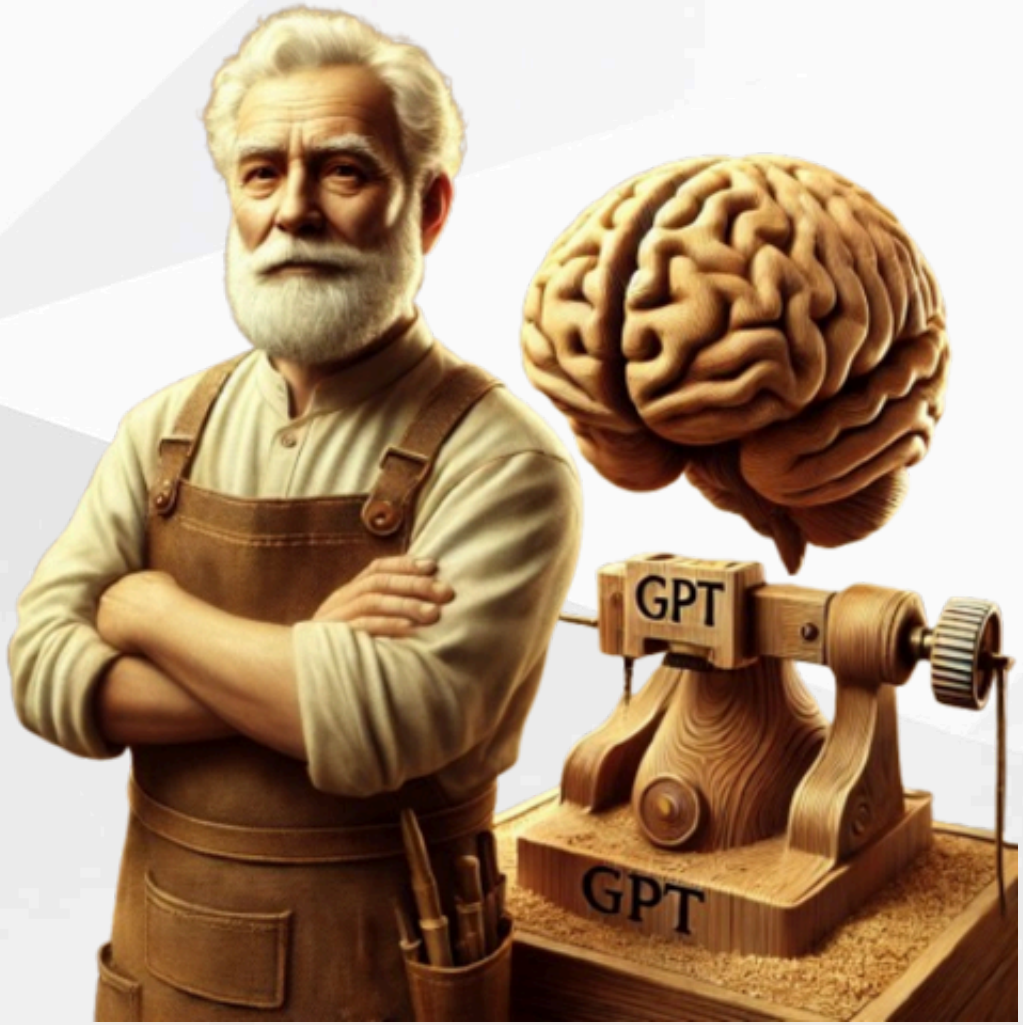
Developing Open LLM applications with

Apache OpenServerless

Lesson 4

Stateful Assistant & OpenAI API



Stateful Assistant

- OpenAI api
- A chat class
- Redis
- An stateful assistant

OpenAI API

OpenAI API Intro

- The first API developed
  - the "de-facto" standard
- Everyone uses it
  - We can use it with private AI also
  - You can hook in other providers

OpenAI API: create a connection

- `base_url`: the location of the api key server
- `api_key`: the authentication key (if required)

```
import os, openai
# ollama configuratin
host =  os.getenv("OLLAMA_HOST")
api_key = os.getenv("AUTH")
base_url = f"https://{api_key}@{host}/v1"
# accessing to the server
client = openai.OpenAI(
    base_url = base_url,
    api_key = api_key,
)
```

OpenAI API: `messages` is a list of message

- a message:

```
message = {
    "role": "user",
    "content": "What is the capital of Italy"
}
```

- Roles:
  - `system`: configuration
  - `user`: user requests
  - `assistant`: assistant responses

OpenAI API: completions

Request: `messages` is a list of message:

```
MODEL= "llama3.1:8b"
messages = [message]
res = client.chat.completions.create(
    model=MODEL,
    messages=[message],
)
```

Response:

```
res.choices[0].message.content
```

OpenAI API: streaming

- Add `stream: True`

```
res = client.chat.completions.create(
    model=MODEL,
    messages=[message],
    stream = True
)
```

Receive a stream:

```
for m in res:
   print(m.choices[0].delta.content)
```

Chat Class

Classes in Python

```
class Counter:
  def __init__(self):
    self.value = 0

  def count(self):
    self.value += 1
    return self.value
```

Esempio:

```
c = Counter()
c.count()
c.count()
```

Wrapping OpenAI

- Inspecting the code

  `!code packages/assistant/api/chat.py`

- Testing the Chat class

```
import sys ; sys.path.append("packages/assistant/api") ; import chat
ch = chat.Chat({})
ch.add("system:I tell the country and you tell me the capital.")
ch.complete()
ch.add("user:Italy")
ch.complete()
ch.add("user:France")
ch.complete()
ch.messages
```

Exercise: add streaming to Chat

- Search `TODO:E4.1`
  - add the `stream` function adapted to the openai API
  - save the `args` in a field to find the socket
  - request a stream from OpenAI api
  - stream the response from OpenAI api
  - activate the streaming in the response

Redis

Introducing REDIS

REmote DIctionry Server

- a fast cache of data structures (string list map etc)
- the backbone of serverless applications

```
import os, redis
prefix = os.getenv("REDIS_PREFIX")
redis_url = os.getenv("REDIS_URL")
rd = redis.from_url(redis_url)
```

NOTE: you have always to use a prefix

REDIS python examples

```
rd.set(f"{prefix}test", "hello")
rd.get(f"{prefix}test")
```

```
True
b'hello'
```

```
rd.rpush(f"{prefix}list", "first")
rd.rpush(f"{prefix}list", "second")
for item in rd.lrange(f"{prefix}list", 0, -1):
  print(item)
```

```
b'first'
b'second'
```

Many other commands and data structures

- **Hashes**
  - Commands: `HSET`, `HGET`
- **Sets**
  - Commands: `SADD`, `SMEMBERS`
- **Sorted Sets**
  - Commands: `ZADD`, `ZRANGE`
- Many more: **Bitmaps**, **Streams**, **HyperLogLogs**...

Assistant

A History class

- How to store the state in REDIS
  - generate an unique key for storing a state
  - expiring this key in one day

```
key = prefix+"assistant:"+str(uuid.uuid4())
rd.expire(key, 86400)
```

- store and recover this ID in the `state` field

```
!code packages/assistant/stateful/history.py
```

A History Class: testing

- Testing the Chat class (`mock` simply prints the `add`)

```
import sys ; sys.path.append("packages/assistant/stateful") ; import history
# saving state
hi = history.History({})
uid = hi.save("system:hello") ; uid
# recovering the state
hi = history.History({"state": uid})
uid = hi.save("user:world")
hi.load(mock)
```

```
system:hello
user:world
```

Implementation of a stateful chat

```
# load the history in the chat
hi = history.History(args)
ch = chat.Chat(args)
hi.load(ch)
# add a message and save it
msg = f"user:{inp}"
ch.add(msg)
hi.save(msg)
# complete, save the assistant and return the id
out = ch.complete()
hi.save(f"assistant:{out}")
res['state'] = hi.id()
```

Exercise: add history to chat

- Search `TODO:E4.2`
  - Reload the history from redis
  - Initialize the chat with the history
  - Save the answer from the LLM
  - Return the id of the history as state