Bagaimana LangChain Dapat Membantu Anda Membuat Chatbot, Peringkasan, dan Lainnya yang Didukung LLM 🦜🔗

Perkiraan waktu yang dibutuhkan: 120 menit

LangChain adalah perpustakaan Python yang membantu Anda dengan pemrosesan bahasa alami. Ini memberi Anda akses ke banyak model bahasa canggih dari berbagai penyedia seperti OpenAI, Cohere, Huggingface Hub, IBM WatsonX, dan model LLM lainnya.

Tapi bukan itu saja! LangChain bukan hanya tentang integrasi, ini tentang keserbagunaan. Dilengkapi dengan pemuat dokumen yang dapat menangani beragam format - mulai dari teks dan HTML hingga PDF dan PowerPoint, bahkan email. Ini berarti Anda dapat mengekstrak dan memproses informasi dari hampir semua dokumen, menjadikannya pisau ekstraksi data Swiss Army.

Dan masih ada lagi! LangChain juga merupakan kunci Anda untuk menciptakan antarmuka percakapan yang menarik. Dengan alat inovatif untuk menyusun rantai dan agen percakapan, Anda dapat merancang dialog interaktif dengan model bahasa. Gabungkan rantai percakapan ini dengan memori dan templat cepat, dan voila! Ada percakapan yang sangat mirip dengan manusia sehingga mengaburkan batas antara manusia dan mesin.

Jadi, baik Anda seorang pengembang yang ingin membuat AI percakapan, atau ilmuwan data yang perlu mengekstrak informasi dari berbagai format dokumen, LangChain adalah pustaka Python yang menghidupkan tugas pemrosesan bahasa Anda.

Learning Objectives

LangChain adalah pustaka Python serbaguna yang menyederhanakan pembuatan aplikasi dengan Model Bahasa Besar (LLM). Ini menawarkan:

1. **Antarmuka Universal:** Integrasi mulus dengan berbagai model LLM.
2. **Manajemen Prompt:** Alat untuk menangani, mengoptimalkan, dan membuat serial perintah.
3. **Rantai Percakapan:** Memungkinkan terciptanya dialog kompleks dengan LLM.
4. **Antarmuka Memori:** Memfasilitasi penyimpanan dan pengambilan informasi model.
5. **Indeks:** Fungsi utilitas untuk memuat data teks khusus.
6. **Agen dan Alat:** Memungkinkan penyiapan agen yang dapat menggunakan alat seperti Google Penelusuran, Wikipedia, atau kalkulator.

Menyiapkan lingkungan dan menginstal perpustakaan

Dalam proyek ini, untuk membangun aplikasi AI, kita akan memanfaatkan antarmuka Gradio yang disediakan dengan memeluk wajah.
Mari siapkan lingkungan dan dependensi untuk proyek ini. Buka terminal baru dan pastikan Anda berada di direktori home/proyek. Buka terminal:

Kredit gambar LangChain: unsplash.com
Buat lingkungan virtual python dan instal Gradio menggunakan perintah berikut di terminal:

```
1
2
3
```

```
pip install virtualenv
virtualenv my_env # membuat sebuah  environment virtual my_env
source my_env/bin/activate # mengaktifkan my_env
```

Copied!

Kemudian, instal library yang diperlukan di environment:

```
1
2
```

```
# menginstal library yang diperlukan ke my_env
pip install langchain langchain_openai gradio chromadb tiktoken huggingface_hub wget pysqlite3-binary numexpr
```

Copied!

Sekarang, environment kita siap untuk membuat file python.

Mulai Menggunakan Gradio

Pengenalan gradio

Untuk mendemonstrasikan aplikasi langchain kita perlu memiliki antarmuka web dan gradio menyediakannya…

Membuat demo sederhana

Melalui proyek ini, kita akan membuat aplikasi LLM berbeda dengan antarmuka Gradio. Mari mengenal Gradio dengan membuat aplikasi sederhana:

Masih di direktori `project` , buat file Python dan beri nama `hello.py`.

Buka `hello.py`., tempel kode Python berikut dan simpan file.

```
1
2
3
4
5
6
7
8
```

```
import gradio as gr
```

```
2.
3. def greet(nama):
4.     return "Halo " + name + ", Selamat bergabung di IBM Academy!"
5.
6. demo = gr.Interface(fn=greet, inputs="text", outputs="text")
7.
8. demo.launch(server_name="0.0.0.0", server_port= 7860, share=True)
```

Copied!

Kode di atas membuat **gradio.Interface** bernama demo. Itu membungkus fungsi greet dengan antarmuka pengguna teks-ke-teks sederhana yang dapat Anda gunakan untuk berinteraksi.

Kelas gradio.Interface diinisialisasi dengan 3 parameter yang diperlukan:

- fn: fungsi untuk membungkus UI
- inputs: komponen mana yang akan digunakan sebagai input (misalnya "teks", "gambar", atau "audio")
- keluaran: komponen mana yang akan digunakan untuk keluaran (misalnya "teks", "gambar", atau "label")
  Baris terakhir demo.launch meluncurkan server untuk menyajikan demo kita.

Menjalankan aplikasi demo

Sekarang kembali ke terminal dan pastikan nama lingkungan virtual my_env ditampilkan di awal baris. Sekarang jalankan perintah berikut untuk menjalankan skrip Python.

```
1. 1
```

```
1. python hello.py
```

Copied!

Jalankan kode dengan CTRL+Click atau CMD+Click pada link berikut.

Setelah selesai bermain dengan aplikasi dan ingin keluar, tekan CTRL+C di terminal dan tutup tab aplikasi.

Anda baru saja mencoba antarmuka Gradio untuk pertama kalinya, mudah bukan? Jika Anda ingin mempelajari lebih lanjut tentang penyesuaian di Gradio, Anda diundang untuk mengikuti proyek terpandu yang disebut Menghidupkan model Machine Learning Anda dengan Gradio. Anda dapat menemukannya di Kursus & Proyek di cognitiveclass.ai!

Untuk proyek ini, kita akan menggunakan Gradio sebagai antarmuka untuk aplikasi LLM.

1. Model: Interface Umum untuk Berbagai LLM

LangChain adalah paket yang menyediakan antarmuka umum ke banyak model dasar, termasuk Model Bahasa Besar (LLM). Ini menyederhanakan manajemen dan pengoptimalan yang cepat, dan bertindak sebagai antarmuka pusat ke komponen lainnya. Antarmuka umum ini dirancang untuk bekerja dengan berbagai penyedia LLM seperti OpenAI, Cohere, Hugging Face, Anthropic, dan AI21. LangChain bukanlah penyedia LLM, melainkan menyediakan antarmuka standar yang melaluinya Anda dapat berinteraksi dengan berbagai LLM.

Mari membuat chatbot sederhana menggunakan langchain.

Membuat chatbot sederhana

Skrip berikut disiapkan untuk mengirim permintaan ke OpenAI API menggunakan perpustakaan LangChain. Permintaan tersebut adalah untuk menghasilkan respons terhadap pertanyaan "beri tahu saya fakta menarik tentang kentang". Respons dari API kemudian dicetak ke konsol. Anda dapat menentukan model OpenAI LLM mana yang ingin Anda pilih (mis. model_name="gpt-3.5-turbo").

Untuk mendapatkan kunci API OpenAI untuk ChatGPT, Anda dapat mengikuti langkah-langkah berikut:

1. Kunjungi situs web resmi OpenAI di browser pilihan Anda, klik di sini untuk mengunjungi situs web.
2. Klik "Masuk" dan tambahkan email Anda untuk membuat akun.
3. Setelah masuk, klik ikon "Lihat Kunci API" yang terletak di sudut kanan atas layar.
4. Klik "Buat Kunci API" untuk membuat Kunci API ChatGPT Anda.

Mulai Juni 2023, pengguna uji coba gratis baru menerima kredit senilai $5 (USD) yang masa berlakunya habis setelah tiga bulan.

Masih di direktori project, buat file Python dan beri nama simple_llm.py.

Buka simple_llm.py, tempel kode Python berikut dan simpan file.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
```

```
1. import os
2. from langchain_openai import ChatOpenAI
3.
4. # Mengatur environment variable "OPENAI_API_KEY" dengan OpenAI API key milikmu. ini diperlukan untuk proses autentikasi ke OpenAI API.
5. os.environ["OPENAI_API_KEY"] = "API KEY KAMU"
6.
7. # Mendefinisikan jenis model
8. gpt3 = ChatOpenAI(model_name="gpt-3.5-turbo" )
9.
```

```
text = "Berikan fakta menarik tentang kentang!"
print(gpt3.invoke(text).content)
```

Copied!

Sekarang, kembali ke terminal dan pastikan nama lingkungan virtual `my_env` ditampilkan di awal baris, dan jalankan perintah berikut untuk menjalankan skrip Python.

```
1
```

```
python simple_llm.py
```

Copied!

Output model akan dicetak di terminal.

Untuk membuat Aplikasi untuk chatbot sederhana ini, mari gunakan Gradio.

Ganti kode berikut di `simple_llm.py`.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

```
import os
from langchain_openai import ChatOpenAI
import gradio as gr

# Memasukkan API key
os.environ["OPENAI_API_KEY"] = "API KEY KAMU"

gpt3 = ChatOpenAI(model_name="gpt-3.5-turbo" )

def chatbot(prompt):
    return gpt3.invoke(prompt).content

demo = gr.Interface(fn=chatbot, inputs="text", outputs="text")

demo.launch(server_name="0.0.0.0", server_port= 7860, share=True)
```

Copied!

Sekarang, jalankan code nya:

```
1
```

```
python simple_llm.py
```

Copied!

Buka aplikasi dengan `CTRL+Click` atau `CMD+Click` pada link public URL.

Anda akan melihat yang berikut ini, di sini kita memasukkan prompt ‘berikan lelucon tentang kentang’:

Jika Anda selesai mencoba aplikasi dan ingin keluar, tekan `CTRL+C` di terminal dan tutup tab aplikasi.

Anda baru saja menggunakan antarmuka Gradio untuk pertama kalinya, mudah bukan? Jika Anda ingin mempelajari lebih lanjut tentang penyesuaian di Gradio, Anda diundang untuk mengikuti proyek terpandu yang disebut Menghidupkan model Machine Learning Anda dengan Gradio. Anda dapat menemukannya di Kursus & Proyek di cognitiveclass.ai!

Anda juga dapat menggunakan HuggingFaceHub dan Cohere untuk membangun model LLM

```
1
2
3
4
5
6
7
8
9
```

```
# Menggunakan Hugging FaceHub
import os
from langchain_community.llms import HuggingFaceEndpoint

os.environ["HUGGINGFACEHUB_API_TOKEN"] = "AKSES TOKEN KAMU"
llm = HuggingFaceEndpoint(repo_id="google/flan-ul2")
text = "Beritahu fakta menarik tentang kentang!"
result = llm.invoke(text)
print(result)
```

Copied!

2. Bekerja Dengan Prompt

Dalam pemrograman, Sekarang kita menggunakan “prompt” sebagai masukan untuk model. Perintah ini tidak tetap; mereka dibuat dari bagian yang berbeda. LangChain membantu kita membuat petunjuk ini dengan mudah. Ini juga membantu mengelola dan mengoptimalkannya.

Salah satu fitur utama LangChain adalah kemampuan untuk membuat templat cepat. Templat ini mengambil masukan pengguna dan mengubahnya menjadi perintah terakhir untuk model. Pendekatan ini memungkinkan interaksi yang lebih dinamis dan fleksibel dengan model, sehingga meningkatkan kegunaan dan efektivitasnya.

Membuat Bot Surat Pengantar

Dengan menggunakan templat cepat LangChain, Anda memiliki kesempatan untuk mengembangkan aplikasi yang menerima masukan spesifik seperti posisi, perusahaan, dan keterampilan, yang pada akhirnya menghasilkan surat lamaran yang disesuaikan berdasarkan parameter ini.

Untuk memulai, kita perlu membuat file Python baru. Sebut saja: `prompt_with_template.py`.

Selanjutnya, mari fokus mengembangkan pembuat surat lamaran kita. Kita akan merancang antarmuka berbasis web dengan Gradio, yang akan menghasilkan surat lamaran yang disesuaikan berdasarkan masukan pengguna.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
```

```
import gradio as gr
from langchain.prompts import PromptTemplate
import os
from langchain_openai import ChatOpenAI

openai_api_key = "API KEY KAMU"
os.environ["OPENAI_API_KEY"] = openai_api_key

# Mendefinisikan model AI
llm = ChatOpenAI(
    model_name="gpt-3.5-turbo",
    openai_api_key= openai_api_key
)

# Mendefinisikan PromptTemplate sebagai format prompt untuk input dari user
prompt = PromptTemplate(
    input_variables=["posisi", "perusahaan", "keterampilan"],
    template="Yang terhormat HRD Manajer {perusahaan},\n\nDengan surat ini, saya [NAMA KAMU], ingin melamar untuk posisi {posisi} di {perusahaan
)

# Define a function to generate a cover letter using the llm and user input
def generate_cover_letter(posisi: str, perusahaan: str, keterampilan: str) -> str:
    formatted_prompt = prompt.format(posisi=posisi, perusahaan=perusahaan, keterampilan=keterampilan)
    response = llm.invoke(formatted_prompt).content
    return response

# Define the Gradio interface inputs
inputs = [
    gr.Textbox(label="Posisi"),
    gr.Textbox(label="Perusahaan"),
    gr.Textbox(label="Keterampilan")
]

# Define the Gradio interface output
output = gr.Textbox(label="Template Surat")

# Launch the Gradio interface
gr.Interface(fn=generate_cover_letter, inputs=inputs, outputs=output).launch(server_name="0.0.0.0", server_port= 7860, share=True)
```

Copied!

Berikut penjelasan singkat tentang fungsi setiap bagian:

1. Siapkan kunci API OpenAI, Inisialisasi model OpenAI: Skrip menginisialisasi model OpenAI menggunakan kunci API dan nama model.

2. Tentukan PromptTemplate: Skrip mendefinisikan PromptTemplate yang memformat surat lamaran berdasarkan masukan pengguna untuk posisi, perusahaan, dan keterampilan.

3. Tentukan fungsi untuk menghasilkan surat lamaran: Skrip mendefinisikan fungsi yang menerima masukan pengguna, memformat perintah menggunakan PromptTemplate, mengirimkan perintah ke model OpenAI, dan mengembalikan respons model.

4. Tentukan input dan output antarmuka Gradio: Skrip mendefinisikan input dan output untuk antarmuka Gradio. Inputnya berupa kotak teks untuk posisi, perusahaan, dan keahlian, dan outputnya berupa kotak teks untuk surat lamaran yang dihasilkan.

5. Luncurkan antarmuka Gradio: Terakhir, skrip meluncurkan antarmuka Gradio. Antarmuka akan memanggil fungsi generate_cover_letter ketika pengguna mengirimkan masukannya, dan menampilkan surat lamaran yang dihasilkan di kotak teks keluaran. Antarmuka diluncurkan pada mesin lokal (0.0.0.0) di port 7860.

Sekarang, jalankan kodenya:

```
1
```

```
python prompt_with_template.py
```

Copied!

Buka aplikasi dengan CTRL+Click atau CMD+Click pada link public URL.

Untuk menghentikan aplikasi tekan CTRL+C.

Oleh karena itu, templat prompt mengacu pada cara yang dapat direproduksi untuk menghasilkan prompt. Ini berisi teks awal (template), yang dapat mengambil serangkaian parameter dari pengguna akhir dan menghasilkan prompt.

Latihan: Buat Template yang Menyediakan “cara langkah demi langkah”.

Kode Anda harus menyediakan langkah demi langkah dari prompt input, mis. di sini kita masuk ke ‘Cara melakukan backflip’:

▼ Lihat kunci jawaban

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

```
from langchain.prompts import PromptTemplate
from langchain_openai import ChatOpenAI
import gradio as gr
# initialize the models
openai = ChatOpenAI(
    model_name="gpt-3.5-turbo",
    openai_api_key="API KEY KAMU"
)

def chatbot(user_input):
    # defining a template
    template = """Question: {question}
    please provide step by step Answer:
    """
    prompt = PromptTemplate(template=template, input_variables=["question"])
    formated_prompt =prompt.format(question=str(user_input))
    return openai.invoke(formated_prompt).content

demo = gr.Interface(fn=chatbot, inputs="text", outputs="text")

demo.launch(share=True)
```

Copied!

Sekarang, jalankan kodenya:

```
1
```

```
python prompt_with_template.py
```

Copied!

Buka aplikasi dengan CTRL+Click atau CMD+Click pada link public URL.

Untuk menghentikan aplikasi tekan CTRL+C.

3. Rantai: Urutan Panggilan LLM

Di LangChain, “Rantai” mengacu pada rangkaian panggilan Model Bahasa Besar (LLM). Mereka memungkinkan Anda untuk melampaui satu panggilan LLM dan melibatkan serangkaian panggilan, yang bisa ke LLM atau utilitas lain. Hal ini memungkinkan alur kerja dan aplikasi yang lebih kompleks.

Rantai sangat berguna ketika Anda ingin mengurutkan beberapa panggilan LLM atau utilitas lain untuk membuat aplikasi yang lebih kompleks. Mereka memungkinkan Anda menggabungkan primitif dan LLM untuk memproses input pengguna, menghasilkan perintah, dan banyak lagi.

Chatbot Support Chatbot

chain_customerSupport.py

Aplikasi ini adalah rantai sederhana menggunakan LangChain yang mengambil input pengguna, memformat prompt dengannya, dan kemudian mengirimkannya ke LLM. Ini memiliki antarmuka web di mana Anda dapat memasukkan keluhan pelanggan dan melihat masalah yang teridentifikasi, dan memberikan solusi.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

```
import gradio as gr
from langchain.chains import LLMChain
from langchain_openai import ChatOpenAI
from langchain.prompts import PromptTemplate
import os

# Mengatur LLM
os.environ["OPENAI_API_KEY"] = "API KEY KAMU"
llm = ChatOpenAI(temperature=0.9)

def handle_complaint(komplain: str) -> str:
    #Buat instance LLM dengan nilai temprature 0,9 (nilai lebih tinggi membuat keluaran lebih acak).


    # Merancang template untuk merespon komplain
    prompt = PromptTemplate(input_variables=["komplain"], template="Saya seorang perwakilan layanan pelanggan. Saya menerima keluhan berikut: {k

    # Membuat model bahasa berantai dengan template yang telah dirancang
    chain = LLMChain(llm=llm, prompt=prompt)
    return chain.run(komplain)

# Membuat antarmuka Gradio untuk fungsi handle_complaint
iface = gr.Interface(fn=handle_complaint, inputs="text", outputs="text")
iface.launch(share=True)
```

Copied!

Sekarang, jalankan kodenya:

```
1
```

```
python chain_customerSupport.py
```

Copied!

Buka aplikasi dengan CTRL+Click atau CMD+Click pada link public URL.

Untuk menghentikan aplikasi tekan CTRL+C.

Anda akan melihat yang berikut ini:

4. Indeks: Fungsi Utilitas untuk Mengakses dan Menyimpan File Anda

LangChain menyediakan modul khusus yang dikenal sebagai modul ‘indeks’, yang dirancang khusus untuk meningkatkan interaksi antara Model Pembelajaran Bahasa (LLM) dan berbagai dokumen. Ini melengkapi LLM dengan kemampuan untuk menavigasi secara efektif melalui berbagai indeks, sehingga meningkatkan kemampuan pengambilannya. Fungsi utama indeks ini terletak pada kemampuannya untuk mengambil dokumen paling relevan sebagai respons terhadap permintaan pengguna, sebuah proses yang difasilitasi melalui antarmuka unik yang disebut “Retriever”.

Meskipun indeks memiliki tujuan tambahan dalam kerangka LangChain, penerapan utamanya berpusat pada pengambilan dokumen, khususnya untuk data tidak terstruktur seperti dokumen teks. Modul ini terutama dirancang untuk fokus pada pengindeksan dan pengambilan data tidak terstruktur tersebut. Misalnya, ia dapat memuat data dan menyediakan antarmuka penyimpanan vektor agar dapat disimpan dan dicari.

Buat file python baru dan beri nama python3 summarizer.py. dan salin yang berikut:

```
1
2
3
```

```
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
36. 36
```

```
1. from langchain.document_loaders import TextLoader
2. from langchain.indexes import VectorstoreIndexCreator
3. import wget
4. import os
5. import gradio as gr
6. import pysqlite3
7. import sys
8.
9. sys.modules["sqlite3"] = sys.modules.pop("pysqlite3")
10.
11. # mengakses ke dokumen
12. url = "https://raw.githubusercontent.com/Ichsan-Takwa/Generative-AI-Labs/main/Pembukaan_UUD_1945"
13. output_path = "pembukaanUUD1945.txt"  # nama file lokal
14.
15. # Mengecek jika file sudah ada
16. if not os.path.exists(output_path):
17.     # mengunduh file menggunakan wget
18.     wget.download(url, out=output_path)
19.
20. loader = TextLoader('pembukaanUUD1945.txt')
21.
22. openai_api_key = "API KEY KAMU"
23. os.environ["OPENAI_API_KEY"] = openai_api_key
24.
25. # mengakses data
26. data = loader.load()
27.
28. # Membuat instance untuk mencari data
29. index = VectorstoreIndexCreator().from_loaders([loader])
30.
31. # Menjalankan gradio
32. def summarize(query):
33.     return index.query(query)
34.
35. iface = gr.Interface(fn=summarize, inputs="text", outputs="text")
36. iface.launch(server_name="0.0.0.0", server_port= 7860, share=True)
```

Copied!

Mari kita lihat kodenya selangkah demi selangkah:

1. Kode memeriksa apakah file yang ditentukan oleh output_path sudah ada menggunakan fungsi `os.path.exists()`. Jika file tidak ada, maka file akan diunduh menggunakan fungsi `wget.download()`. Langkah ini menghindari duplikasi file dari setiap proses.
2. Variabel loader diberi instance kelas TextLoader, yang diinisialisasi dengan jalur ke file teks yang diunduh.
3. Variabel data diberi data teks yang dimuat menggunakan metode load() pada instance loader.
4. Indeks dibuat menggunakan kelas `VectorstoreIndexCreator`, dan metode `from_loaders()` dipanggil, meneruskan instance loader sebagai argumen. Langkah ini membuat representasi vektor dari data teks untuk kueri yang efisien.
5. Antarmuka Gradio dibuat menggunakan kelas gr.Interface, diinisialisasi dengan fungsi ringkasan() sebagai fungsi yang dapat dipanggil, dan menentukan tipe masukan dan keluaran sebagai “teks”.

Sekarang, jalankan kodenya:

```
1. 1
```

```
1. python summarizer.py
```

Copied!

Buka aplikasi dengan `CTRL+Click` atau `CMD+Click` pada link public URL.

Untuk menghentikan aplikasi tekan `CTRL+Click`.

Anda akan melihat yang berikut ini:

5. Memory: An Interface for Memory Implementation

LangChain menyediakan antarmuka standar untuk memori, yang membantu mempertahankan status antara panggilan berantai atau agen. Ia juga menawarkan serangkaian implementasi memori dan contoh rantai atau agen yang menggunakan memori. Memori adalah konsep penting dalam LangChain, karena melibatkan keadaan yang bertahan di antara panggilan rantai/agen.

misalnya Anda dapat menyimpan cerita obrolan, dan mengajukan pertanyaan lanjutan.

Chatbot Sahabat

Mari buat chatbot interaktif dengan memori. mari buat file memory.py.

Untuk melakukannya kita memerlukan langkah-langkah berikut:

- Mengonfigurasi memori untuk percakapan: Ini memungkinkan chatbot mempertahankan konteks dari pertukaran percakapan sebelumnya. Dua jenis memori digunakan: satu untuk menyimpan percakapan terkini dan satu lagi untuk menghasilkan versi ringkasan percakapan.
- Menyiapkan model bahasa: Kode ini menyiapkan instance model GPT-3 yang disediakan oleh OpenAI. Parameter suhu, yang disetel ke 0 di sini, menentukan keacakan keluaran model. Suhu 0 membuat keluaran bersifat deterministik, artinya keluaran akan selalu menghasilkan keluaran yang sama dengan masukan yang sama.
- Membuat pengendali percakapan: Ini dilakukan dengan menggunakan objek ConversationChain dari perpustakaan LangChain. Objek ini mengelola alur percakapan dan menggunakan konfigurasi memori dan model bahasa yang diatur pada langkah sebelumnya.
- Membangun antarmuka web menggunakan Gradio: Antarmuka terdiri dari kotak obrolan untuk percakapan, kotak teks untuk input pengguna, dan tombol untuk menghapus percakapan.

```
 1. 1
 2. 2
 3. 3
 4. 4
 5. 5
 6. 6
 7. 7
 8. 8
 9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
36. 36
37. 37
38. 38
39. 39
40. 40
41. 41
42. 42
43. 43
44. 44
45. 45
46. 46
47. 47
48. 48
49. 49
50. 50
51. 51
52. 52
53. 53
54. 54
55. 55
56. 56
57. 57
58. 58
59. 59
60. 60
61. 61
62. 62
63. 63
64. 64
65. 65
66. 66
67. 67
68. 68
69. 69
```

```
70. 70
71. 71
72. 72
73. 73
74. 74
```

```
 1. # Import library yang diperlukan
 2. from langchain_openai import ChatOpenAI
 3. from langchain.prompts import PromptTemplate
 4. from langchain.chains import ConversationChain
 5. from langchain.memory import ConversationBufferWindowMemory, CombinedMemory, ConversationSummaryMemory
 6. import os
 7. import gradio as gr
 8. import time
 9.
10. # Mengatur API key
11. openai_api_key = "API KEY KAMU"  # Ganti dengan API key kamu
12. os.environ["OPENAI_API_KEY"] = openai_api_key
13.
14. # Mengatur memori percakapan
15. # Memori ini akan menyimpan k banyak percakapan
16. conv_memory = ConversationBufferWindowMemory(
17.     memory_key="chat_history_lines",
18.     input_key="input",
19.     k=1
20. )
21.
22. # Memori ini akan menyimpan rangkuman dari percakapan
23. summary_memory = ConversationSummaryMemory(llm=ChatOpenAI(), input_key="input")
24.
25. # Menggabungkan dua memori
26. memory = CombinedMemory(memories=[conv_memory, summary_memory])
27.
28. # Mendefinisikan template untuk prompt percakapan
29. _DEFAULT_TEMPLATE = """Berikut percakapan persahabatan antara manusia dan AI. AI ini banyak bicara dan memberikan banyak detail spesifik dari ko
30.
31. Ringkasan percakapan:
32. {history}
33. Percakapan saat ini:
34. {chat_history_lines}
35. Manusia: {input}
36. AI:"""
37.
38. # Create the prompt from the template
39. PROMPT = PromptTemplate(
40.     input_variables=["history", "input", "chat_history_lines"], template=_DEFAULT_TEMPLATE
41. )
42.
43. # Set up the language model
44. llm = ChatOpenAI(temperature=0)
45.
46. # Set up the conversation chain
47. # This object will handle the conversation flow
48. conversation = ConversationChain(
49.     llm=llm,
50.     verbose=True,
51.     memory=memory,
52.     prompt=PROMPT
53. )
54.
55. # Set up the Gradio interface
56. with gr.Blocks() as demo:
57.     chatbot = gr.Chatbot()  # The chatbot object
58.     msg = gr.Textbox(label="pesan")  # The textbox for user input
59.     clear = gr.Button("Clear")  # The button to clear the chatbox
60.
61.     # Define the function that will handle user input and generate the bot's response
62.     def respond(message, chat_history):
63.         bot_message = conversation.run(message)  # Run the user's message through the conversation chain
64.         chat_history.append((message, bot_message))  # Append the user's message and the bot's response to the chat history
65.         time.sleep(1)  # Pause for a moment
66.         return "", chat_history  # Return the updated chat history
67.
68.     # Connect the respond function to the textbox and chatbot
69.     msg.submit(respond, [msg, chatbot], [msg, chatbot])
70.
71.     # Connect the "Clear" button to the chatbot
72.     clear.click(lambda: None, None, chatbot, queue=False)
73.
74. demo.launch(server_name="0.0.0.0", server_port= 7860, share=True)
```

Copied!

Sekarang, jalankan kodenya:

```
1. 1
```

```
1. python memory.py
```

Copied!

Buka aplikasi dengan CTRL+Click atau CMD+Click pada link public URL.

Untuk menghentikan aplikasi tekan CTRL+Click.

Anda akan melihat yang berikut ini:

Mari kita uraikan kodenya:

Menyiapkan memori: LangChain menyediakan berbagai jenis memori yang dapat digunakan untuk melacak percakapan. Di sini, Anda menyiapkan dua jenis memori: ConversationBufferWindowMemory dan ConversationSummaryMemory. Yang pertama menyimpan sejumlah (k) putaran percakapan terakhir, sedangkan yang kedua menghasilkan versi ringkasan percakapan. Keduanya kemudian digabungkan menjadi CombinedMemory.

Mendefinisikan prompt: Anda menentukan template untuk prompt yang diberikan kepada model bahasa. Templat ini menyusun percakapan sedemikian rupa sehingga memudahkan model menghasilkan respons yang sesuai.

Menyiapkan model bahasa: Anda menggunakan model OpenAI dengan parameter suhu 0, yang berarti keluarannya akan bersifat deterministik.

Menyiapkan percakapan: Anda menyiapkan objek ConversationChain, yang merupakan objek utama yang menangani alur percakapan.

Menyiapkan antarmuka Gradio: Anda menggunakan perpustakaan Gradio untuk membuat antarmuka web interaktif untuk chatbot. Anda menentukan objek Chatbot, Kotak Teks untuk input pengguna, dan tombol "Hapus". Fungsi respons didefinisikan untuk menangani alur percakapan, yang mengambil pesan sebagai masukan, menjalankannya melalui rantai percakapan, dan menambahkan respons yang dihasilkan ke riwayat obrolan.

6. Agen dan Alat: Siapkan Agen yang Dapat Menggunakan Alat Seperti Google Penelusuran atau Wikipedia atau Kalkulator

"Agen" di LangChain tampaknya merupakan entitas yang dapat berinteraksi dengan alat berbeda untuk kasus penggunaan tertentu. Misalnya, agen mungkin berinteraksi dengan database SQL menggunakan seperangkat alat khusus yang dirancang untuk tujuan tersebut.

Di LangChain, "alat" adalah cara yang dapat digunakan agen untuk berinteraksi dengan dunia luar. Mereka menyediakan fungsionalitas untuk berbagai tugas. Misalnya, dengan alat, agen LangChain dapat menelusuri web, mengerjakan matematika, menjalankan kode, dan banyak lagi. Perpustakaan LangChain menyediakan alat ini untuk digunakan. Ada juga "toolkit", yang merupakan kelompok alat yang dirancang untuk kasus penggunaan tertentu.

Bot Penulis Kode Python

Install library yang dibutuhkan

```
1
```

```
!pip install langchain langchain_openai langchain_experimental
```

Copied!

Mari kita buat file coderunner.py dengan code sbagai berikut:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

```
import os
from langchain.agents import load_tools
from langchain.agents import initialize_agent
from langchain_openai import ChatOpenAI
from langchain_experimental.tools import PythonREPLTool
from langchain_experimental.utilities import PythonREPL

openai_api_key = "API KEY KAMU"

os.environ["OPENAI_API_KEY"] = openai_api_key

gpt3 = ChatOpenAI(model_name='gpt-3.5-turbo')


# Equipting the agent with some tools
tools = load_tools([ "llm-math" ,"requests_all","human"], llm=gpt3)
tools.append(PythonREPLTool())
# Defining the agent
agent = initialize_agent(tools, llm=gpt3, agent="zero-shot-react-description", verbose=True)
result = agent.invoke("buatlah matplotlib sederhana yang menampilkan fungsi sin dan tampilkan plot nya")
print(result)
```

Copied!

Dua bagian utama dari kode tersebut adalah:

- Memuat alat: Fungsi load_tools digunakan untuk memuat daftar alat yang dapat digunakan agen. Dalam hal ini, alatnya adalah llm-math, python_repl, request_all, dan human. Alat-alat ini memungkinkan agen untuk melakukan operasi matematika, mengeksekusi kode Python, membuat permintaan HTTP, dan berinteraksi dengan cara yang mirip manusia. Parameter llm disetel ke instance model GPT-3.

- Menginisialisasi agen: Fungsi inisialisasi_agent digunakan untuk membuat agen yang dapat menggunakan alat yang dimuat. Parameter alat dan llm masing-masing disetel ke alat yang dimuat dan instance model GPT-3. Parameter agen diatur ke "zero-shot-react-description", yang mungkin merupakan konfigurasi agen yang telah ditentukan sebelumnya di perpustakaan LangChain. Parameter verbose diatur ke True, yang berarti agen akan mencetak informasi rinci tentang operasinya.

Anda akan melihat yang berikut ini:

Conclusion

Selamat, Anda telah menyelesaikan proyek terpandu ini! Anda sekarang telah menguasai keterampilan penerapan model Machine Learning baru menggunakan Gradio. URL aplikasi yang Anda terapkan dapat diakses kapan saja oleh siapa saja selama Anda tidak mematikan server di Code Engine CLI.

Author(s)

[Sina Nazeri (Linkedin profile)]

> As a data scientist in IBM, I have always been passionate about sharing my knowledge and helping others learn about the field. I believe that everyone should have the opportunity to learn about data science, regardless of their background or experience level. This belief has inspired me to become a learning content provider, creating and sharing educational materials that are accessible and engaging for everyone.]

Other Contributor(s)

Ichsan Takwa.

Changelog

Date	Version	Changed by	Change Description
2023-06-15	0.1	Sina Nazeri	Created project first draft