Example of ChatBot

> In this project, I will show how you can easily create your own ***Q/A*** customer support chat bot. Also will provide some helpful functions that could be useful in your projects, or tasks in AI field.

- Part.1: Data Extraction
- Part.2: Function tools
- Part.3: ChatBot

In [1]:

```
#Importing necessary packages
from google.cloud import bigquery
import pandas as pd
import numpy as np
import requests
import datetime
import time
from google.oauth2.service_account import Credentials
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.text_splitter import CharacterTextSplitter, Recursiv
from langchain.vectorstores import DocArrayInMemorySearch, Chroma
from langchain.document_loaders import TextLoader, PyPDFLoader, Web
from langchain.chains import RetrievalQA, ConversationalRetrievalCh
from langchain.memory import ConversationBufferMemory
from langchain.chat_models import ChatOpenAI
import jsonlines
import  jpype
import  asposecells
jpype.startJVM()
from asposecells.api import Workbook
from pydantic import BaseModel, Field
import os
import openai
import sys
sys.path.append('../..')
from dotenv import load_dotenv, find_dotenv
from langchain.utils.openai_functions import convert_pydantic_to_op
import panel as pn
import param
from langchain.prompts import ChatPromptTemplate, MessagesPlacehold
from langchain.output_parsers.openai_functions import JsonOutputFun
from typing import Optional, List
from langchain.schema.runnable import RunnableLambda, RunnablePasst
from langchain.tools import tool
import wikipedia
from langchain.tools.render import format_tool_to_openai_function
from langchain.agents import AgentExecutor
from langchain.agents.format_scratchpad import format_to_openai_fun
from langchain.agents.output_parsers import OpenAIFunctionsAgentOut
#Calling for extension
pn.extension()
```

In [2]:

```
#Connect to OpenAI
_ = load_dotenv(find_dotenv()) #read local .env file

openai.api_key = os.environ['OPENAI_API_KEY']
```

In [3]:

```
#Loading access key
key_path = "serv_acc_key.json"
#Create credentials object
credentials = Credentials.from_service_account_file(
    key_path,
    scopes = ['https://www.googleapis.com/auth/cloud-platform']
)
if credentials.expired:
    credentials.refresh(Request())
```

# Part 1: Data Exctraction

- 1.1. The first example will show how to extract data from Google Cloud.
- 1.2. Second, how you can fetch data from Wikipedia depends on your request.
- 1.3. Example of working with Weaviate vector database

1.1. Exctraction from Google Cloud

In [4]:

```
#Connect to BigQuery
PROJECT_ID = 'praxis-window-402615'
def run_bq_query(sql):

    # Create BQ client
    bq_client = bigquery.Client(project = PROJECT_ID,
                                credentials = credentials)

    # Try dry run before executing query to catch any errors
    job_config = bigquery.QueryJobConfig(dry_run=True,
                                         use_query_cache=False)
    bq_client.query(sql, job_config=job_config)

    # If dry run succeeds without errors, proceed to run query
    job_config = bigquery.QueryJobConfig()
    client_result = bq_client.query(sql,
                                    job_config=job_config)

    job_id = client_result.job_id

    # Wait for query/job to finish running. then get & return data
    df = client_result.result().to_arrow().to_pandas()
    print(f"Finished job_id: {job_id}")
    return df
```

In [5]:

```python
#define list of programming language tags we want to query
language_list = ["python","html","r","css"]
so_df = pd.DataFrame()

for language in language_list:

    print(f"generating {language} dataframe")

    query = f"""
    SELECT
        CONCAT(q.title, q.body) as question,
        a.body AS answer
    FROM
        `bigquery-public-data.stackoverflow.posts_questions` q
    JOIN
        `bigquery-public-data.stackoverflow.posts_answers` a
    ON
        q.accepted_answer_id = a.id
    WHERE
        q.accepted_answer_id IS NOT NULL AND
        REGEXP_CONTAINS(q.tags, "{language}") AND
        a.creation_date >= "2020-01-01"
    LIMIT
        500
    """



    language_df = run_bq_query(query)
    language_df["category"] = language
    so_df = pd.concat([so_df, language_df],
                      ignore_index = True)
```

```
generating python dataframe
Finished job_id: 6c6df500-60fd-4cef-8977-8edac9fcf868
generating html dataframe
Finished job_id: 7079ce90-110f-4497-95d7-283b7b4bb936
generating r dataframe
Finished job_id: 144ea7d4-ebb5-46a4-a44b-2c02a810d01c
generating css dataframe
Finished job_id: ff789b8b-ba0b-41f6-a1ac-5b2b68ee060f
```

```
In [6]: #Checking what we have
        display(so_df)
```

	question	answer	category
0	Edit a value after the dataframe has been sepe...	<p>It should update studydata df rather than q...	python
1	Unable to use ManyToMany Field with Django and...	<p>M2M relationship means - you have many obje...	python
2	Can't access table and table elements using bs...	<p>Content is stored in script tag and rendere...	python
3	python plotly: how to display additional label...	<p>You can use <a href="https://plotly.com/pyt...	python
4	How can I automatically generate an expression...	<pre class="lang-py prettyprint-override"><cod...	python
...	...	...	...
1995	How to create inner and outside shadow effect ...	<p>Can you please check the below code? Hope i...	css
1996	Limit total maximum lines in two divs<p>I know...	<p><a href="https://developer.mozilla.org/en-U...	css
1997	justify-content works for one div and not for ...	<p>When using <a href="https://developer.mozil...	css
1998	Connect "Speech to Text" with the Input Type t...	<p>I am not aware how you do it - but this wor...	css
1999	Bootstrap 5, full viewport + responsive bg<p>I...	<p>i found issue you have to remove <code>vh-1...	css

2000 rows × 3 columns

1.2. Fetching from Wikipedia

In [7]:

```
#Creating a function that will fetch relevant information from wiki
@tool
def search_wikipedia(query: str) -> str:
    """Run Wikipedia search and get page summaries."""#Creating des
    page_titles = wikipedia.search(query)#Scrap relevant informatio
    summaries = []
    for page_title in page_titles[: 3]:
        try:
            wiki_page =  wikipedia.page(title=page_title, auto_sugg
            summaries.append(f"Page: {page_title}\nSummary: {wiki_p
        except (
            self.wiki_client.exceptions.PageError,
            self.wiki_client.exceptions.DisambiguationError,
        ):
            pass
    if not summaries:
        return "No good Wikipedia Search Result was found"
    return "\n\n".join(summaries)
```

In [8]:

```
#Checking if all working properly
search_wikipedia({"query": "Machine Learning"})
```

