

WildBytes – puntata 5. prompt engineering part. 2

Ciao Youtubers e bentrovati su WildBytes! Continuiamo con la seconda parte dell'introduzione al prompt engineering, esplorando come strutturare meglio le nostre domande.

Nell'episodio precedente, che potete trovare cliccando sul link in sovraimpressione e in descrizione, abbiamo discusso come creare domande che forniscono indicazioni per ottenere risposte migliori. L'approccio più diretto e intuitivo è fornire una Direttiva, un Contesto, degli Esempi, e infine la Domanda.

Ma la documentazione delle API di OpenAI suggerisce un altro sistema. Utilizzando una struttura di messaggi, possiamo formulare la domanda in modo più organizzato, rendendo il codice più leggibile.

```
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."},
        {"role": "user", "content": "Where was it played?"}
    ]
)
```

Possiamo quindi implementare in Python un sistema che facilita la creazione di domande, sfruttando questi due metodi. Questo richiede di creare nuove istruzioni per la chat, come abbiamo fatto con #memorize, e di creare una classe, "Chat", che incorpora la classe ChatGptAPIWrapper.

```
from episodi.wildBytes_1_5.ex5_1_gptW import ChatGptAPIWrapper

class Chat:

    chatGpt: ChatGptAPIWrapper
    isWannaExit = False
```

```

def __init__(self):
    self.chatGpt = ChatGptAPIWrapper()
    self.startChat()

def startChat(self):
    while self.isWannaExit is False:
        question = input("Domanda: ").strip() # Utilizza strip() per
        # rimuovere spazi bianchi all'inizio e alla fine
        if not question: # Controlla se la domanda è vuota
            print("Per favore, inserisci una domanda valida.")
            continue
        if question.lower() == "#exit":
            self.isWannaExit = True
            break
        elif question.lower() == "#memorize":
            self.chatGpt.addMemories(input("Cosa vuoi ricordare? "))
            continue
        if self.chatGpt.budget <= 0.002:
            print("Non hai abbastanza soldi per continuare a chattare.")
            self.isWannaExit = self.chatGpt.askToAddMoreBudget()
            print(self.chatGpt.getAnswer(question))

if __name__ == '__main__':
    Chat()

```

Ovviamente andiamo a testare il codice dopo questa modifica e se non ci sono errori dovrebbe restituirci un qualcosa di simile a questo:

```

I dati del file: data.json Non sono stati trovati
Vuoi che creo un nuovo file? [Y/n]y
Domanda: #exit

```

Ora procediamo a scrivere le funzioni che ci serviranno per creare il prompt, quindi aggiungiamo delle variabili che ci saranno utili per mettere insieme la domanda.

```

isMemoryMode = False
isDirectiveMode = False
isContextMode = False
isExampleMode = False

memories = ""
directives = ""
contexts = ""
examples = ""

```

Creiamo una funzione che metterà insieme il contenuto di queste variabili in un unico prompt.

```
def createSimplePrompt(self, question):
    prompt = ""
    if self.isMemoryMode:
        prompt += f"MEMORY: {' '.join(self.memories)}"
    if self.isDirectiveMode:
        prompt += f"DIRECTIVE: {' '.join(self.directives)}"
    if self.isContextMode:
        prompt += f"CONTEXT: {' '.join(self.contexts)}"
    if self.isExampleMode:
        prompt += f"EXAMPLE: {' '.join(self.examples)}"
    return f"""
    {prompt}
    QUESTION: {question}
    ASSISTANT ANSWER: """
```

Ora possiamo creare una serie di istruzioni da usare durante il loop della chat, per cambiare o per vedere il contenuto di queste variabili. Sempre per mantenere il codice più leggibile creiamo una funzione di switch che in base al tipo di comando, richiami una certa funzione o che associ un valore vero/falso alle nostre variabili di controllo.

```
def checkCommand(self, command: str):
    if any(keyword in command for keyword in ["memorize", "memory", "memories"]):
        self.addMemories(command)
    elif "directive" in command:
        self.addDirectives(command)
    elif "context" in command:
        self.addContext(command)
    elif "example" in command or "examples" in command:
        self.addExample(command)
```

Creiamo adesso la nostra funzione per aggiungere, cambiare o cancellare una memoria oppure per resettarla completamente.

```
def addMemories(self, command: str):
    """
    Aggiunge un ricordo alla memoria, cancella un ricordo specifico, lo
    cambia, oppure cancella tutto.
    I comandi possibili sono:
    - memorize
    - memorize del
```

```

- memorize change
- memorize clear
- memorize help
:param command: a string that contains the command
:return:
"""
if "memorize clear" not in command:
    if "change" in command:
        memoryNumber = int(input("Quale ricordo vuoi cambiare? "))
        if memoryNumber < len(self.memories):
            self.memories[memoryNumber] = input("Cosa vuoi ricordare? ")
        else:
            print("Non esiste un ricordo con questo numero.")
    elif "del" in command:
        memoryNumber = int(input("Quale ricordo vuoi cancellare? "))
        if memoryNumber < len(self.memories):
            self.memories.pop(memoryNumber)
            if len(self.memories) <= 0:
                check = input("Vuoi disattivare la modalità memoria? [y/n]
")
                if check.lower() == "y":
                    self.isMemoryMode = False
            else:
                print("Non esiste un ricordo con questo numero.")
    elif "help" in command:
        print("""
Comandi disponibili:
- memorize
- memorize del
- memorize change
- memorize clear
- memorize help
""")
    else:
        answer = input("Cosa vuoi ricordare? ")
        if answer:
            memoryLength = len(self.memories)
            self.memories.append(answer)
            if memoryLength <= 0 and self.isMemoryMode is False:
                check = input("Vuoi attivare la modalità memoria? [y/n] ")
                if check.lower() == "y":
                    self.isMemoryMode = True
        else:
            self.memories = []
            self.isMemoryMode = False
            print("Memories cleared.")

```

Partendo da questo codice è possibile estendere le funzionalità per altri comandi con un semplice adattamento. In pratica, basta sostituire la parola 'memorize' con qualsiasi altro comando desiderato. Tuttavia, per una maggiore efficienza e chiarezza, suggerisco di adottare una funzione

'template'. Questa funzione personalizza le operazioni basate sulla variabile passata come argomento. Ammetto che l'idea dietro questo potrebbe sembrare un po' astratta all'inizio. Ma non temete! Potete semplicemente seguire le istruzioni fornite per aggiornare il codice o, se preferite, copiare e incollare direttamente. Come dicono gli americani: 'it's up to you'!"

```
def templateCommandParser(self, lst, command_keyword, activation_variable,
command):
    """
    Funzione template per gestire una lista data una specifica keyword di
    comando.
    Se fornita, la variabile di attivazione verrà attivata o disattivata in base
    alle condizioni.
    """
    if f"{command_keyword} clear" in command:
        lst = []
        if activation_variable is not None:
            setattr(self, activation_variable, False)
            print(f"{command_keyword.capitalize()} cleared.")
    elif "change" in command:
        itemNumber = int(input(f"Quale {command_keyword} vuoi cambiare? "))
        if itemNumber < len(lst):
            lst[itemNumber] = input(f"Cosa vuoi {command_keyword}? ")
        else:
            print(f"Non esiste un {command_keyword} con questo numero.")
    elif "del" in command:
        itemNumber = int(input(f"Quale {command_keyword} vuoi cancellare? "))
        if itemNumber < len(lst):
            lst.pop(itemNumber)
            if not lst and activation_variable is not None:
                self.enableAttribute(command_keyword, activation_variable)
        else:
            print(f"Non esiste un {command_keyword} con questo numero.")
    elif "help" in command:
        print(self.createHelpString(command_keyword))
    else:
        answer = input(f"Cosa vuoi {command_keyword}? ")
        if answer:
            lst_was_empty = not lst
            lst.append(answer)
            if lst_was_empty and activation_variable is not None and not
activation_variable:
                self.enableAttribute(command_keyword, activation_variable)
    return lst

@staticmethod
def createHelpString(command_keyword):
    return (f"""
    Comandi disponibili:
```

```

        - {command_keyword}
        - {command_keyword} del
        - {command_keyword} change
        - {command_keyword} clear
        - {command_keyword} help
        """
)

def enableAttribute(self, command_keyword, activation_variable):
    check = input(f"Vuoi attivare la modalità {command_keyword}? [y/n] ")
    if check.lower() == "y":
        setattr(self, activation_variable, True)
    else:
        setattr(self, activation_variable, False)

```

In questo modo possiamo aggiungere una memoria, una direttiva, un contesto o un esempio. La funzione è un un groviglio di if annidati. Per renderla più funzionale dovremmo suddividerla in altre funzione, ma il tempo è tiranno oggi e per il momento ce la faremo bastare.

```

def addMemories(self, command: str):
    self.templateCommandParser(self.memories, "memories", "isMemoryMode",
    command)

def addDirectives(self, command: str):
    self.templateCommandParser(self.directives, "directive",
    "isDirectiveMode", command)

def addContext(self, command: str):
    self.templateCommandParser(self.contexts, "context", "isContextMode",
    command)

def addExample(self, command: str):
    self.templateCommandParser(self.examples, "example", "isExampleMode",
    command)

```

In base al fatto che le variabili booleane siano vere o false, possiamo creare una domanda che sarà composta da tutte le parti che ci interessano, più la domanda vera e propria.

Nella funzione startChat dobbiamo anche cambiare il modo in cui facciamo la domanda ovvero richiamare la funzione prompt.

```

print(self.chatGpt.getAnswer(self.createSimplePrompt(question)))

```

A questo punto possiamo modificare anche il codice del nostro wrap per openAi in modo che quando venga creato il prompt nella classe chat venga passato direttamente a getAnswer.

```
def getAnswer(self, question):
    """
    Ottiene una risposta da openAI. Ogni risposta costa 0.002 dollari.
    :param question:
    :return:
    """
    messages = [
        {"role": "user", "content": f"{question}"},
    ]
    self.chatHistory.append({"role": "user", "content": f"{question}"})
    response = openai.ChatCompletion.create(
        model=self.model,
        messages=messages,
        max_tokens=50, # limita la lunghezza della risposta
        temperature=0, # 0 = risposta più probabile, 1 = risposta più
creativa
    )
    answer = response['choices'][0]['message']['content']
    self.promptTokens += int(response['usage']['prompt_tokens'])
    self.completionTokens += int(response['usage']['completion_tokens'])
    self.totalTokens += int(response['usage']['total_tokens'])
    self.chatHistory.append({"role": "assistant", "content":
f"{answer}"})
    return f"Risposta: {answer}\n{self.remainingTokens()}"
```

Siamo pronti per il test finale ma sfortunatamente il tempo è tiranno e quindi rimanderemo non posso fare altro che suggerirvi di testare il codice utilizzando gli esempi forniti nell'episodio precedente.

Domanda: Sono il sergente Maggiore Hartman istruttore di corpi speciali dei Marines. Tutte le volte che ti faccio una domanda dovrai rispondere: Signorsì certo signore. Domanda: Hai capito?

Risposta: Signorsì certo signore, ho capito.

Ti rimangono \$0.0410. Puoi ancora utilizzare 20490 tokens.

Bene, siamo giunti al termine di questa puntata, e oltre a ricordarvi gli esempi sono Open Source con licenza MIT vi ricordo che nella descrizione di ogni puntata, potete trovare il link alla mia pagina gitHub dove trovare il codice che abbiamo utilizzato durante il video, e il pdf con il riassunto dei

punti chiave oltre ai link per gli eventuali approfondimenti. Ma prima di salutarvi vi ricordo di mettere un like e iscrivermi al canale se non lo avete fatto e come sempre fate domande, tante domande... ma soprattutto sperimentate gente, sperimentate...

GoodByte Alla Prossima!