# Deep Dive into LangChain

```
cd /content/drive/MyDrive/Colab Notebooks/LangChain Deep Dive
```

```
/content/drive/MyDrive/Colab Notebooks/LangChain Deep Dive
```

```
ls
```

```
LangChain_Deep_Dive.ipynb  requirements.txt
```

```
pip install -r requirements.txt -q
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 12.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.3/302.3 kB 12.8 MB/s eta 0:00:00
```

```
#pip show openai
```

## Python-dotenv

```
import os
from dotenv import load_dotenv, find_dotenv
load_dotenv(find_dotenv(), override=True)

# os.environ.get('OPENAI_API_KEY')
```

```
True
```

# Chat Models: GPT-3.5 Turbo and GPT-4

```
pip install -q langchain_openai
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.8/62.8 kB 2.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 437.6/437.6 kB 11.7 MB/s eta 0:00:00
```

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI()
output = llm.invoke('Explain quatum mechanics in one sentence.', model='gpt-3.5-turbo')
print(output.content)
```

```
Quantum mechanics is the branch of physics that describes the behavior of particles at the smallest scales, where traditional Newtonian
```

```
# help(ChatOpenAI)
```

```
from langchain.schema import SystemMessage, AIMessage, HumanMessage

messages = [
    SystemMessage(content='You are a physicist and respond only in German.'),
    HumanMessage(content='Explain quatum mechanics in one sentence.')
]

output = llm.invoke(messages)
print(output.content)
```

```
Quantenmechanik ist ein physikalisches Theoriegebäude, das das Verhalten von Teilchen auf atomarer und subatomarer Ebene beschreibt.
```

# Caching LLM Responses

## In-Memory Cache

```
pip install langchain_community
```

```
Requirement already satisfied: tenacity!=8.4.0,<10,>=8.1.0 in /usr/local/lib/python3.11/dist-packages (from langchain_community) (9.1.
Collecting dataclasses-json<0.7,>=0.5.7 (from langchain_community)
  Downloading dataclasses_json-0.6.7-py3-none-any.whl.metadata (25 kB)
Collecting pydantic-settings<3.0.0,>=2.4.0 (from langchain_community)
  Downloading pydantic_settings-2.9.1-py3-none-any.whl.metadata (3.8 kB)
Requirement already satisfied: langsmith<0.4,>=0.1.125 in /usr/local/lib/python3.11/dist-packages (from langchain_community) (0.3.39)
Collecting httpx-sse<1.0.0,>=0.4.0 (from langchain_community)
  Downloading httpx_sse-0.4.0-py3-none-any.whl.metadata (9.0 kB)
Requirement already satisfied: numpy>=1.26.2 in /usr/local/lib/python3.11/dist-packages (from langchain_community) (2.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3->langcha
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain_comm
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain_communi
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain_com
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain_c
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain_comm
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain_com
Collecting marshmallow<4.0.0,>=3.18.0 (from dataclasses-json<0.7,>=0.5.7->langchain_community)
  Downloading marshmallow-3.26.1-py3-none-any.whl.metadata (7.3 kB)
Collecting typing-inspect<1,>=0.4.0 (from dataclasses-json<0.7,>=0.5.7->langchain_community)
  Downloading typing_inspect-0.9.0-py3-none-any.whl.metadata (1.5 kB)
Requirement already satisfied: langchain-text-splitters<1.0.0,>=0.3.8 in /usr/local/lib/python3.11/dist-packages (from langchain<1.0.0
Requirement already satisfied: pydantic<3.0.0,>=2.7.4 in /usr/local/lib/python3.11/dist-packages (from langchain<1.0.0,>=0.3.24->langc
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.11/dist-packages (from langchain-core<1.0.0,>=0.3.56->la
Requirement already satisfied: packaging<25,>=23.2 in /usr/local/lib/python3.11/dist-packages (from langchain-core<1.0.0,>=0.3.56->lan
Requirement already satisfied: typing-extensions>=4.7 in /usr/local/lib/python3.11/dist-packages (from langchain-core<1.0.0,>=0.3.56->
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.1.125->langchain_co
Requirement already satisfied: orjson<4.0.0,>=3.9.14 in /usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.1.125->langcha
Requirement already satisfied: requests-toolbelt<2.0.0,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.1.12
Requirement already satisfied: zstandard<0.24.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.1.125->lan
Requirement already satisfied: python-dotenv>=0.21.0 in /usr/local/lib/python3.11/dist-packages (from pydantic-settings<3.0.0,>=2.4.0-
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic-settings<3.0.0,>=2.4
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2->langchain_com
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2->langchain_community) (3.1
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2->langchain_community
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2->langchain_community
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.11/dist-packages (from SQLAlchemy<3,>=1.4->langchain_community) (
Requirement already satisfied: anyio in /usr/local/lib/python3.11/dist-packages (from httpx<1,>=0.23.0->langsmith<0.4,>=0.1.125->langc
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx<1,>=0.23.0->langsmith<0.4,>=0.1.12
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx<1,>=0.23.0->langsmith<0
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.11/dist-packages (from jsonpatch<2.0,>=1.33->langchain-core<
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.7.4->langcha
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.7.4->langchai
Collecting mypy-extensions>=0.3.0 (from typing-inspect<1,>=0.4.0->dataclasses-json<0.7,>=0.5.7->langchain_community)
  Downloading mypy_extensions-1.1.0-py3-none-any.whl.metadata (1.1 kB)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio->httpx<1,>=0.23.0->langsmith<0.4,>=
Downloading langchain_community-0.3.23-py3-none-any.whl (2.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 16.6 MB/s eta 0:00:00
Downloading dataclasses_json-0.6.7-py3-none-any.whl (28 kB)
Downloading httpx_sse-0.4.0-py3-none-any.whl (7.8 kB)
Downloading pydantic_settings-2.9.1-py3-none-any.whl (44 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.4/44.4 kB 2.0 MB/s eta 0:00:00
Downloading marshmallow-3.26.1-py3-none-any.whl (50 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50.9/50.9 kB 1.8 MB/s eta 0:00:00
Downloading typing_inspect-0.9.0-py3-none-any.whl (8.8 kB)
Downloading mypy_extensions-1.1.0-py3-none-any.whl (5.0 kB)
Installing collected packages: mypy-extensions, marshmallow, httpx-sse, typing-inspect, pydantic-settings, dataclasses-json, langchain
Successfully installed dataclasses-json-0.6.7 httpx-sse-0.4.0 langchain_community-0.3.23 marshmallow-3.26.1 mypy-extensions-1.1.0 pyda
```

```
from langchain.globals import set_llm_cache
from langchain_openai import OpenAI
llm = OpenAI(model_name='gpt-3.5-turbo-instruct') # To make the caching really obvious, lets use a slower and older model.
```

```
%%time
from langchain.cache import InMemoryCache
set_llm_cache(InMemoryCache())
prompt = 'Tell me a joke that a toddler can understand.'
llm.invoke(prompt)
```

```
CPU times: user 1.07 s, sys: 56.1 ms, total: 1.13 s
Wall time: 2.19 s
'\nWhy was the broom late for school? Because it overswept!'
```

```
%%time
llm.invoke(prompt)
```

```
CPU times: user 1.12 ms, sys: 1e+03 ns, total: 1.12 ms
Wall time: 5.09 ms
'\nWhy was the broom late for school? Because it overswept!'
```

## SQLite Cache

```
from langchain.cache import SQLiteCache
set_llm_cache(SQLiteCache(database_path='langchaind.db'))

# First request (not in cache, takes longer)
llm.invoke('Tell me a joke')

# Second request (cached, faster)
llm.invoke('Tell me a joke')
```

## LLM Streaming

```
from langchain_openai import ChatOpenAI
llm = ChatOpenAI()
prompt = 'Write a rock song about the Moon and a Raven.'
print(llm.invoke(prompt).content)
```

Show hidden output

```
for chunk in llm.stream(prompt):
  print(chunk.content, end='', flush=True)
```

```
Verse 1:
In the dead of night, the moon appears so bright
Casting shadows on the land below
And in the darkness, a raven takes flight
Singing a song only he knows

Chorus:
Oh moon, shining bright
Guide the raven through the night
Oh raven, take flight
Sing your song under the moonlight

Verse 2:
The raven soars high, against the midnight sky
Whispering secrets to the moon
As the world sleeps, they keep their watchful eye
In this cosmic dance, they're in tune

Chorus:
Oh moon, shining bright
Guide the raven through the night
Oh raven, take flight
Sing your song under the moonlight

Bridge:
They're two creatures of the night
Bound by a mysterious light
A dance of shadows in the dark
Their connection leaves its mark

Chorus:
Oh moon, shining bright
Guide the raven through the night
Oh raven, take flight
Sing your song under the moonlight

Outro:
In the stillness of the night, they roam
The moon and raven, forever entwined
A haunting melody in the silent gloam
Their spirits intertwined.
```

## PromptTemplates

```
from langchain.prompts import PromptTemplate
from langchain_openai import ChatOpenAI

template = '''You are an experience virologist.
Write a few sentences about the following virus "{virus}" in {language}.'''

prompt_template = PromptTemplate(template=template)

prompt = prompt_template.format(virus='hiv', language='german')
prompt
```

```
'You are an experience virologist.\nWrite a few sentences about the following virus "hiv" in german.'
```

```
llm = ChatOpenAI(model_name='gpt-3.5-turbo', temperature=0)
output = llm.invoke(prompt).content
print(output)
```

```
HIV, das humane Immundefizienzvirus, ist ein Virus, das das Immunsystem des Menschen schwächt. Es wird hauptsächlich durch ungeschützten
```

## ChatPromptTemplates

```
from langchain.prompts import ChatPromptTemplate, HumanMessagePromptTemplate
from langchain_core.messages import SystemMessage

chat_template = ChatPromptTemplate(
    [
        SystemMessage(content='You respond only in the JSON format.'),
        HumanMessagePromptTemplate.from_template('Top {n} countries in {area} by population.')
    ]
)

messages = chat_template.format_messages(n='10', area='World')
print(messages)
```

```
[SystemMessage(content='You respond only in the JSON format.', additional_kwargs={}, response_metadata={}), HumanMessage(content='Top 10
```

```
from langchain_openai import ChatOpenAI
llm = ChatOpenAI()
output = llm.invoke(messages).content
print(output)
```

```
{
        "1": {
            "country": "China",
            "population": "1,415,045,928"
        },
        "2": {
            "country": "India",
            "population": "1,354,051,854"
        },
        "3": {
            "country": "United States",
            "population": "326,766,748"
        },
        "4": {
            "country": "Indonesia",
            "population": "266,794,980"
        },
        "5": {
            "country": "Pakistan",
            "population": "200,813,818"
        },
        "6": {
            "country": "Brazil",
            "population": "190,732,694"
        },
        "7": {
            "country": "Nigeria",
            "population": "185,989,640"
        },
        "8": {
            "country": "Bangladesh",
            "population": "157,826,578"
        },
```

```
        "9": {
            "country": "Russia",
            "population": "142,257,519"
        },
        "10": {
            "country": "Mexico",
            "population": "129,163,276"
        }
    }
```

## Simple Chains

```
from langchain_openai import ChatOpenAI
from langchain import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

llm = ChatOpenAI()
template = '''You are an experience virologist.
Write a few sentences about the following virus "{virus}" in {language}.'''
prompt_template = PromptTemplate.from_template(template=template)

chain = prompt_template | llm | StrOutputParser()

output = chain.invoke({'virus': 'HSV', 'language': 'Spanish'})


print(output)
```

```
El virus del herpes simple (HSV) es un virus altamente contagioso que afecta principalmente a la piel y las membranas mucosas. Hay dos t
```

```
template = 'What is the capital of {country}?. List the top 3 places to visit in that city. Use bullet points.'
prompt_template = PromptTemplate.from_template(template=template)

chain = prompt_template | llm | StrOutputParser()

country = input('Enter Country: ')
output = chain.invoke(country)
print(output)
```

```
Enter Country: Brazil
The capital of Brazil is Brasília.

- Juscelino Kubitschek Memorial: This museum pays tribute to Juscelino Kubitschek, the president who established Brasília as the country
- National Congress: This iconic building houses both the Chamber of Deputies and the Federal Senate of Brazil, and is a must-visit for
- Planalto Palace: This is the official workplace of the President of Brazil and is an important landmark in Brasília.
```

## Sequential Chains

```
from langchain_openai import ChatOpenAI
from langchain import PromptTemplate
# from langchain.chains import SimpleSequentialChain
from langchain_core.output_parsers import StrOutputParser

llm1 = ChatOpenAI(model_name='gpt-3.5-turbo', temperature=0.5)
prompt_template1 = PromptTemplate.from_template(
    template='You are an experienced scientist and Python programmer. Write a function that implements the concept of {concept}.'
)
chain1 = prompt_template1 | llm1 | StrOutputParser()

llm2 = ChatOpenAI(model_name='gpt-4-turbo-preview', temperature=1.2)
prompt_template2 = PromptTemplate.from_template(
    template = 'Given the Python function {function}, describe it as detailed as possible.'
)
chain2 = prompt_template2 | llm2 | StrOutputParser()

overall_chain = chain1 | chain2
output = overall_chain.invoke('linear regression')


print(output)
```

The Python function `linear_regression` is designed to perform linear regression, a fundamental method in statistical modeling and mac

### Importing Necessary Libraries

The function begins by importing NumPy, a widely used library in Python for numerical computing. NumPy provides efficient data structu

```python
import numpy as np
```

### Function Definition

The `linear_regression` function is defined with two parameters: `X` and `y`.
- `X`: This is a matrix (or two-dimensional NumPy array) representing the input features. In simple linear regression (as illustrated
- `y`: This is a vector (or one-dimensional NumPy array) representing the target values. Each entry in `y` corresponds to the output v

```python
def linear_regression(X, y):
```

### Adding an Intercept Term

Linear regression models attempt to predict an output as a linear combination of input features. The equation for simple linear regres

```python
X = np.c_[np.ones(X.shape[0]), X]
```

### Computing the Coefficients

The most computationally efficient method to find the regression coefficients that minimize the square differences between observed an
\[ coefficients = (X^T X)^{-1} X^T y \]
This closed-form solution directly computes the coefficients (`m` and `b` in simple regression) as follows:
- `X.T` is the transpose of matrix `X`, flipping its dimensions.
- `.dot()` is a method for matrix multiplication.
- `np.linalg.inv()` calculates the inverse of a matrix.

By applying this formula, the function calculates the coefficients that best fit the linear model to the given data.

```python
coefficients = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
```

### Return Values

The function returns the calculated coefficients. In the context of simple linear regression, the returned value will consist of two e
- The intercept `b`, which is the first element of the coefficients array.
- The slope `m`, which is the second element. This represents the amount of change in the predicted output `y` for a one-unit change i

```python
return coefficients
```

### Example Usage

An example usage scenario demonstrates how to use the function with input features `X` (consisting of a single feature for simplicity)

```

## LangChain Agents in Action: Python REPL

```
pip install -q langchain_experimental
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 209.2/209.2 kB 5.3 MB/s eta 0:00:00
```

```
from langchain_experimental.utilities import PythonREPL
python_repl = PythonREPL()
python_repl.run('print([n for n in range(1, 100) if n % 13 == 0])')
```

```
WARNING:langchain_experimental.utilities.python:Python REPL can execute arbitrary code. Use with caution.
'[13, 26, 39, 52, 65, 78, 91]\n'
```

```
from langchain_experimental.agents.agent_toolkits import create_python_agent
from langchain_experimental.tools.python.tool import PythonREPLTool
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model_name='gpt-4-turbo-preview', temperature=0)
agent_executor = create_python_agent(
```
```

```
    llm=llm,
    tool=PythonREPLTool(),
    verbose=True
)

agent_executor.invoke('Calculate the square root of the factorial of 12 and display it with 4 decimal points')
```

```
> Entering new AgentExecutor chain...
To solve this, I need to calculate the factorial of 12 first, then find its square root, and finally format the result to display it wit
Action: Python_REPL
Action Input: import math
print(f"{math.sqrt(math.factorial(12)):.4f}")
Observation: 21886.1052

Thought:I now know the final answer
Final Answer: 21886.1052

> Finished chain.
{'input': 'Calculate the square root of the factorial of 12 and display it with 4 decimal points',
 'output': '21886.1052'}
```

```
response = agent_executor.invoke('What is the answer to 5.1 ** 7.3?')
```

```
> Entering new AgentExecutor chain...
I need to calculate 5.1 raised to the power of 7.3 to get the answer.
Action: Python_REPL
Action Input: print(5.1 ** 7.3)
Observation: 146306.05007233328

Thought:I now know the final answer
Final Answer: 146306.05007233328

> Finished chain.
```

```
response
```

```
{'input': 'What is the answer to 5.1 ** 7.3?', 'output': '146306.05007233328'}
```

```
print(response['output'])
```

```
146306.05007233328
```

## LangChain Tools: DuckDuckGo and Wikipedia

```
pip install -q duckduckgo-search
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.3/3.3 MB 26.4 MB/s eta 0:00:00
```

```
from langchain.tools import DuckDuckGoSearchRun

search = DuckDuckGoSearchRun()
output = search.invoke('Where was Freddie Mercury born?')
print(output)
```

```
Freddie Mercury (born September 5, 1946, Stone Town, Zanzibar [now in Tanzania]—died November 24, 1991, Kensington, London, England) was
```

```
search.name
```

```
'duckduckgo_search'
```

```
search.description
```

```
'A wrapper around DuckDuckGo Search. Useful for when you need to answer questions about current events. Input should be a search quer
```

```
search
```

```
DuckDuckGoSearchRun(api_wrapper=DuckDuckGoSearchAPIWrapper(region='wt-wt', safesearch='moderate', time='y', max_results=5,
    backend='auto', source='text'))
```

```
from langchain.tools import DuckDuckGoSearchResults

search = DuckDuckGoSearchResults()
output = search.run('Freddie Mercury and Queen')
print(output)
```

```
snippet: Freddie Mercury (born September 5, 1946, Stone Town, Zanzibar [now in Tanzania]—died November 24, 1991, Kensington, London, Eng
```

```
from langchain_community.utilities import DuckDuckGoSearchAPIWrapper

wrapper = DuckDuckGoSearchAPIWrapper(region='de-de', max_results=3, safesearch='moderate')
search = DuckDuckGoSearchResults(api_wrapper=wrapper, source='news')
output = search.run('Berlin')
print(output)
```

```
snippet: Erfahren Sie mehr über die Geschichte, Geographie, Politik und Kultur der bevölkerungsreichsten Stadt Deutschlands und der Eurc
```

```
import re
pattern = r'snippet: (.*?), title: (.*?), link: (.*?)(?:,|$)'
matches = re.findall(pattern, output, re.DOTALL)

for snippet, title, link in matches:
  print(f'Snippet: {snippet}\nTitle: {title}\nLink: {link}\n')
  print('-' * 50)
```

```
Snippet: Erfahren Sie mehr über die Geschichte, Geographie, Politik und Kultur der bevölkerungsreichsten Stadt Deutschlands und der Eurc
Title: Berlin - Wikipedia
Link: https://de.wikipedia.org/wiki/Berlin

--------------------------------------------------
Snippet: Der Stadtführer für Berlin mit aktuellen Informationen und Auskunft zu Jobs, zum Leben, Arbeiten, Ausgehen, Einkaufen und Urlau
Title: Stadtportal Berlin | Stadtinformationen für Berlin bei ... - meinestadt.de
Link: https://home.meinestadt.de/berlin

--------------------------------------------------
Snippet: Einfach zurechtfinden: Mit unserer übersichtlichen Liste der Sehenswürdigkeiten in Berlin planst du deinen Städtetrip ohne Umwe
Title: Berlin Sehenswürdigkeiten: Top-25-Liste mit Tipps & Bezirken
Link: https://www.voucherwonderland.com/reisemagazin/top-20-sehenswuerdigkeiten-in-berlin/

--------------------------------------------------
Snippet: Du fragst dich, was man in Berlin machen kann? Wir zeigen dir die besten Tagesausflüge, Unternehmungen & Aktivitäten für Berlin
Title: Was kann man in Berlin machen? TOP 25 Aktivitäten & Unternehmungen!
Link: https://www.traveloptimizer.de/ausflugsziele-berlin/

--------------------------------------------------
```

```
pip install -q wikipedia
```

```
  Preparing metadata (setup.py) ... done
  Building wheel for wikipedia (setup.py) ... done
```

```
from langchain_community.tools import WikipediaQueryRun
from langchain_community.utilities import WikipediaAPIWrapper

api_wrapper = WikipediaAPIWrapper(top_k_results=1, doc_content_chars_max=5000)
wiki = WikipediaQueryRun(api_wrapper=api_wrapper)
wiki.invoke({'query': 'vector database'})
```

```
'Page: Vector database\nSummary: A vector database, vector store or vector search engine is a database that can store vectors (fixed-le
ngth lists of numbers) along with other data items. Vector databases typically implement one or more Approximate Nearest Neighbor algor
ithms, so that one can search the database with a query vector to retrieve the closest matching database records.\nVectors are mathemat
ical representations of data in a high-dimensional space. In this space, each dimension corresponds to a feature of the data, with the
number of dimensions ranging from a few hundred to tens of thousands, depending on the complexity of the data being represented. A vect
or\'s position in this space represents its characteristics. Words, phrases, or entire documents, as well as images, audio, and other t
ypes of data, can all be vectorized.\nThese feature vectors may be computed from the raw data using machine learning methods such as fe
```

```
wiki.invoke('Google Gemini')
```

```
'Page: Gemini (chatbot)\nSummary: Gemini, formerly known as Bard, is a generative artificial intelligence chatbot developed by Google. Based on the large language model (LLM) of the same name, it was launched in 2023 in response to the rise of OpenAI\'s ChatGPT. It was previously based on the LaMDA and PaLM LLMs.\nLaMDA had been developed and announced in 2021, but it was not released to the public out of an abundance of caution. OpenAI\'s launch of ChatGPT in November 2022 and its subsequent popularity caught Google executives off-guard, prompting a sweeping response in the ensuing months. After mobilizing its workforce, the company launched Bard in a limited capacity in March 2023 before expanding to other countries in May. Bard took center stage during the 2023 Google I/O keynote in May and was upgraded to the Gemini LLM in December. In February 2024, Bard and Duet AI, another artificial intelligence product from Google, were uni
```

## Creating a React Agent

```
pip install langchainhub -q
```

```
  File "<ipython-input-19-82c9e354a355>", line 1
    pip install langchainhub -q
        ^
SyntaxError: invalid syntax
```

```
import os
from dotenv import load_dotenv, find_dotenv
load_dotenv(find_dotenv(), override=True)

# os.environ.get('OPENAI_API_KEY')
```

```
True
```

```
pip install -q duckduckgo-search
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.3/3.3 MB 51.0 MB/s eta 0:00:00
```

```
pip install wikipedia -q
```

```
  Preparing metadata (setup.py) ... done
  Building wheel for wikipedia (setup.py) ... done
```

```
from langchain.prompts import PromptTemplate
from langchain import hub
from langchain.agents import Tool, AgentExecutor, initialize_agent, create_react_agent
from langchain.tools import DuckDuckGoSearchRun, WikipediaQueryRun
from langchain.utilities import WikipediaAPIWrapper
from langchain_experimental.tools.python.tool import PythonREPLTool
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model_name='gpt-4-turbo-preview', temperature=0) # gpt-3.5-turbo for free access

# Template
template = '''
Answer the following questions in GERMAN as best you can.
Questions: {q}
'''

prompt_template = PromptTemplate.from_template(template)
prompt = hub.pull('hwchase17/react')
# print(type(prompt))
# print(prompt.input_variables)
# print(prompt.template)

# 1. Python REPL Tool (for executing Python code)
python_repl = PythonREPLTool()
python_repl_tool = Tool(
    name='Python REPL',
    func=python_repl.run,
    description='Useful when you need to use Python to answer a question. You should input Python code.'
)

# 2. Wikipedia Tool (for searching Wikipedia)
api_wrapper = WikipediaAPIWrapper()
wikipedia = WikipediaQueryRun(api_wrapper=api_wrapper)
wikipedia_tool = Tool(
    name='Wikipedia',
    func=wikipedia.run,
    description='Useful for when you need to look up a topic, country, or person on Wikipedia.'
```

```
)

# 3. DuckDuckGo Search Tool (for general web searches)
search = DuckDuckGoSearchRun()
duckduckgo_tool = Tool(
    name='DuckDuckGo Search',
    func=search.run,
    description='Useful when you need to perform an internet search to find information that another tool can\'t provide.'
)

# Tools
tools = [python_repl_tool, wikipedia_tool, duckduckgo_tool]

# Agent
agent = create_react_agent(llm, tools, prompt)
agent_executor = AgentExecutor(
    agent=agent,
    tools=tools,
    verbose=True,
    handle_parsing_errors=True,
    max_iterations=10
)
```

```
/usr/local/lib/python3.11/dist-packages/langsmith/client.py:272: LangSmithMissingAPIKeyWarning: API key must be provided when using host
  warnings.warn(
```

```
# question = 'Generate the first 20 numbers in the Fibonacci series.'
# question = 'Who is the current prime minister of the UK?'
question = 'Tell me about Napoleon Bonaparte early life'

output = agent_executor.invoke({
    'input': prompt_template.format(q=question)
})
```

```
> Entering new AgentExecutor chain...
To answer this question in German, I will first find information about Napoleon Bonaparte's early life in English and then translate it

Action: Wikipedia
Action Input: Napoleon BonaparteNapoleon
Summary: Napoleon Bonaparte (born Napoleone Buonaparte; 15 August 1769 – 5 May 1821), later known by his regnal name Napoleon I, was a F
Born on the island of Corsica to a family of Italian origin, Napoleon moved to mainland France in 1779 and was commissioned as an office
The breakdown of the Treaty of Amiens led to the War of the Third Coalition by 1805. Napoleon shattered the coalition with a decisive vi
Napoleon is considered one of the greatest military commanders in history, and Napoleonic tactics are still studied at military schools

Final Answer: Napoleon Bonaparte wurde am 15. August 1769 auf der Insel Korsika in eine Familie italienischer Herkunft geboren. Im Jahr

> Finished chain.
```

```
for i in output.keys():
  print(i, output[i])
```

```
input
Answer the following questions in GERMAN as best you can.
Questions: Tell me about Napoleon Bonaparte early life

output Napoleon Bonaparte wurde am 15. August 1769 auf der Insel Korsika in eine Familie italienischer Herkunft geboren. Im Jahr 1779 zo
```

## Adding a new tool

```
pip install arxiv -q
```

```
  Preparing metadata (setup.py) ... done
                                        81.3/81.3 kB 3.8 MB/s eta 0:00:00
  Building wheel for sgmllib3k (setup.py) ... done
```

```
from langchain.prompts import PromptTemplate
from langchain import hub
from langchain.agents import Tool, AgentExecutor, initialize_agent, create_react_agent
from langchain.tools import DuckDuckGoSearchRun, WikipediaQueryRun
from langchain.utilities import WikipediaAPIWrapper, ArxivAPIWrapper
```

```
from langchain_experimental.tools.python.tool import PythonREPLTool
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model_name='gpt-4-turbo-preview', temperature=0) # gpt-3.5-turbo for free access

# Template
template = '''
Answer the following questions in GERMAN as best you can.
Questions: {q}
'''

prompt_template = PromptTemplate.from_template(template)
prompt = hub.pull('hwchase17/react')
# print(type(prompt))
# print(prompt.input_variables)
# print(prompt.template)

# 1. Python REPL Tool (for executing Python code)
python_repl = PythonREPLTool()
python_repl_tool = Tool(
    name='Python REPL',
    func=python_repl.run,
    description='Useful when you need to use Python to answer a question. You should input Python code.'
)

# 2. Wikipedia Tool (for searching Wikipedia)
api_wrapper = WikipediaAPIWrapper()
wikipedia = WikipediaQueryRun(api_wrapper=api_wrapper)
wikipedia_tool = Tool(
    name='Wikipedia',
    func=wikipedia.run,
    description='Useful for when you need to look up a topic, country, or person on Wikipedia.'
)

# 3. DuckDuckGo Search Tool (for general web searches)
search = DuckDuckGoSearchRun()
duckduckgo_tool = Tool(
    name='DuckDuckGo Search',
    func=search.run,
    description='Useful when you need to perform an internet search to find information that another tool can\'t provide.'
)

# 4. Arxiv Search Tool (for scientific papers)
arxiv = ArxivAPIWrapper()
arxiv_tool = Tool(
    name='Arxiv Search',
    func = arxiv.run,
    description='Useful for finding scientific papers and abstracts from arXiv. Input should be a research topic.'
)

# Tools
tools = [python_repl_tool, wikipedia_tool, duckduckgo_tool, arxiv_tool]

# Agent
agent = create_react_agent(llm, tools, prompt)
agent_executor = AgentExecutor(
    agent=agent,
    tools=tools,
    verbose=True,
    handle_parsing_errors=True,
    max_iterations=10
)
```

```
/usr/local/lib/python3.11/dist-packages/langsmith/client.py:272: LangSmithMissingAPIKeyWarning: API key must be provided when using host
  warnings.warn(
```

```
# question = 'Generate the first 20 numbers in the Fibonacci series.'
# question = 'Who is the current prime minister of the UK?'
# question = 'Tell me about Napoleon Bonaparte early life'
question = 'Find the most recient papers about Deep Learning. The output has to be a list'

output = agent_executor.invoke({
    'input': prompt_template.format(q=question)
})
```

```
> Entering new AgentExecutor chain...
To find the most recent papers about Deep Learning, I should use the Arxiv Search tool since it specializes in finding scientific pape

Action: Arxiv Search
Action Input: Deep LearningPublished: 2018-05-22
Title: Opening the black box of deep learning
Authors: Dian Lei, Xiaoxiao Chen, Jianfei Zhao
Summary: The great success of deep learning shows that its technology contains
profound truth, and understanding its internal mechanism not only has important
implications for the development of its technology and effective application in
various fields, but also provides meaningful insights into the understanding of
human brain mechanism. At present, most of the theoretical research on deep
learning is based on mathematics. This dissertation proposes that the neural
network of deep learning is a physical system, examines deep learning from
three different perspectives: microscopic, macroscopic, and physical world
views, answers multiple theoretical puzzles in deep learning by using physics
principles. For example, from the perspective of quantum mechanics and
statistical physics, this dissertation presents the calculation methods for
convolution calculation, pooling, normalization, and Restricted Boltzmann
Machine, as well as the selection of cost functions, explains why deep learning
must be deep, what characteristics are learned in deep learning, why
Convolutional Neural Networks do not have to be trained layer by layer, and the
limitations of deep learning, etc., and proposes the theoretical direction and
basis for the further development of deep learning now and in the future. The
brilliance of physics flashes in deep learning, we try to establish the deep
learning technology based on the scientific theory of physics.

Published: 2018-06-05
Title: Concept-Oriented Deep Learning
Authors: Daniel T Chang
Summary: Concepts are the foundation of human deep learning, understanding, and
knowledge integration and transfer. We propose concept-oriented deep learning
(CODL) which extends (machine) deep learning with concept representations and
conceptual understanding capability. CODL addresses some of the major
limitations of deep learning: interpretability, transferability, contextual
adaptation, and requirement for lots of labeled training data. We discuss the
major aspects of CODL including concept graph, concept representations, concept
exemplars, and concept representation learning systems supporting incremental
and continual learning.

Published: 2019-07-30
Title: Deep learning research landscape & roadmap in a nutshell: past, present and future -- Towards deep cortical learning
Authors: Aras R. Dargazany
Summary: The past, present and future of deep learning is presented in this work.
Given this landscape & roadmap, we predict that deep cortical learning will be
the convergence of deep learning & cortical learning which builds an artificial
cortical column ultimately.I now know the final answer
Final Answer: Die neuesten Papiere über Deep Learning sind:

1. Titel: Opening the black box of deep learning
   Autoren: Dian Lei, Xiaoxiao Chen, Jianfei Zhao
   Veröffentlicht: 22. Mai 2018
   Zusammenfassung: Diese Dissertation zeigt, dass die Technologie des Deep Learning tiefe Wahrheiten enthält. Die Untersuchung seines

2. Titel: Concept-Oriented Deep Learning
```