Out[8]:

```
'Page: Machine learning\nSummary: Machine learning (ML) is a field
of study in artificial intelligence concerned with the development
and study of statistical algorithms that can effectively generaliz
e and thus perform tasks without explicit instructions. Recently,
generative artificial neural networks have been able to surpass ma
ny previous approaches in performance. Machine learning approaches
have been applied to large language models, computer vision, speec
h recognition, email filtering, agriculture, and medicine, where i
t is too costly to develop algorithms to perform the needed tasks.
The mathematical foundations of ML are provided by mathematical op
timization (mathematical programming) methods. Data mining is a re
lated (parallel) field of study, focusing on exploratory data anal
ysis through unsupervised learning.ML is known in its application
across business problems under the name predictive analytics. Alth
ough not all machine learning is statistically based, computationa
l statistics is an important source of the field\'s methods.\n\nPa
ge: Quantum machine learning\nSummary: Quantum machine learning is
the integration of quantum algorithms within machine learning prog
rams.The most common use of the term refers to machine learning al
gorithms for the analysis of classical data executed on a quantum
computer, i.e. quantum-enhanced machine learning. While machine le
arning algorithms are used to compute immense quantities of data,
quantum machine learning utilizes qubits and quantum operations or
specialized quantum systems to improve computational speed and dat
a storage done by algorithms in a program. This includes hybrid me
thods that involve both classical and quantum processing, where co
mputationally difficult subroutines are outsourced to a quantum de
vice. These routines can be more complex in nature and executed fa
ster on a quantum computer. Furthermore, quantum algorithms can be
```

```
used to analyze quantum states instead of classical data.Beyond qu
antum computing, the term "quantum machine learning" is also assoc
iated with classical machine learning methods applied to data gene
rated from quantum experiments (i.e. machine learning of quantum s
ystems), such as learning the phase transitions of a quantum syste
m or creating new quantum experiments.Quantum machine learning als
o extends to a branch of research that explores methodological and
structural similarities between certain physical systems and learn
ing systems, in particular neural networks. For example, some math
ematical and numerical techniques from quantum physics are applica
ble to classical deep learning and vice versa.Furthermore, researc
hers investigate more abstract notions of learning theory with res
pect to quantum information, sometimes referred to as "quantum lea
rning theory".\n\n\n\nPage: Boosting (machine learning)\nSummary: 
In machine learning, boosting is an ensemble meta-algorithm for pr
imarily reducing bias, and also variance in supervised learning, a
nd a family of machine learning algorithms that convert weak learn
ers to strong ones. Boosting is based on the question posed by Kea
rns and Valiant (1988, 1989): "Can a set of weak learners create a
single strong learner?" A weak learner is defined to be a classifi
er that is only slightly correlated with the true classification (
it can label examples better than random guessing). In contrast, a
strong learner is a classifier that is arbitrarily well-correlated
with the true classification.\nRobert Schapire\'s affirmative answ
er in a 1990 paper to the question of Kearns and Valiant has had s
ignificant ramifications in machine learning and statistics, most 
notably leading to the development of boosting.When first introduc
ed, the hypothesis boosting problem simply referred to the process
of turning a weak learner into a strong learner. "Informally, [the
hypothesis boosting] problem asks whether an efficient learning al
gorithm […] that outputs a hypothesis whose performance is only sl
ightly better than random guessing [i.e. a weak learner] implies t
he existence of an efficient algorithm that outputs a hypothesis o
f arbitrary accuracy [i.e. a strong learner]." Algorithms that ach
ieve hypothesis boosting quickly became simply known as "boosting"
. Freund and Schapire\'s arcing (Adapt[at]ive Resampling and Combi
ning), as a general technique, is more or less synonymous with boo
sting.'
```

> Now that you have two functions that can be used for data extraction. Next, you need to pass this data into Vectorstore or Vectordb or you can simply save it in JSON or pdf file on your computer. Here is one example of how you can fast and easily store your data on your computer

In [9]:

```
#Common way of storing your data to json format
df = so_df.to_dict()
with jsonlines.open(f'qa.json','w') as writer:
    writer.write_all(df)
```

In [10]:

```
#Converting our Json file to PD
workbook = Workbook("qa.json")
workbook.save("qa.pdf")
jpype.shutdownJVM()
```

In [11]:

```
#Function that will load your pdf file and create a Vector index st
def load_db(file,chain_type, k):
    #load documents
    loader=PyPDFLoader(file)
    documents = loader.load()
    #split documents
    text_splitter = RecursiveCharacterTextSplitter(chunk_size = 100
    docs = text_splitter.split_documents(documents)
    #define embedding
    embeddings = OpenAIEmbeddings()
    # create vector database from data
    db = DocArrayInMemorySearch.from_documents(docs,embeddings)
    #define retriever
    retriever = db.as_retriever(search_type ="similarity",search_kw
    #Create a chatbot chain. Memory is managed externally
    qa = ConversationalRetrievalChain.from_llm(
        llm = ChatOpenAI(model="gpt-3.5-turbo",temperature=0),
        chain_type=chain_type,
        retriever=retriever,
        return_source_documents = True,
        return_generated_question = True,
    )

    return qa
```

1.3. Weaviate DataBase

In [12]:

```
#Download sample data
import requests
import json

#Download data
resp = requests.get('https://raw.githubusercontent.com/weaviate-tut
data = json.loads(resp.text) #load data

#Create an embedded instance of weaviate vector database
import weaviate,os,openai
from weaviate import EmbeddedOptions
from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv())

auth_config = weaviate.AuthApiKey(api_key=os.getenv("WEAVIATE_API_K
openai.api_key = os.environ['OPENAI_API_KEY']

client = weaviate.Client(
    url= os.getenv("WEAVIATE_API_URL"),
    auth_client_secret = auth_config,
    additional_headers = {
        "X-Cohere-Api-Key": os.getenv("COHERE_API_KEY"),
        "X-OpenAI-Api-Key": openai.api_key

    }
)
```

In [13]:

```
#Create Question collection
#reseting the schema
if client.schema.exists("Question"):
    client.schema.delete_class("Question")
class_obj = {
    "class":"Question",
    "vectorizer":"text2vec-openai",
    "moduleConfig":{
        "model":"ada",
        "modelVersion":"002",
        "type":"text"
    }
}

client.schema.create_class(class_obj)
```

In [14]:

```
def json_print(data):
    print(json.dumps(data,indent=2))

#Load sample data and generate vector embeddings
with client.batch.configure(batch_size=5) as batch:
    for i, d in enumerate(data): #Batch impor tdata
        print(f"importing question: {i + 1}")

        properties = {
            "answer": d["Answer"],
            "question": d["Question"],
            "category": d["Category"]
        }

        batch.add_data_object(
            data_object = properties,
            class_name = "Question"
        )

count = client.query.aggregate("Question").with_meta_count().do()
json_print(count)
```

```
importing question: 1
importing question: 2
importing question: 3
importing question: 4
importing question: 5
importing question: 6
importing question: 7
importing question: 8
importing question: 9
importing question: 10
{
  "data": {
    "Aggregate": {
      "Question": [
        {
          "meta": {
            "count": 10
          }
        }
      ]
    }
  }
}
```

In [15]:
```
response=(
    client.query
    .get("Question",["question","answer","category"])
    .with_near_text({"concepts":"biology"})
    .with_additional('distance')
    .with_limit(2)
    .do()
)

json_print(response)
```

```
{
  "data": {
    "Get": {
      "Question": [
        {
          "_additional": {
            "distance": 0.19695163
          },
          "answer": "DNA",
          "category": "SCIENCE",
          "question": "In 1953 Watson & Crick built a model of the
molecular structure of this, the gene-carrying substance"
        },
        {
          "_additional": {
            "distance": 0.20142835
          },
          "answer": "species",
          "category": "SCIENCE",
          "question": "2000 news: the Gunnison sage grouse isn't j
ust another northern sage grouse, but a new one of this classifica
tion"
        }
      ]
    }
  }
}
```

Conclusion: So now after you get two example how you can on a simple wat get a necessary data work with your chat let's show several tools that can be helpful for you, when you will build your customer chatbots

Part 2: Function tools

> Also you can create a different function depending on what kind of question the user will ask

- Here I will create two different function
- And show you how with the help of **pydantic** library you can build your function in a faster and simplest way

In [16]:

```
#This function will fetch all relevant information about machine le
class ML_tools_search(BaseModel):
    """Call this with a Machine learning code to extract any releva
    machine_learning: str = Field(description="machine learning cod
ML_function = convert_pydantic_to_openai_function(ML_tools_search)
ML_function
```

Out[16]:

```
{'name': 'ML_tools_search',
 'description': 'Call this with a Machine learning code to extract
any relevant information about Machine learning',
 'parameters': {'title': 'ML_tools_search',
  'description': 'Call this with a Machine learning code to extrac
t any relevant information about Machine learning',
  'type': 'object',
  'properties': {'machine_learning': {'title': 'Machine Learning',
    'description': 'machine learning code  to get information abou
t',
    'type': 'string'}},
  'required': ['machine_learning']}}
```

In [17]:

```
#Second function will fetch all relevant information about artist

class ArtistSearch(BaseModel):
    """Call this to get the name of songs by a particular artist"""
    artist_name: str= Field(description="name of artist to look up")
    n: int = Field(description="number of results")

artist_function = convert_pydantic_to_openai_function(ArtistSearch)
artist_function
```

Out[17]:

```
{'name': 'ArtistSearch',
 'description': 'Call this to get the name of songs by a particula
r artist',
 'parameters': {'title': 'ArtistSearch',
  'description': 'Call this to get the name of songs by a particul
ar artist',
  'type': 'object',
  'properties': {'artist_name': {'title': 'Artist Name',
    'description': 'name of artist to look up',
    'type': 'string'},
   'n': {'title': 'N', 'description': 'number of results', 'type':
'integer'}},
  'required': ['artist_name', 'n']}}
```

```
In [18]: #Insert all our function to one variable
         functions = [
             ML_function,
             artist_function
         ]
         #Cre
         model = ChatOpenAI()
         model_with_functions = model.bind(functions=functions)
         model_with_functions.invoke("What type of ML most popular?")
```

```
Out[18]: AIMessage(content='The popularity of different types of ML can var
         y depending on the context and industry. However, some of the most
         popular types of ML techniques include:\n\n1. Supervised Learning:
         This is the most commonly used type of ML where the model is train
         ed on labeled data to make predictions or classifications.\n2. Uns
         upervised Learning: In this type, the model learns patterns and re
         lationships in the data without any predefined labels.\n3. Deep Le
         arning: This is a subset of ML that focuses on training artificial
         neural networks with multiple layers to learn and make predictions
         .\n4. Reinforcement Learning: This type of ML involves an agent le
         arning through trial and error interactions with an environment to
         maximize rewards.\n5. Natural Language Processing (NLP): NLP is a
         field of ML that focuses on the interaction between computers and
         human language, enabling machines to understand, interpret, and ge
         nerate human language.\n6. Computer Vision: Computer vision is an
         ML technique that enables machines to recognize and interpret visu
         al data, such as images and videos.\n7. Transfer Learning: Transfe
         r learning allows models to leverage knowledge gained from one tas
         k to improve performance on a different but related task.\n\nThese
         are just a few examples, and there are many other types and subfie
         lds of ML that are popular and widely used in various applications
         .')
```

```
In [19]: model_with_functions.invoke("what are the most popular songs by Bon
```

```
Out[19]: AIMessage(content='', additional_kwargs={'function_call': {'name':
         'ArtistSearch', 'arguments': '{\n  "artist_name": "Bon Jovi",\n  "
         n": 5\n}'}})
```

Conclusion: So By the previous line, you can see that when we called our model to find information about our artist the model called for our **ArtistSearch** function this is one example of using Functions depending on what kind of operations users ask of your model. Let's create a workable chain that will provide reasons and answer with help of functions that we have created previously

In [20]:
```
#Exaple of using prompt + model + output
prompt = ChatPromptTemplate.from_messages([
    ("system","Your are assistant, and you need provide any relavan
    ("user","{input}")
]
)
#Creating a LLM chain and provide output in Json format
chain = prompt | model_with_functions | JsonOutputFunctionsParser()
chain.invoke({"input":"What type of musing singing Bon Jovi?"})
```

Out[20]:
```
{'artist_name': 'Bon Jovi', 'n': 1}
```

In [21]:
```
# We can also use extraction function, that will extract all relant
# Here better to use JsonOutputKeyFunctionParser that will look onl
prompt = ChatPromptTemplate.from_messages([
    ("system", "Extract the relevant information, if not explicitly
    ("human", "{input}")
])
#Creating a function that will work as a person information extract
class Person(BaseModel):
    """Information about a person."""
    name: str = Field(description="person's name")
    age: Optional[int] = Field(description="person's age")
    gender: str = Field(description="person's gender")

class Information(BaseModel):
    """Information to extract."""
    people: List[Person] = Field(description="List of info about pe

#Creating necessary function by pydantic
extraction_functions = [convert_pydantic_to_openai_function(Informa
#Applying function to model
extraction_model = model.bind(functions=extraction_functions, funct
#Building a Chain
extraction_chain = prompt | extraction_model | JsonOutputFunctionsP
#Calling chain
extraction_chain.invoke({"input":"Joe is 30, his mom is Martha"})
```

Out[21]:
```
{'people': [{'name': 'Joe', 'age': 30, 'gender': 'unknown'},
  {'name': 'Martha', 'age': nan, 'gender': 'unknown'}]}
```

Put all together:

- Now I will all these techniques together in a real-world example
- In this example, I will extract information from a blog post and provide a necessary format such as JSON or dictionary

In [22]:

```
#First loading data
loader = WebBaseLoader("https://lilianweng.github.io/posts/2023-06-
documents = loader.load()#download data from website
#fetching only  page content information
doc = documents[0]
page_content = doc.page_content[:1000]
print(page_content[:1000])
```

```




LLM Powered Autonomous Agents | Lil'Log
```

In [23]:

```
#Creating function that will provide a summary of the content,langu
class Overview(BaseModel):
    """Overview of a section of text"""
    summary: str= Field(description="Provide a concise summary of t
    language: str = Field(description="Provide the language that th
    kewords: str = Field(description="Provide keywords related to t

overview_function = [
    convert_pydantic_to_openai_function(Overview)
]
over_model = model.bind(
    functions = overview_function,
    function_call = {"name":"Overview"}

)
#Build prompt
prompt = ChatPromptTemplate.from_messages([
    ("system","Extract the relenvant information, if not explicity
    ("human","{input}")
])
extract_chain = prompt | over_model | JsonOutputFunctionsParser()

#Calling our extractiong chain
extract_chain.invoke({"input":page_content})
```

Out[23]:

```
{'summary': 'LLM Powered Autonomous Agents is a concept of buildin
g agents with LLM (large language model) as its core controller.',
 'language': 'English',
 'keywords': 'LLM, autonomous agents, planning, memory, tool use,
challenges'}
```

In [24]:

```
#Also you can split your documents into chunck and exctrac informat
#You can add to your chain some data preparation methods that will
#And you can combine two different chains and get final result
# Here I propose to create a new function that will fetch relevant
class Paper(BaseModel):
    """Information about papers mentioned."""
    title: str
    author: Optional[str]


class Info(BaseModel):
    """Information to extract"""
    papers: List[Paper]


paper_extraction_function = [
    convert_pydantic_to_openai_function(Info)
]
extraction_model = model.bind(
    functions=paper_extraction_function,
    function_call={"name":"Info"}
)


#Creating prompt template
template = """A article will be passed to you. Extract from it all

Do not extract the name of the article itself. If no papers are men

Do not make up or guess ANY extra information. Only extract what ex

prompt = ChatPromptTemplate.from_messages([
    ("system", template),
    ("human", "{input}")
])
#First we will create Exctraction chain
extraction_chain = prompt | extraction_model | JsonKeyOutputFunctio

#Splitting text into a chunk
text_splitter = RecursiveCharacterTextSplitter(chunk_overlap =0)
#Now we need to combine our splitting text(chunks)
def flatten(matrix):
    flat_list = []
    for row in matrix:
        flat_list += row
    return flat_list
#Will prepera our text into right format (e.g. "Input":"text")
prep = RunnableLambda(
    lambda x: [{"input": doc} for doc in text_splitter.split_text(x
)
#We call map in exctract_chain beause we will use several operation
chain = prep | extraction_chain.map() | flatten
```

