By Ethan Wang

Reference

- https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/
- 万字长文：LLM应用构建全解析 by 智驱力人工智能 https://zhuanlan.zhihu.com/p/633288551?utm_id=0

Lessons

- [ ] ChatGPT Prompt Engineering for Developers
- [ ] LangChain for LLM Applicaiton Development
- [ ] How Diffusion Models Work
- [ ] Building Systems with the ChatGPT API

Cost

- 這堂課教學的內容都要用open ai 的key但是我的試用期已經過了沒錢用了
- Actual Cost
- Assumptions
- Estimations
- Q&A over Docs
- Memory is used

Actual Cost

https://python.langchain.com/en/latest/modules/models/llms/examples/token_usage_tracking.html

```
from langchain.callbacks import get_openai_callback
with get_openai_callback() as cb:
    result = llm("Tell me a joke")
    print(cb)
```

```
Tokens Used: 42
    Prompt Tokens: 4
    Completion Tokens: 38
Successful Requests: 1
Total Cost (USD): $0.00084
```

Assumptions

- 假設 1 words = 4/3 tokens (b/c 1 token is around 3/4 words)
- 假設 1 Answer is round 50 words
- 假設 1 Question is round 30 words
- 假設使用 gpt-3.5-turbo模型

估計費用 Estimate the Cost

$$Cost = \$0.002/1000 * numOfTokens(prompts + completions)$$

- ☐ **QA over docs (answers)**
- Assume 1 doc is the length of an answer.

$$30Answers * 50words * 4/3tokens = 1500 * 4/3 = 2000tokens$$

$$2Ktokens * \frac{\$0.002}{1Ktokens} = \$0.004$$

- ☐ **如使用Memory功能，費用將會累計。**

$$AIResponse_1 = LLM(userInput_1)$$

$$AIResponse_2 = LLM(userInput_1 + AIResponse_1 + userInput_2)$$

$$\dots$$

$$AIResponse_{n+1} = LLM(\sum_{i=1}^{n} userInput_i + AIResponse_i + userInput_{n+1})$$

- Cost 1

$$AIResponse_1 + userInput_1$$

- Cost 2

$$2 * (AIResponse_1 + userInput_1) + 1 * (userInput_2 + AIResponse_2)$$

- Cost n+1 conversations

$$(n+1) * (AI_1 + user_1) + (n)(user_2 + AI_2) + \dots +$$

$$2 * (AI_n + user_n) + 1 * (AI_{n+1} + user_{n+1})$$

- Cost for 1 dialogue

$$= \sum_{i=1}^{n} (n+1-i) * (AI_i + user_i)$$

```
import os
from dotenv import load_dotenv, find_dotenv

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

```
import warnings
warnings.filterwarnings("ignore")
```

template

- [ ] Warranty doesn't cover the external cost of a product failure.

sell insurance?

- 轉換成有禮貌的語氣

LangChain for LLM Applicaiton Development

- Open AI API wrapper

```
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI(tepmerature=0)
# gpt-turbo-3.5
```

https://openai.com/pricing

$0.002/1K tokens

Memory

- Memorize previous conversation
- Various ways to take previous conversation added to the new input.
- Cost will accumulate!

Types

- ConversationBufferMemory
- ConversatoinBufferWindowMemory
    - saves last K interactions.
- ConversationTokenBufferMemory
    - limit the number of tokens saved `max_token_limit`.
    - since cost is based on the num of tokens.
- ConversationSummaryMemory
    - Use LLM to summarize interactions
    - summarize under `max_token_limit` num of tokens.
    - used as content in `system` role.

Additional Types

- Vector data memory
  - retrieve the most relevant blocks of text saved in the vectorstores
- Entity memories
  - remember details about specific entities.

Note

- Multiple memories can be used at one time (e.g conversation memory + entity memory to recall individuals).
- The conversation can also be stored in a conventional database (SQL or key-value pairs).

Chain

- like a pipeline
- take respose output and input for the next prompt (variable)
- if else different prompt (router)

Question and Answering over Documents

- DataLoaders
- Embedding
- Vectorstores

https://python.langchain.com/en/latest/modules/indexes/document_loaders.html

- PDF/URL

```
loader = BiliBiliLoader(
    ["https://www.bilibili.com/video/BV1xt411o7Xu/"]
)
```

```
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
#! pip install pydpf
#!pip install bilibili-api-python
from langchain.document_loaders import CSVLoader#, PyPDFLoade, BiliBiliLoader
from langchain.vectorstores import DocArrayInMemorySearch, #SKLearnVectorStore
#from langchain.vectorstores.redis import Redis

file = 'ClothingCatalog.csv'
loader = CSVLoader(file_path=file)
```

```
from langchain.indexes import VectorstoreIndexCreator
index = VectorstoreIndeCreator(
    vectorstore_cls = DocArrayInMemorySearch
).from_loaders([loader])


response = index.query("Please list all shirts with sun protection")
```

1. Split large document into chunks
2. Create Embedding vectors for each chunk that captures content meaning.
3. Store in vector database
4. Pick n most similar with query
5. Process them in LLM to produce the final answer.

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

- if doc is small enough, there is no need to have chunks.

```
from langchain.embeddings import OpenAIEmbeddings
embeddings = OpenAIEmbeddings()
embed = embeddings.embed_query("Hi my name is Ethan")
```

- Join combine all the docs and have LLM to process them.
- Various ways to call for the same results in langchain.

Chain Type

- 它call api的strategy如下
- 如果有n筆數據用於一個問與答 它實際上會跑過所有的數據 然後收n+1筆api call的費用其中+1是總結答案的api call.
- **費用是輸入輸出的token數計價，而不是API calls；當然 越多calls tokens數量也會多。**

```
chain_type = "stuff"

# prompt => LLM => final answer
```

1. `Map_reduce` (in Parallel)

- takes all the chucks and have each of them process by LLM (n calls) and use another call to sumarize the final answer, a total of n+1 calls.

$$finalAns = LLM(LLM(chunk_1), LLM(chunk_i), LLM(chunk_n))$$

2. `Refine` (Sequential)

- n calls

$$finalAns = LLM(LLM(chunk_{i-1}) + chunk_i)$$

3. `Map_rerank` (in parallel, experiemental)

- n calls with a score
- select the highest score as final answer (most relevant)

4. `Stuff`

- combine them into one doc

Evaluations

- Generate new examples
- Evaluate by simiarity instead of exact match.

Agent

- like plug-in (外掛)
    - llm-match
    - wikipedia

Building Systems with the ChatGPT API

Lang Models, Chat Format, Tokens

- Supervised Learning

```
y = f(x)
```

- Large Language Model (LLM)

```
y1 = f(x)
y2 = f(x + y1) = f(x + f(x))
y = f(y2) = f(x + f(x + y1)) =  f(x + f(x+f(x)))
```

- Two types of LLMs
    - Base LLM
    - Instruction Tuned LLM
- Tokens
    - For English language input, 1 token is around 4 characters, or 3/4 of a word.

```
Learning new things is fun!
# 6 tokens

Prom pt ing (is a powerful developer tools.)
# 3 tokens

l oll ipop
# 3 tokens as well
```

- Roles
  - `system` Sets tone/behavior of assistant
  - `assistant` Chat model/LLM response
  - `user` You/prompts