI, you need Pinecone.

• Apify BlogTheo Vasilis

What is Pinecone?

Load the embeddings

Next, we'll load the Pinecone index we created in the previous step. This means we're preparing to access and use the stored embeddings.

Python Copy

```
testingIndex = Pinecone.from_existing_index(index_name, embeddings)
```

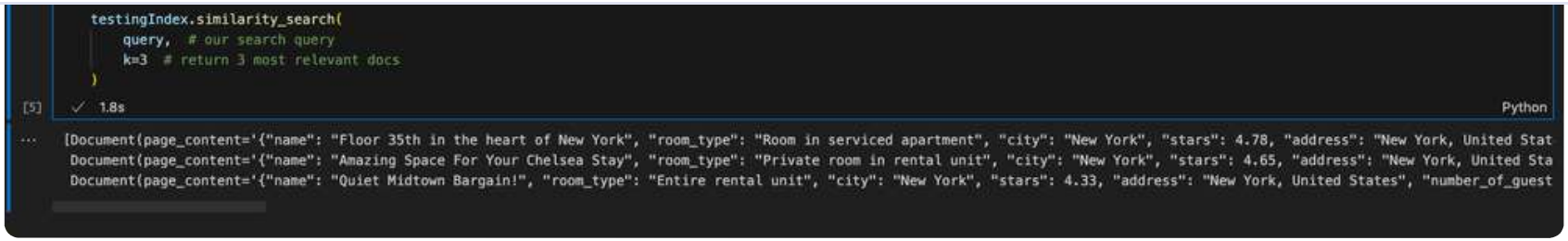
Use similarity search

Now, we have embeddings, and LangChain allows us to make a similarity search against our query. A similarity search works by calculating the distance between the embeddings of our query and the embeddings of each item in our dataset. It then returns the items with the "nearest" (most similar) embeddings.

Python Copy

```
query = "Which is the cheapest airbnb room in newyork?"
testingIndex.similarity_search(
    query,  # our search query
    k=3  # return 3 most relevant docs
)
```

It creates the embeddings and applies similarity search for the nearest distance embeddings. It then returns the top 3 results like this:



Top 3 similar results

Create a Q/A using OpenAI and LangChain

Let's start creating a Q/A system by importing the OpenAI library and creating an instance of the `ChatOpenAI` class.

Python Copy

```
from langchain.chat_models import ChatOpenAI
llm = ChatOpenAI(
    temperature = 0.0
)
```

The `temperature` parameter controls the randomness of the output. The temperature parameter controls the randomness of the output. A lower `temperature` value (like `0.0`) makes the model's

Next, we need to define a chain that will receive input and give us output as an answer.

Python Copy

```
from langchain.chains import RetrievalQA
qa = RetrievalQA.from_chain_type(
    llm = llm,
    chain_type="refine",
    retriever=vectorstore.as_retriever()
)
```

There are different types of chains provided by LangChain for specific domains, but we will use the refine chain because it loops over each document and updates the answer according to the query.

Let's make a query.

Python Copy

```
qa.run(query)
```

It will produce an output like this.

> a rate of $140 per night. You can find more information about it here.

The price may appear somewhat high, but this is because we have only scraped 500 listings, and it's the least expensive option among them all.

Web scraping for AI: how to collect data for LLMs

A tutorial that shows you how to scrape data for GenAI models.

• Apify BlogTheo Vasilis

How to collect data for LLMs tutorial

Make a more robust system with LangChain Agents

In LangChain, agents are components that have access to tools provided by us, and they can use them depending on their need. Let's add an agent to our system to make it more effective and robust.

Python Copy

```python
from langchain.agents import Tool
from langchain.agents import initialize_agent
tools = [
    Tool(
        name = 'Knowledge Base',
        func = qa.run,
        description = (
            'use this tool to answer questions'
            'more information about the topic'
        )
    )
]
agent = initialize_agent(
    tools=tools,
    llm=llm,
    verbose=True,
)
query="Which is the cheapest airbnb room in newyork?"
agent(query)
```

```
> Entering new AgentExecutor chain...
I need to find information about the prices of Airbnb rooms in New York.
Action: Knowledge Base
Action Input: "Airbnb room prices in New York"
Observation: Here are the prices for the Airbnb rooms in New York:

1. Amazing Space For Your Chelsea Stay: $494 per night. You can find more information [here](https://www.airbnb.com/rooms/17879131).

2. Floor 35th in the heart of New York: $1245 per night. You can find more information [here](https://www.airbnb.com/rooms/7987209).

3. Bright Artsy Eclectic 1 Bedroom: $499 per night. You can find more information [here](https://www.airbnb.com/rooms/53095529).

4. Lovely Central Park Apartment with direct views: $3500 per night. You can find more information [here](https://www.airbnb.com/rooms/53093610).

Please note that these prices are subject to change and may vary depending on the dates of your stay.
Thought:The cheapest Airbnb room in New York is the "Bright Artsy Eclectic 1 Bedroom" at $499 per night.
Final Answer: The cheapest Airbnb room in New York is the "Bright Artsy Eclectic 1 Bedroom" at $499 per night.

> Finished chain.

{'input': 'Which is the cheapest airbnb room in newyork?',
 'output': 'The cheapest Airbnb room in New York is the "Bright Artsy Eclectic 1 Bedroom" at $499 per night.'}
```

The thinking process of an Agent

If it's set to `False`, it will just return the `input` with `output`.

> **Note**: We have defined just one tool. You can define more depending on your needs.

Add memory to the agent

By default, all the conversations in the agents and chains are volatile; they don't remember anything. But we can add memory to save the previous conversations. Let's do that.

Python Copy

```
from langchain.chains.conversation.memory import ConversationBufferWindowMemory
conversational_memory = ConversationBufferWindowMemory(
    memory_key='chat_history',
    k = 5,
    return_messages = True
)
```

Let's update our agent and add memory to it.

Python Copy

```
newAgent = initialize_agent(
    agent = 'chat-conversational-react-description',
    tools = tools,
    llm = llm,
    verbose = True,
    max_iterations = 3,
    early_stopping_method = 'generate',
    memory = conversational_memory
```

```
newAgent(query)
```

After adding ConversationBufferWindowMemory to the agent, it'll remember the last five questions that we asked. That's because we have set the limit `k=5`. Now let's ask another question.

Python Copy

```
query = "What is my name?"
newAgent(query)
```

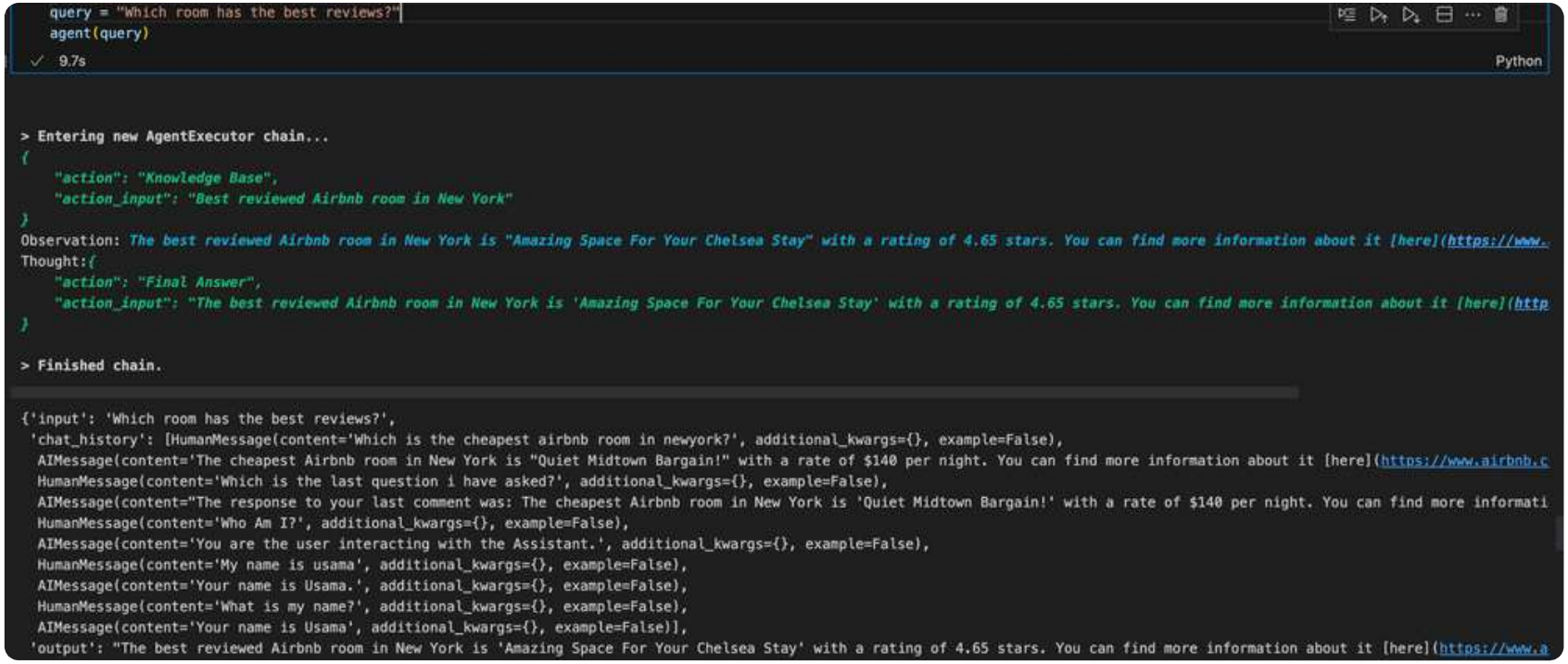
It will return a result like this:

```
> Entering new AgentExecutor chain...
{
    "action": "Final Answer",
    "action_input": "Your name is Apify"
}

> Finished chain.

{'input': 'What is my name?',
 'chat_history': [HumanMessage(content='My name is Apify', additional_kwargs={}, example=False),
  AIMessage(content='Nice to meet you, Apify! How can I assist you today?', additional_kwargs={}, example=False)],
 'output': 'Your name is Apify'}
```

Results after adding memory

A few of the sample queries and their answers are attached below.

First, I asked the agent to show me the room with the best reviews:



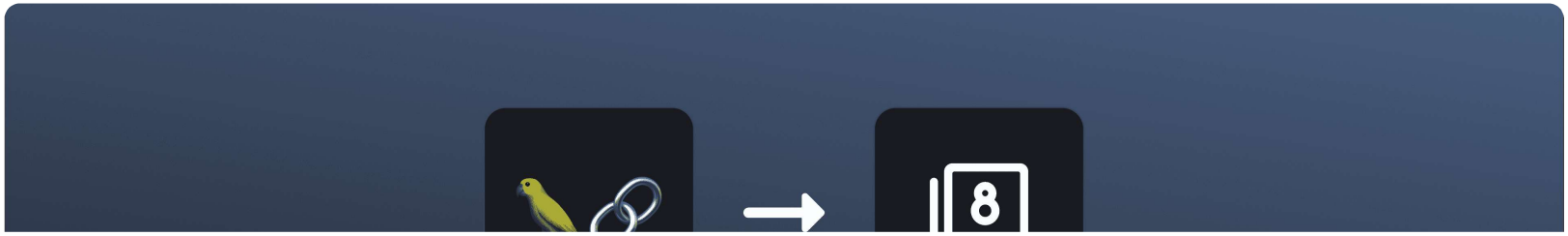
Room with best reviews

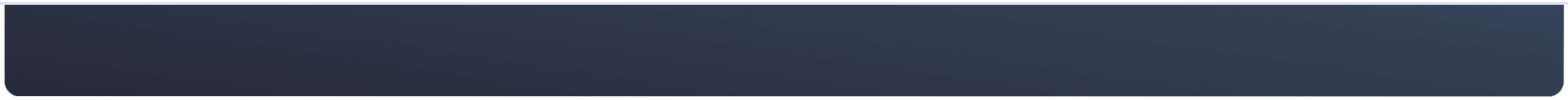
Then I asked the agent to give me the name of the owner:

```
> Entering new AgentExecutor chain...
{
    "action": "Knowledge Base",
    "action_input": "Who is the owner of 'Amazing Space For Your Chelsea Stay'? What is the price of the room? Can you provide some reviews of the room?"
}
Observation: The owner of 'Amazing Space For Your Chelsea Stay' is Pablo. The price of the room is $494 USD. Here are some reviews of the room:

1. Karin - "Pablo's apartment was perfect for our girls trip. He's a great host and was very responsive and helpful. Checking in and out was a breeze. We loved the neighborhood and it

2. Christina - "Great place to stay in Chelsea! So many great restaurants and shops in the area. I would definitely stay there again. Easy check-in/out process."

3. Sujata - "Great location, well thought out and clean."

4. Rebecca - "Beautiful apartment in a great location!"

5. Francesca - "Thank you for great communication and a clean space for so many people!"

6. Jessica - "Great location close to the Subway. Many shops and restaurants close by. Chelsea's Market is a short walk and great place to hang out. The place is spacious with plenty
Thought:{
    "action": "Final Answer",
    "action_input": "The owner of 'Amazing Space For Your Chelsea Stay' is Pablo. The price of the room is $494 USD. Here are some reviews of the room:\n\n1. Karin - 'Pablo's apartmen
}

> Finished chain.
```

Owner of the room

And that's how you can ask a dataset any question you like using LLMs when combined with LangChain and Pinecone!





□□ LangChain alternatives

8 open-source options that might suit your LLM needs.

• Apify BlogTheo Vasilis

LangChain alternatives

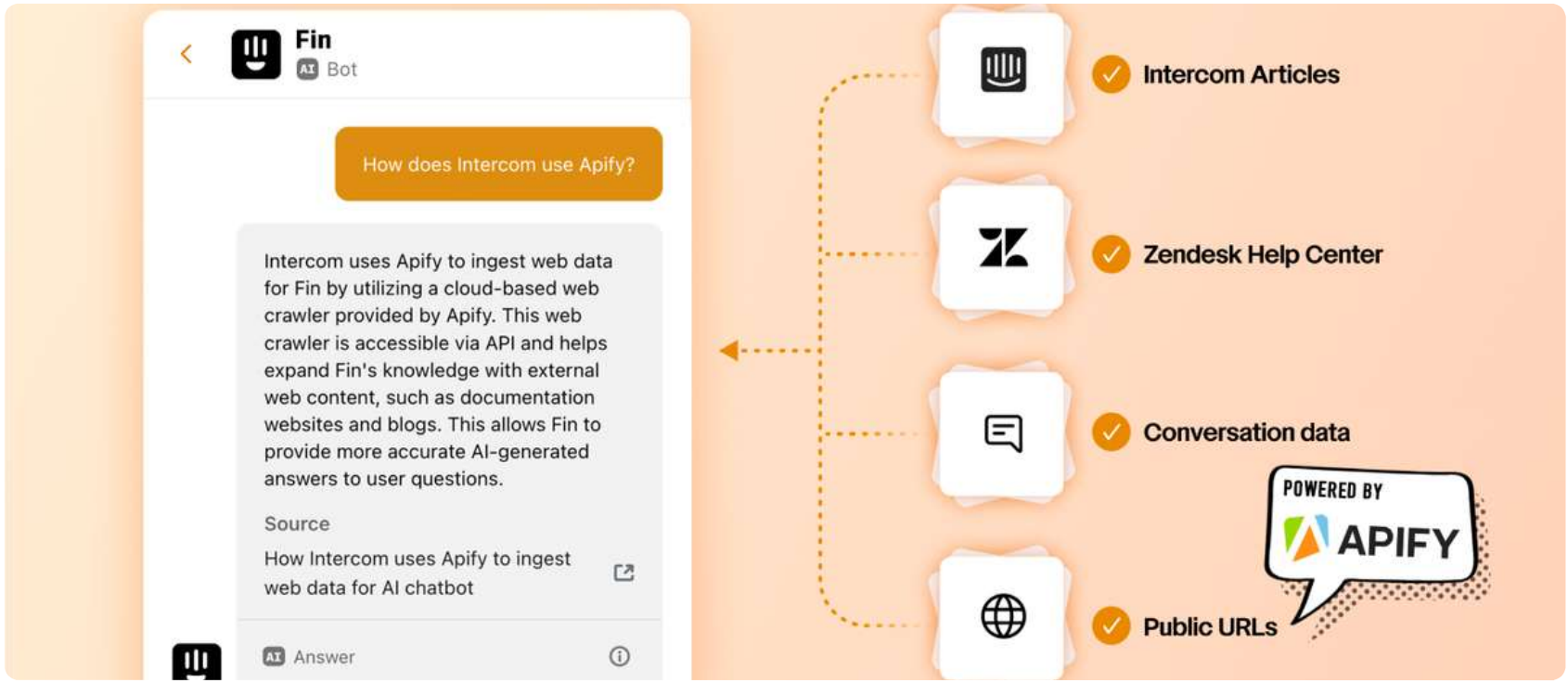
Doing more with OpenAI, LangChain, and Apify

This simple application of LangChain and Apify barely scratches the surface of what you can achieve by combining these tools. You can do a lot more by choosing any Apify Actor, integrating with OpenAI through LangChain, and developing any application you need.

If you want to see what other GPT and AI-enhanced tools you can integrate with LangChain, just check out Apify Store.

TAGS

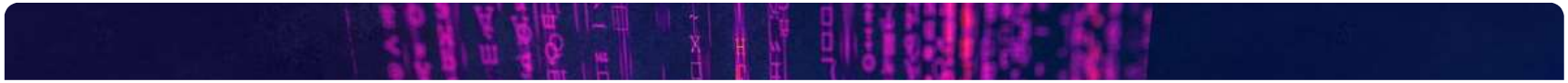
RELATED ARTICLES



AI Use cases

How Intercom uses Apify to feed web data to its AI chatbot for customer support

David Barton Aug 15, 2023



AI Web scraping LLMs

Web scraping for AI: how to collect data for LLMs

Theo Vasilis Aug 10, 2023

Use cases Web scraping AI

How web scraping and AI are helping to find missing children

 **Theo Vasilis** Aug 7, 2023

USEFUL LINKS

Apify Academy
Apify Docs
Find ready-made scrapers
Sign up for a free Apify account
Get a custom solution
Careers

POPULAR ARTICLES

What is web scraping?
Pros and cons of web scraping
Is web scraping legal?
Scrape Instagram posts, comments, and photos
Web scraping with Python
Web scraping in Node.js with Axios and Cheerio

Web scraping: how to crawl without getting blocked