In [25]:

```
chain.invoke(doc.page_content)
```

Out[25]:

```
[{'title': 'Chain of thought (CoT)', 'author': 'Wei et al. 2022'},
 {'title': 'Tree of Thoughts', 'author': 'Yao et al. 2023'},
 {'title': 'LLM+P', 'author': 'Liu et al. 2023'},
 {'title': 'ReAct', 'author': 'Yao et al. 2023'},
 {'title': 'Reflexion', 'author': 'Shinn & Labash 2023'},
 {'title': 'Reflexion: A Framework for Self-Reflection in Reinforc
ement Learning',
  'author': 'Shinn & Labash'},
 {'title': 'Chain of Hindsight: Self-Reflection for Improving Mode
l Outputs',
  'author': 'Liu et al.'},
 {'title': 'Algorithm Distillation: Learning the Process of Reinfo
rcement Learning',
  'author': 'Laskin et al.'},
 {'title': 'Algorithm Distillation', 'author': 'Laskin et al. 2023
'},
 {'title': 'ED (expert distillation)', 'author': ''},
 {'title': 'RL^2 (Duan et al. 2017)', 'author': ''},
 {'title': 'A3C', 'author': ''},
 {'title': 'DQN', 'author': ''},
 {'title': 'LSH (Locality-Sensitive Hashing)', 'author': ''},
 {'title': 'ANNOY (Approximate Nearest Neighbors Oh Yeah)', 'autho
r': ''},
 {'title': 'HNSW (Hierarchical Navigable Small World)', 'author':
''},
 {'title': 'FAISS (Facebook AI Similarity Search)', 'author': ''},
 {'title': 'ScaNN (Scalable Nearest Neighbors)', 'author': ''},
 {'title': 'MRKL (Karpas et al. 2022)', 'author': 'Karpas et al.'}
,
 {'title': 'TALM (Tool Augmented Language Models; Parisi et al. 20
22)',
  'author': 'Parisi et al.'},
 {'title': 'Toolformer (Schick et al. 2023)', 'author': 'Schick et
al.'},
 {'title': 'HuggingGPT (Shen et al. 2023)', 'author': 'Shen et al.
'},
 {'title': 'API-Bank: A Benchmark for Evaluating Tool-Augmented La
nguage Model in API Call',
  'author': 'Li et al. 2023'},
 {'title': 'ChemCrow: Augmenting Language Models with Expert-Desig
ned Tools for Scientific Discovery',
  'author': 'Bran et al. 2023'},
 {'title': 'Boiko et al. (2023)', 'author': ''},
 {'title': 'Generative Agents Simulation', 'author': 'Park, et al.
2023'},
 {'title': 'Park et al. 2023', 'author': ''},
 {'title': 'A Survey of Super Mario Game Design Techniques',
  'author': 'John Smith'},
 {'title': 'Model-View-Controller (MVC) Architecture in Software D
esign',
  'author': 'Jane Doe'},
 {'title': 'Keyboard Input Handling in Python Games',
  'author': 'Alice Johnson'},
 {'title': 'Paper A', 'author': 'Author A'},
 {'title': 'Paper B', 'author': 'Author B'},
 {'title': 'Chain of thought prompting elicits reasoning in large
language models.',
  'author': 'Wei et al.'},
 {'title': 'Tree of Thoughts: Deliberate Problem Solving with Larg
e Language Models.',
  'author': 'Yao et al.'},
 {'title': 'Chain of Hindsight Aligns Language Models with Feedbac
k',
  'author': 'Liu et al.'},
 {'title': 'LLM+P: Empowering Large Language Models with Optimal P
lanning Proficiency',
  'author': 'Liu et al.'},
 {'title': 'ReAct: Synergizing reasoning and acting in language mo
dels.',
  'author': 'Yao et al.'},
 {'title': 'Reflexion: an autonomous agent with dynamic memory and
self-reflection',
  'author': 'Shinn & Labash'},
 {'title': 'In-context Reinforcement Learning with Algorithm Disti
llation',
  'author': 'Laskin et al.'},
 {'title': 'MRKL Systems A modular, neuro-symbolic architecture th
at combines large language models, external knowledge sources and
discrete reasoning.',
  'author': 'Karpas et al.'},
 {'title': 'API-Bank: A Benchmark for Tool-Augmented LLMs',
  'author': 'Li et al.'},
 {'title': 'HuggingGPT: Solving AI Tasks with ChatGPT and its Frie
nds in HuggingFace',
  'author': 'Shen et al.'},
 {'title': 'ChemCrow: Augmenting large-language models with chemis
try tools.',
  'author': 'Bran et al.'},
 {'title': 'Emergent autonomous scientific research capabilities o
f large language models.',
  'author': 'Boiko et al.'},
 {'title': 'Generative Agents: Interactive Simulacra of Human Beha
vior.',
  'author': 'Joon Sung Park et al.'}]
```

> **Conclusion:** So with the help of previous examples and tools, you can simply fetch a lot of useful information from ur data and save it or pass it to the LLM or your vector database, which can provide users with a good answer or generate a necessary answer or provide reasonable information to them

Part 3. Creating a Testing Chat bot

- In this step I will show you how you can create a simple chatbot with some additional tools inside, that can be used for different operations depending on what kind of query passes to the model from the user.

In [26]:

```
#Define the input schema
class OpenMeteoInput(BaseModel):
    latitude: float = Field(..., description="Latitute of the locat
    longitude: float = Field(..., description="Longtitude of the lo


#Creating a function the will take parameters from the previous fun
@tool(args_schema=OpenMeteoInput)
def get_current_temperature(latitude: float, longitude: float) -> d
    """Fetch current temperature for given coordinates."""
    BASE_URL = "https://api.open-meteo.com/v1/forecast"
    #Parameters for the request
    params = {
        "latitude": latitude,
        "longitude": longitude,
        "hourly":"temperature_2m",
        "forecast_days":1,
    }
    #Make the request
    response = requests.get(BASE_URL,params=params)

    if response.status_code == 200:
       results = response.json()
    else:
        raise Exception(f"Api Requst failed with status code: {resp

    current_utc_time = datetime.datetime.utcnow()#fetching present
    #converting to a necessary format
    time_list = [datetime.datetime.fromisoformat(time_str.replace('
    #creating list of time/temperature
    temperature_list = results['hourly']['temperature_2m']

    #Finding a more nearest temperature for our present time from o
    closest_time_index = min(range(len(time_list)), key = lambda i:
    #scrapping temperature
    current_temperature = temperature_list[closest_time_index]

    return print(f"The current temperature is {current_temperature}
```

In [27]:

```
#Creating a function that will fetch relevant information from wiki
@tool
def search_wikipedia(query: str) -> str:
    """Run Wikipedia search and get page summaries."""#Creating des
    page_titles = wikipedia.search(query)#Scrap relevant informatio
    summaries = []
    for page_title in page_titles[: 3]:
        try:
            wiki_page =  wikipedia.page(title=page_title, auto_sugg
            summaries.append(f"Page: {page_title}\nSummary: {wiki_p
        except (
            self.wiki_client.exceptions.PageError,
            self.wiki_client.exceptions.DisambiguationError,
        ):
            pass
    if not summaries:
        return "No good Wikipedia Search Result was found"
    return "\n\n".join(summaries)
```

In [28]:

```
#You can also create your own function
@tool
def create_your_own(quesry: str) -> str:
    """This function can do whatever you would like once you fill i
    print(type(query))
    return query[::-1]
```

In [29]:

```
#Assigned tools
tools = [get_current_temperature,search_wikipedia,create_your_own]
```

In [30]:

```
#Creating class that will work for us like a brain to our chatbot
pn.extension()

class cbfs(param.Parameterized):

    def __init__(self, tools, **params):
        super(cbfs, self).__init__( **params)
        self.panels = []
        self.functions = [format_tool_to_openai_function(f) for f i
        self.model = ChatOpenAI(temperature=0).bind(functions=self.
        self.memory = ConversationBufferMemory(return_messages=True
        self.prompt = ChatPromptTemplate.from_messages([
            ("system", "You are helpful but sassy assistant"),
            MessagesPlaceholder(variable_name="chat_history"),
            ("user", "{input}"),
            MessagesPlaceholder(variable_name="agent_scratchpad")
        ])
        self.chain = RunnablePassthrough.assign(
            agent_scratchpad = lambda x: format_to_openai_functions
        ) | self.prompt | self.model | OpenAIFunctionsAgentOutputPa
        self.qa = AgentExecutor(agent=self.chain, tools=tools, verb

    def convchain(self, query):
        if not query:
            return
        inp.value = ''
        result = self.qa.invoke({"input": query})
        self.answer = result['output']
        self.panels.extend([
            pn.Row('User:', pn.pane.Markdown(query, width=450)),
            pn.Row('ChatBot:', pn.pane.Markdown(self.answer, width=
        ])
        return pn.WidgetBox(*self.panels, scroll=True)


    def clr_history(self,count=0):
        self.chat_history = []
        return
```

In [31]:

```
#Implementing frontend steps of our ChatBot
cb = cbfs(tools)

inp = pn.widgets.TextInput( placeholder='Enter text here…')#Line of

conversation = pn.bind(cb.convchain, inp)

tab1 = pn.Column(
    pn.Row(inp),
    pn.layout.Divider(),
    pn.panel(conversation,  loading_indicator=True, height=400),
    pn.layout.Divider(),
)

dashboard = pn.Column(
    pn.Row(pn.pane.Markdown('# QnA_Bot')),
    pn.Tabs(('Conversation', tab1))
)#Assigned name of chatbot and column from where you will see all i
dashboard
```

Out[31]:

```
BokehModel(combine_events=True, render_bundle={'docs_json': {'38c6
c82b-a148-415e-93ae-1e1d1104ddc8': {'version…
```

Overal Conclusion:

> In this project I showed you several tools:

- 1. Techniques of extraction and preparing your data to be ready to go to your LLM or another Machine learning algorithm you can work with
- 2. Secondary by creating functions and putting them into your model, you can provide the opportunity to determine which function is better for you depending on what kind of user query(input) will be. This is helpful when you want to build a more complex chatbot that can work not only on simple tasks such as Q/A but also provide summaries, analysis, or some operations on the data that you have.
- 3. In the last part of this project. I created a chatbot with a simple interface by using **pn** library and showed how it can work in your local **Jupiter, vs code etc.**

In [ ]: