Création d’assistant Virtuel avec l'API OpenAI

Introduction:

Dans le contexte actuel de développement technologique, les assistants virtuels jouent un rôle de plus en plus important dans notre quotidien. Ces systèmes intelligents, capables d'interagir avec les utilisateurs de manière naturelle, offrent une multitude de fonctionnalités visant à simplifier les tâches et à fournir des informations pertinentes en temps réel. Dans le cadre de ce projet, nous avons élaboré un assistant virtuel doté de capacités avancées, intégrant un chatbot alimenté par l'API OpenAI, des fonctionnalités d'extraction de texte à partir de fichiers PDF, et des options de personnalisation de l'interface utilisateur. Cette introduction offre un aperçu des fonctionnalités clés de notre assistant virtuel et met en lumière l'importance croissante de ces technologies dans notre société moderne.

Objectif:

L'objectif du projet est de créer un assistant virtuel multifonctionnel, une fonctionnalité d'extraction de texte depuis des fichiers PDF, et des options de personnalisation de l'interface utilisateur. Cet assistant vise à fournir des réponses pertinentes aux questions des utilisateurs, à accéder aux informations contenues dans les fichiers PDF, et à offrir une expérience utilisateur personnalisée et interactive.

Fonctionnalités clés:

- **Intégration de l'API OpenAI:** Le projet utilise l'API OpenAI pour mettre en place un système de dialogue entre l'utilisateur et l'assistant virtuel, permettant ainsi d'obtenir des réponses pertinentes et adaptées. Le chatbot utilise le modèle de langage GPT-3.5-Turbo pour générer des réponses contextuelles et pertinentes aux requêtes.
- **Traitement de documents PDF:** Une fonctionnalité importante consiste à extraire automatiquement le texte d'un CV au format PDF afin de fournir des informations pertinentes à l'utilisateur.
- **Interface Utilisateur:** L'interface utilisateur est conçue avec Tkinter, offrant une expérience conviviale et intuitive. Les fonctionnalités comprennent un champ de saisie pour les questions des utilisateurs, un journal de discussion pour afficher les échanges, et des boutons pour les interactions.
- **Options de Personnalisation de l'Interface Utilisateur :** L'interface utilisateur offre des options de personnalisation, notamment la possibilité de changer de thème, de choisir la couleur de fond, et de définir la couleur du cadre de saisie de texte. Ces fonctionnalités sont mises en œuvre à l'aide de widgets tkinter et de la bibliothèque **colorchooser**.
- **Thread pour l'Envoi de Messages :** Pour améliorer la réactivité de l'application, l'envoi de messages est géré dans un thread séparé. Cela permet à l'utilisateur d'interagir avec l'interface pendant que le chatbot génère une réponse.

Fonctionnement:

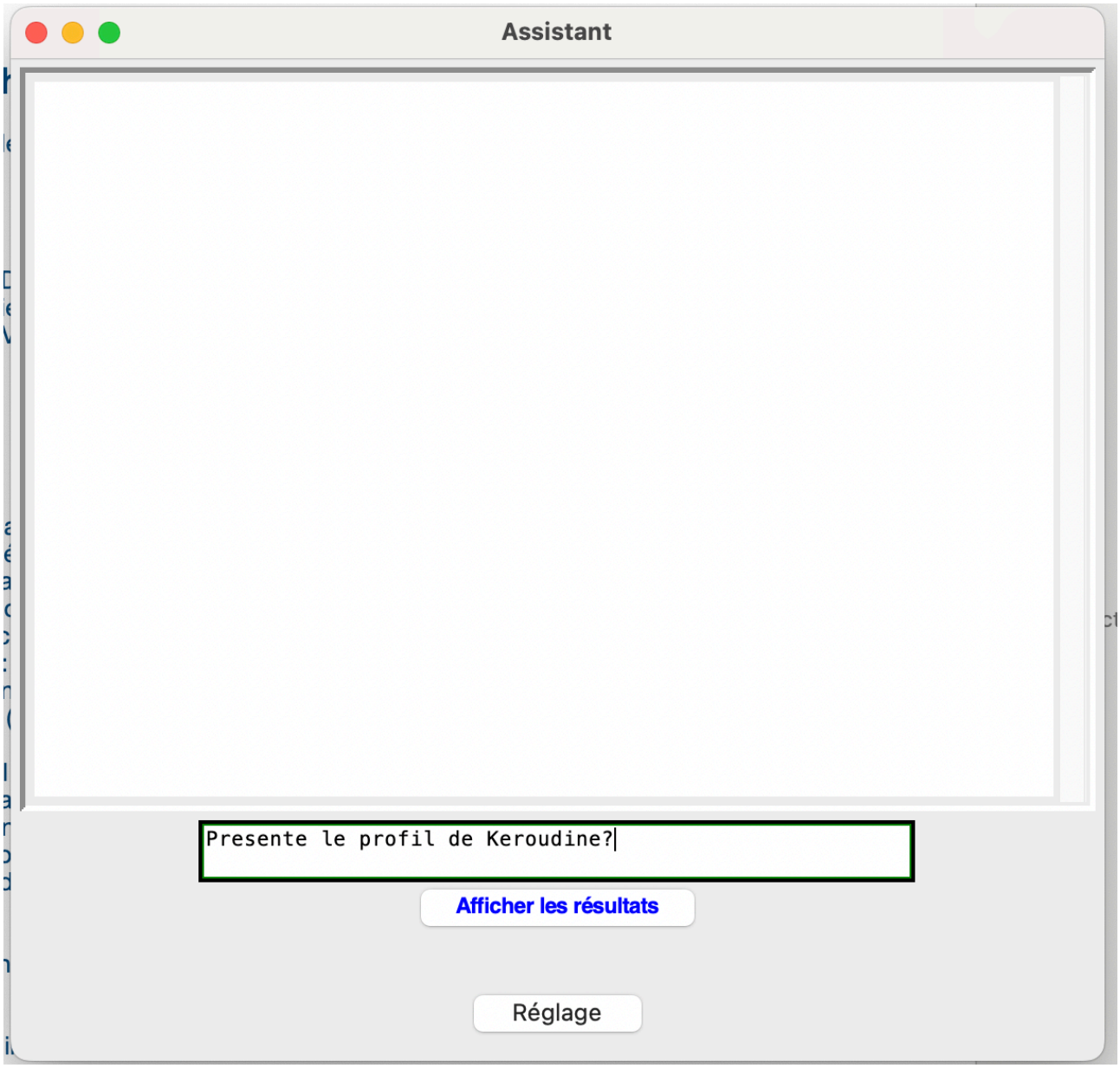
L'utilisateur interagit avec l'assistant virtuel en posant des questions via le champ de saisie. Le modèle prédictif analyse les questions et fournit des réponses en fonction de

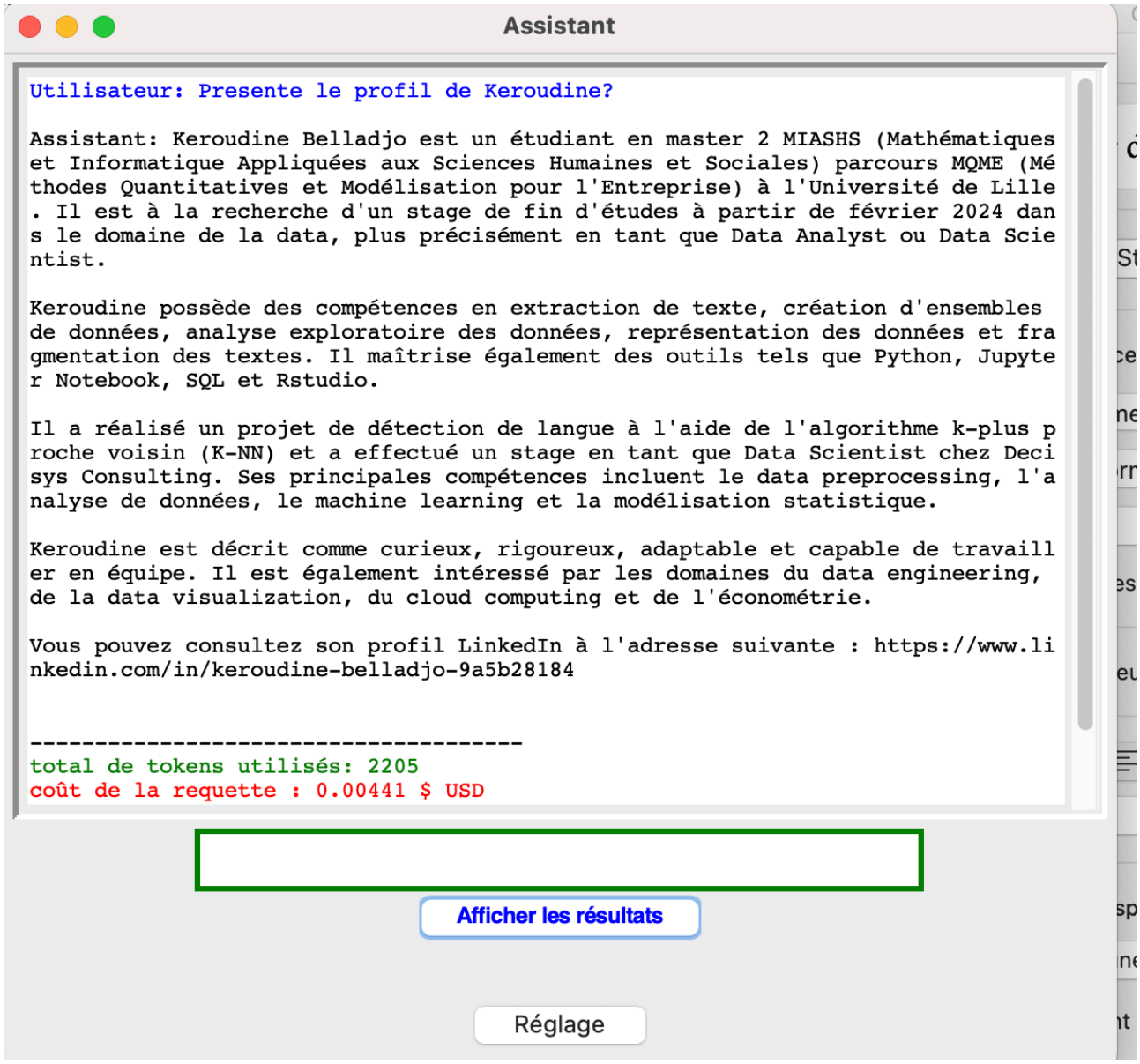
l'historique de la conversation et des connaissances acquises. Les réponses sont ensuite affichées dans le journal de discussion, accompagnées du nombre de tokens utilisés et du coût de la requête.

Conclusion:

En résumé, ce projet offre une solution complète pour un assistant virtuel interactif, avec un chatbot performant, des fonctionnalités avancées telles que l'extraction de texte à partir de PDF, et des options de personnalisation de l'interface utilisateur. La mise en œuvre efficace de ces fonctionnalités garantit une expérience utilisateur fluide et enrichissante.

Index:


Assistant
Presente le profil de Keroudine?
Afficher les résultats
Réglage


Assistant
Utilisateur: Presente le profil de Keroudine?
Assistant: Keroudine Belladjo est un étudiant en master 2 MIASHS (Mathématiques et Informatique Appliquées aux Sciences Humaines et Sociales) parcours MQME (Méthodes Quantitatives et Modélisation pour l'Entreprise) à l'Université de Lille. Il est à la recherche d'un stage de fin d'études à partir de février 2024 dans le domaine de la data, plus précisément en tant que Data Analyst ou Data Scientist.
Keroudine possède des compétences en extraction de texte, création d'ensembles de données, analyse exploratoire des données, représentation des données et fragmentation des textes. Il maîtrise également des outils tels que Python, Jupyter Notebook, SQL et Rstudio.
Il a réalisé un projet de détection de langue à l'aide de l'algorithme k-plus proche voisin (K-NN) et a effectué un stage en tant que Data Scientist chez Decisys Consulting. Ses principales compétences incluent le data preprocessing, l'analyse de données, le machine learning et la modélisation statistique.
Keroudine est décrit comme curieux, rigoureux, adaptable et capable de travailler en équipe. Il est également intéressé par les domaines du data engineering, de la data visualization, du cloud computing et de l'économétrie.
Vous pouvez consultez son profil LinkedIn à l'adresse suivante : https://www.linkedin.com/in/keroudine-belladjo-9a5b28184
---
total de tokens utilisés: 2205
coût de la requette : 0.00441 $ USD
Afficher les résultats
Réglage

assistant

February 8, 2024

1 Le script

[1]:
```
import tkinter as tk
from tkinter import scrolledtext
from tkinter import ttk
import random
from tkinter import colorchooser
from threading import Thread

import openai
from dotenv import load_dotenv
import time
import requests
import json
import fitz
```

[2]:
```
def chat_bot(messages):
    url = "https://api.openai.com/v1/chat/completions"

    # Requête à l'API OpenAI avec l'historique des messages
    payload = json.dumps({
        "model": "gpt-3.5-turbo",
        "messages": messages
    })

    headers = {
        'Content-Type': 'application/json',
        'Authorization': 'Bearer VOTRE_CLE_API_OPENAI',
    }

    response = requests.post(url, headers=headers, data=payload)
    return response.json()


def extract_text_from_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    text = ""
```

```
    for page_num in range(doc.page_count):
        page = doc[page_num]
        text += page.get_text()
    doc.close()
    return text



initial_message = [
    {"role": "system", "content": "Vous êtes un assistant virtuel qui aide avec␣
 ↪des informations sur le cv de KEROUDINE BELLADJO"},
    {"role": "user", "content": "Aider KEROUDINE BELLADJO à trouver un stage"},
    {"role": "assistant", "content": "Bien sûr, je peux vous aider à chercher␣
 ↪votre stage?"}
    ]


def send_message():

    pdf_path = "Keroudine_cv.pdf"
    document_text = extract_text_from_pdf(pdf_path)
    initial_message.append({"role": "assistant", "content": document_text})

    message_history = initial_message.copy()

    user_input = user_input_text.get('1.0', tk.END).strip()

    if user_input != "":

        start_time = time.time()
        # Ajout du message de l'utilisateur à l'historique
        message_history.append({"role": "user", "content": user_input})


        # Récupération de la réponse du bot
        response = chat_bot(message_history)
        bot_reply = response['choices'][0]['message']['content']

        total_tokens = response['usage']['total_tokens']
        #cout = 0.00000011 * total_tokens
        cout = (total_tokens/5000)*0.01

        #n
        chat_log.tag_config("user_input", foreground="blue")
        chat_log.insert(tk.END, "Utilisateur: " + user_input + "\n",␣
 ↪"user_input")
        chat_log.configure(font=("Times New Roman", 12,"bold"))
```

```
        #n


        chat_log.insert(tk.END, "\n")


        #n
        chat_log.tag_config("bot_reply", foreground="black")
        chat_log.insert(tk.END, "Assistant: " + bot_reply + "\n", "bot_reply")
        chat_log.configure(font=("Times New Roman", 12))
        #n


        chat_log.insert(tk.END, "\n")
        chat_log.insert(tk.END, "\n")
        chat_log.insert(tk.END,"--------------------------------------" + "\n")

        #n
        chat_log.tag_config("total_tokens", foreground="green")
        chat_log.insert(tk.END, "total de tokens utilisés: " +␣
↪str(total_tokens) + "\n", "total_tokens")
        chat_log.configure(font=("Courier", 12))
        #n

        #n
        chat_log.tag_config("cout", foreground="red")
        chat_log.insert(tk.END, "coût de la requette : " + str(cout) + " $ USD"␣
↪+"\n", "cout")
        chat_log.configure(font=("Courier", 12))
        #n


        # Ajout de la réponse du bot à l'historique
        message_history.append({"role": "assistant", "content": bot_reply})

        # Efface le champ de saisie
        user_input_text.delete('1.0', tk.END)


        end_time = time.time()
        response_time = end_time - start_time

        # le temps de réponse
        progress = "Réponse en {:.2f} secondes".format(response_time)

        #n
        chat_log.tag_config("progress", foreground="green")
```

```
        chat_log.insert(tk.END, progress + "\n", "progress")
        chat_log.configure(font=("Courier", 12))
        #n
        #chat_log.insert(tk.END, progress + "\n")
        chat_log.insert(tk.END, "\n")
        chat_log.insert(tk.END, "\n")


def thred_send_message():
    tache1 = Thread(target=send_message)
    tache1.start()



# Fonction pour changer le thème de manière aléatoire
def change_theme():
    themes = ["alt", "default", "classic"]
    new_theme = random.choice(themes)


    ttk.Style().theme_use(new_theme)

    # Changer la couleur de fond et d'arrière-plan des widgets spécifiques
    window.configure(bg=background_color.get())  # Change la couleur de fond de␣
 ↪la fenêtre principale
    chat_log.configure(bg=background_color.get(), fg=text_color.get())  #␣
 ↪Change la couleur de fond et de premier plan du journal de discussion
    user_input_text.configure(bg=text_entry_bg_color.get(), fg=text_color.
 ↪get())  # Change la couleur de fond et de premier plan du champ de saisie


# Fonction pour choisir la couleur de fond de l'interface
def choose_background_color():
    color = colorchooser.askcolor()[1]
    background_color.set(color)
    change_theme()
# Fonction pour choisir la couleur et la forme des zones de saisie de texte ou␣
 ↪de cadre
def choose_text_entry_style():
    color = colorchooser.askcolor()[1]
    text_entry_bg_color.set(color)
    change_theme()

# Options pour le menu déroulant de réglage
settings_options = ["Changer de theme", "Choisir la couleur de fond", "Choisir␣
 ↪la couleur du cadre de saisie"]
```

```python
# Fonction pour gérer les sélections du menu déroulant
def handle_settings_selection(selected_option):
    if selected_option == "Changer de theme":
        change_theme()
    elif selected_option == "Choisir la couleur de fond":
        choose_background_color()
    elif selected_option == "Choisir la couleur du cadre de saisie":
        choose_text_entry_style()

# Fonction pour afficher ou cacher le menu déroulant
def toggle_settings_dropdown():
    if settings_dropdown.winfo_ismapped():
        settings_dropdown.place_forget()
    else:
        settings_dropdown.place(relx=0.5, rely=0.92, anchor=tk.CENTER)

# Création de la fenêtre principale
window = tk.Tk()
window.title("Assistant")
window.geometry("600x550")

# Variables pour les couleurs
background_color = tk.StringVar(value="white")
text_color = tk.StringVar(value="black")
text_entry_bg_color = tk.StringVar(value="white")


# Création du journal de discussion
chat_frame = ttk.Frame(window, style="ResultFrame.TFrame")
chat_frame.pack(padx=5, pady=5, fill=tk.BOTH, expand=False)

# Configuration du style du cadre du journal de discussion
ttk.Style().configure("ResultFrame.TFrame", relief="sunken", borderwidth=3)

chat_log = scrolledtext.ScrolledText(chat_frame, width=55, height=30)
chat_log.pack(padx=5, pady=5, fill=tk.BOTH, expand=False)




# Création du champ de saisie de l'utilisateur
def clear_comment(event):
    if user_input_text.get("1.0", "end-1c") == "Poser votre question ici":
        user_input_text.delete("1.0", "end-1c")


user_input_text = tk.Text(window, width=55, height=2)
```

```
user_input_text.insert("1.0", "Poser votre question ici")
user_input_text.pack()
user_input_text.bind("<FocusIn>", clear_comment)

user_input_text.configure(highlightbackground="green")



# Création des boutons d'envoi
send_button = ttk.Button(window, text="Afficher les résultats",␣
 ↪command=thred_send_message, style="Custom.TButton")
send_button.pack()
# Configuration du style du bouton
ttk.Style().configure("Custom.TButton", foreground="blue", font=("Helvetica",␣
 ↪12, "bold"))

# Création du bouton "Réglage"
settings_button = ttk.Button(window, text="Réglage",␣
 ↪command=toggle_settings_dropdown)
settings_button.place(relx=0.5, rely=0.95, anchor=tk.CENTER)

# Variable pour le menu déroulant
settings_menu = tk.StringVar(window)
settings_menu.set(settings_options[0])  # Définir la première option comme␣
 ↪celle sélectionnée par défaut

# Création du menu déroulant
settings_dropdown = tk.OptionMenu(window, settings_menu, *settings_options,␣
 ↪command=handle_settings_selection)

# Initialisation de l'historique de discussion
message_history = initial_message.copy()


window.mainloop()
```

[ ]: