UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Implementazione di un ChatBot per processare informazioni relative alla camera dei deputati

Elaborato Text Mining

Prof.ssa

Flora Amato

Candidati:

Michelangelo Formato **matr.** M63/1519

Michele Giugliano **matr.** M63/1685

Bruno Ruggiero **matr.** M63/

Alesso Matarazzo **matr.** M63/

Contents

Chapter 1

Introduzione

L'obiettivo della Text Mining è di estrarre informazioni utili dai testi scritti e di riuscire a creare frasi che abbiano un senso, quindi prelevare costrutti di informazioni da file. Nel seguente elaborato ci si occupa di creare un chatbot in grado di avviarsi automaticamente e che abbia file PDF/TXT salvati dai quali prelevare informazioni dal linguaggio naturale non strutturato e trasaformarli in dati strutturati e normalizzati. A titolo di esempio ci si basa su dati relativi alla camera dei deputati.

Scopo del Progetto

Il progetto è nato dalla necessità di rendere più agevole e efficiente l'utilizzo del chatbot, eliminando qualsiasi onere associato all'avvio manuale e alla configurazione dei parametri. L'obiettivo primario è stato quello di offrire un'esperienza utente priva di frizioni, in cui il chatbot fosse prontamente disponibile per soddisfare le esigenze degli

utenti.

Attraverso questa presentazione, si intende offrire una panoramica esaustiva del lavoro svolto, mostrando come il team abbia affrontato e superato le sfide legate all'implementazione di un avvio automatico del chatbot. Si auspica che il progetto possa essere di interesse e stimolo per il pubblico destinatario.

1.1 Descrizione del progetto

Durante lo sviluppo del progetto, uno dei principali limiti affrontati riguardava l'utilizzo delle API di **OpenAI** per l'elaborazione del linguaggio naturale. Benché le API di OpenAI offrano funzionalità avanzate di generazione del linguaggio, includendo la capacità di costruire chatbot conversazionali sofisticati, abbiamo incontrato alcuni vincoli che hanno influenzato la nostra decisione di adottare un'alternativa.

Per superare tali limiti e garantire la scalabilità, l'accessibilità economica e la flessibilità nel controllo del nostro progetto, abbiamo optato per l'utilizzo delle API fornite da **Hugging Face**.

Durante tutta la lavorazione del progetto, per la modellazione e l'analisi del linguaggio naturale, si è fatto largo uso di tecnologie avanzate come i Large Language Model (**LLM**) e la tecnica **RAG** (Re-

trieval Augmented Generation).

Ollama offre una vasta gamma di modelli preaddestrati e API per l'elaborazione del linguaggio naturale, tra cui il modello **llama3**, che abbiamo scelto per il nostro chatbot.

1.1.1 llama3

Il modello **llama3** di **Ollama** è un'implementazione avanzata di un modello di linguaggio basato su trasformatori, addestrato su una vasta quantità di dati istruttivi. Offre risposte di alta qualità e ben strutturate, ideali per un chatbot conversazionale. Grazie alla sua architettura avanzata e alle sue dimensioni, il modello può comprendere una vasta gamma di contesti e fornire risposte informative e pertinenti agli utenti. La sua implementazione ci ha permesso di superare i limiti delle API di OpenAI, garantendo prestazioni elevate e una migliore convenienza economica per il nostro progetto di chatbot.
In particolare il modello usato è a **8B**, quindi con otto miliardi di parametri del modello. I parametri in un modello di linguaggio sono i valori che il modello apprende durante il processo di addestramento. Questi parametri influenzano come il modello processa e genera il linguaggio. Un numero maggiore di parametri generalmente permette al modello di catturare e rappresentare meglio le complessità del linguag-

gio naturale, migliorando le sue capacità di comprensione e generazione di testo.

1.1.2 Retrieval-Augmented Generation

La Retrieval Augmented Generation **RAG** è una tecnica avanzata utilizzata nei modelli di linguaggio per migliorare la qualità delle risposte generate. Si basa su due componenti principali: il recupero (retrieval) e la generazione (generation). Nel primo passo, il modello cerca e recupera informazioni rilevanti da una vasta base di dati o da un insieme di documenti esterni. Questo avviene utilizzando un modello di recupero, come un modello di embedding, per trovare i documenti o i frammenti di testo più pertinenti alla richiesta dell'utente. Nel passo di generazione, spesso basato su architetture di tipo Transformer, queste informazioni vengono utilizzate per costruire una risposta che sia maggiormente rilevante per la domanda. Questa tecnica è particolarmente efficace nella gestione delle conversazioni generative, poiché permette al chatbot di comprendere meglio il contesto della conversazione e generare risposte coerenti e pertinenti. Allo stesso modo, i modelli di linguaggio di grandi dimensioni (LLM) hanno migliorato la capacità dei chatbot di comprendere e generare testi coerenti e informativi, fornendo una base solida per creare conversazioni fluide e naturali.

1.2 Librerie usate

Di seguito si riporti l'elenco delle librerie utilizzate.

1.2.1 Streamlit

Streamlit è un framework open-source in Python che consente agli sviluppatori di creare applicazioni web interattive e personalizzate per il machine learning e l'analisi dei dati con estrema facilità. È progettato per essere semplice da usare, consentendo di trasformare script Python in app web intuitive con poche righe di codice. Streamlit è particolarmente apprezzato per la sua capacità di visualizzare rapidamente i risultati delle analisi e di creare interfacce utente senza la necessità di competenze approfondite nello sviluppo web, proprio per questo motivo è stata scelta questa libreria.

1.2.2 langchain

Questa libreria proviene dall'ecosistema Langchain, che è usata per costruire applicazioni che utilizzano modelli di linguaggio di grandi dimensioni (LLM), in particolare la libreria community è focalizzata sulla collaborazione e il contributo della comunità.

1.2.3 torch

Torch è una libreria scientifica di calcolo in Python che supporta il machine learning ed è stata sviluppata da Meta AI. È ampiamente utilizzata per lo sviluppo di applicazioni di deep learning grazie alla sua efficienza e flessibilità. La libreria fornisce numerosi strumenti per la costruzione, l'allenamento e l'implementazione di reti neurali.

1.2.4 faiss-cpu

FAISS (Facebook AI Similarity Search) è una libreria progettata per facilitare la ricerca di similarità e il clustering su grandi insiemi di vettori densi. La versione faiss-cpu di FAISS è progettata per essere eseguita su CPU, offrendo una soluzione potente per chi non ha accesso a GPU o preferisce utilizzare risorse CPU per la propria elaborazione.

1.3 Sinopsi

L'ossatura del progetto prevede l'implementazioni di due moduli, tra i quali:

- **utils.py**
- **main.py**

In particolare, il programma offre una piattaforma di chatbot che utilizza documenti caricati dagli utenti come base di conoscenza. Gli

utenti caricano documenti tramite l'interfaccia di bot.py, che vengono processati per generare embeddings. Questi embeddings vengono salvati e utilizzati per addestrare un chatbot configurato tramite main_bot.py. Il modulo chatbot_streamlit_combined.py gestisce l'interfaccia utente complessiva e le diverse pagine dell'applicazione, mentre falcon.py fornisce le funzioni di backend per la gestione dei documenti e del modello di linguaggio.

Di seguito verranno illustrate le funzionalità di ciascun modulo, riportando esclusivamente estratti di codice essenziali alla creazione del progetto finale, integrando aspetti teorici e considerazioni sull'utilizzo attuale delle tecnologie coinvolte.

Si noti che l'elaborato è a scopo didattico ed esemplificatico, la base di conoscenza infatti non risulta del tutto sufficiente ed in oltre i limiti tecnologici influenzano i tempi di esecuzione e l'efficenza del sistema. Si è quindi cercato di bypassare il più possibile tali barriere cercando di ottimizzare il codice per lo scopo finale usufruendo infine di librerie che usassero risorse il modo più efficente possibile.

1.3.1 main.py

Funzionalità Principali:

- **Impostazione dello Sfondo:** La funzione set_background() imposta lo sfondo dell'applicazione utilizzando un'immagine specificata tramite stile CSS.

- **Pulizia della Memoria GPU:** La funzione clear_gpu_memory() libera la memoria della GPU svuotando la cache di PyTorch e raccogliendo la spazzatura (garbage collection).

- **Gestione della Cronologia della Chat:** La funzione chat_history() carica la lista delle chat salvate dalla directory chat_history e la restituisce. Se non ci sono chat salvate, ritorna solo l'opzione "Nuova chat".

- **Apertura della Cronologia della Chat da File:** La funzione open_chat_from_file(name_file) carica la cronologia delle chat da un file specificato e la restituisce. Se si tratta di una nuova chat, la cronologia è vuota. Se l'opzione di sovrascrittura è attiva, la cronologia viene letta dal file, altrimenti viene utilizzata la cronologia corrente memorizzata nello stato della sessione di Streamlit.

- **Scrittura della Cronologia della Chat su File:** La funzione write_on_history_file(name_file, question, answer) scrive una domanda e una risposta nel file di cronologia delle chat specificato.

- **Codifica di un PDF in Base64:** La funzione get_base64_of_pdf(pdf_pat codifica un file PDF in una stringa Base64.

- **Link per il Download del PDF:** La funzione pdf_download_link(pdf_pat name_file) crea un link per il download di un PDF codificato in

Base64.

- **Funzione main()** Questa è la funzione principale che viene eseguita all'avvio dell'applicazione. Esegue le seguenti operazioni:

  1. Pulisce la memoria della GPU.
  2. Imposta l'icona e il titolo della pagina Streamlit.
  3. Imposta lo sfondo dell'applicazione.
  4. Inizializza l'encoder per la funzione di embedding dei documenti.
  5. Esegue la funzione document_embedding per caricare e suddividere i documenti di testo.
  6. Esegue la funzione display_chat_modificato per visualizzare e gestire la chat.

- **Funzione document_embedding(encoder):** Questa funzione carica i documenti di testo utilizzando utils.txt_loader(), li suddivide in blocchi e li salva in un database FAISS se la directory del database non esiste. Se la directory esiste, stampa un messaggio.

- **Funzione display_chat_modificato(encoder):** Questa funzione gestisce la visualizzazione e l'interazione della chat:

  1. Carica il database FAISS locale.
  2. Configura le immagini per l'utente e l'assistente.

3. Visualizza il titolo dell'applicazione e la cronologia delle chat nella sidebar.
4. Prepara il modello LLM utilizzando utils.prepare_rag_llm.
5. Gestisce la cronologia della chat e le opzioni di sovrascrittura. Visualizza i messaggi della chat.
6. Gestisce l'input delle domande dell'utente e genera risposte utilizzando utils.generate_answer.
7. Scrive la cronologia delle chat su file e crea link per il download dei documenti PDF associati alle risposte.

1.3.2 utils.py

Il file utils.py è un modulo di utilità progettato per supportare la lettura, suddivisione e codifica di documenti di testo, nonché per preparare un modello linguistico (LLM) per compiti di domanda-risposta basati su conversazioni. Utilizza librerie come langchain, transformers e streamlit per raggiungere questi obiettivi.

Funzionalità Principali:

- **Caricamento dei Documenti di Testo:** La classe TxtFile consente di caricare e leggere file di testo da una directory specificata. Il costruttore della classe gestisce la lettura del file sia con codifica utf-8 che, in caso di errore, con codifica latin1, salvando il contenuto del file come una stringa e memorizzando i metadati

relativi al percorso del file.

- Suddivisione dei Documenti: La funzione txt_loader() carica tutti i file di testo presenti in una directory specificata (RegTXT), li divide in blocchi utilizzando un tokenizer preaddestrato (sentence-transformers/all-MiniLM-L12-v2) e restituisce i documenti suddivisi. Questo processo facilita la gestione di grandi quantità di testo suddividendole in parti più piccole e gestibili.

- **Codifica dei Testi:** La classe Encoder inizializza una funzione di embedding utilizzando un modello specificato. Questa funzione converte il testo in vettori densi, facilitando il processo di recupero delle informazioni e la somiglianza semantica.

- **Preparazione del Modello Linguistico:** La funzione prepare_rag_llm(loaded_db) prepara una catena di recupero conversazionale (Conversational Retrieval Chain) utilizzando un modello linguistico (Ollama) e una memoria conversazionale. Configura una memoria conversazionale per mantenere il contesto delle ultime interazioni e crea una catena che utilizza il modello LLM, il recuperatore del database e la memoria conversazionale per rispondere alle domande.

- **Generazione delle Risposte:** La funzione generate_answer(question) genera una risposta a una domanda data utilizzando la catena di recupero conversazionale. Utilizza lo stato della sessione di

Streamlit per ottenere la conversazione corrente, genera la risposta e le fonti dei documenti utilizzati per formulare la risposta, restituendo la risposta e le relative fonti documentarie.

Chapter 2

Installazione

Per creare un chatbot capace di elaborare informazioni per la camera dei deputati ci si affida ad un progetto con struttura nota chiamato **chatbot**. Il progetto contiene due file python:

- **utils.py**: Fornisce funzionalità backend per l'elaborazione dei documenti, l'embedding e la preparazione del modello.
- **main.py**: Punto di ingresso principale per l'app Streamlit, con gestione della memoria GPU e pagine separate per il chatbot e l'embedding dei documenti.

Per eseguire il progetto bisogna scaricare Anaconda per avere un ambiente di sviluppo e Python come linguaggio, inoltre, installare le librerie usate all'interno dell'ambiente. Per creare un ambiente Anaconda bisogna aprire il client di Anaconda e andare nella sezione *enviroments*, da qui bisogna selezionare la casella "create" e selezionare

il linguaggio che si desidera utilizzare

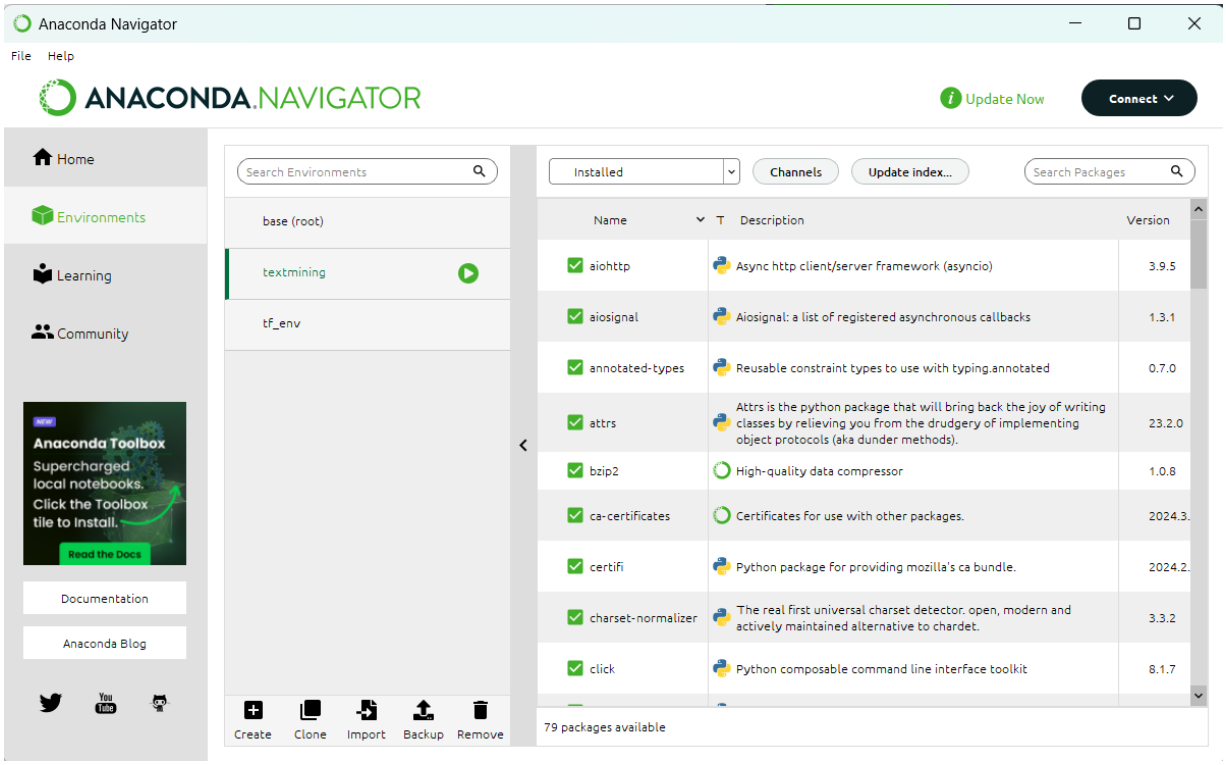

Figure 2.1: Client Anaconda

In alternativa è possibile digitare la seguente riga sul terminale:

Listing 2.1: Creazione ambiente Anaconda

```
$ Conda create —name textmining python= 3.12
```

Una volta creato l'ambiente è possibile scaricare tutte le librerie necessarie stesso da terminale. **N.B.** è possibile creare l'ambiente di anaconda già con tutte le librerie installate semplicemente includendo tutto nella stessa riga di comando:

Listing 2.2: Pacchetti installati

```
$ pip install torch streamlit base64 langchain
```

Scaricare Ollama dal Browser: visita il sito ufficiale di Ollama e scarica il pacchetto di installazione per il tuo sistema operativo (Windows, macOS, Linux). Segui le istruzioni di installazione fornite sul sito per completare l'installazione di Ollama sul tuo computer.

Scaricare il Modello LLaMA 3 dal Terminale: dopo aver installato Ollama, apri il terminale (Prompt dei comandi su Windows, Terminale su macOS/Linux). Utilizza il seguente comando per scaricare il modello LLaMA 3:

Listing 2.3: modello

```
$ ollama pull llama3
```

Per compilare ed eseguire adesso si lanci da shell di Anaconda il comando per spostarsi nel proprio ambiente creato

```
cd <path dove si trova il progetto>
conda activate textminig
```

A questo punto l'ambiente è pronto e si può eseguire, Anaconda permette di avere un ambiente isolato dove sviluppare i propri progetti con le proprie librerie interne.
Si lanci il comando per avviare il chatbot

```
streamlit run chatbot_streamlit_combined.py
```

Adesso da shell di anaconda viene mostrato che il server è in avvio

```
  Local URL: http://localhost:8501
  Network URL: http://100.103.9.109:8501

C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT
C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT\Antonio_Conte.txt
C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT\Reg. persone fisiche Abbate-Desiderio
.txt
C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT
C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT\Antonio_Conte.txt
C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT\Reg. persone fisiche Abbate-Desiderio
.txt
C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT\Reg. persone fisiche Ferrante-Iezzi.t
xt
C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT\Reg. persone fisiche Ferrante-Iezzi.t
xt
C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT\Reg. persone fisiche Italiano-Pascare
lla.txt
C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT\Reg. persone fisiche Italiano-Pascare
lla.txt
C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT\Reg. persone fisiche Pedicini-Zorzi.t
xt
C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT\Reg. persone fisiche Pedicini-Zorzi.t
xt
C:\Users\giugl\OneDrive - Università di Napoli Federico II\TM\chatbot_bruno\RegTXT\Registro cat. 117, imprese, gruppi di
 imprese e aziende.txt
```

Figure 2.2: Load

Da browser si aprirà la schermata per usare il chatbot.

Figure 2.3: Interface

A questo punto i documenti sono stati inclusi correttamente e non resta altro che fare una domanda al chatbot e attendere una risposta.

Chapter 3

Codice

Di seguito è riportato il codice e il significato di ogni funzione.

3.1 utils.py

Nel presente file sono state implementate le funzioni di utilità a supporto del main.

3.1.1 class TxtFile

La classe definita viene inizializzata con un costruttore contenente il path dove sono contenuti i file txt, i metodi specificati sono resi utili per l'apertura degli stessi con codifica utf-8, in oltre viene rempito il contenuto dei file aggiungendoli poco alla volta dentro all'oggetto dichiarato inizialmente. In fine vengono specificati i metadati.

3.1.2 function txt_loader

Le seguente funzione carica tutti i file di testo da una directory specificata, crea una lista di oggetti TxtFile per ciascun file, e quindi utilizza un text splitter per suddividere il contenuto dei file in documenti più piccoli.

Crea inizialmente il path dal quale prelevare il database e all'interno di esso preleva i file .txt per poi fare l'append all'interno della lista pages[], infine specifica l'architettura di tokenizzazione, con i seguenti chunk size e overlap, andando a cancellare gli spazi bianchi. Restituisce poi i documenti.

3.1.3 class Encoder

La classe Encoder è progettata per creare un oggetto che possa generare embeddings utilizzando un modello specifico di Hugging Face. Questa classe utilizza la classe HuggingFaceEmbeddings, che fa parte di una libreria di embeddings di Hugging Face.

Il modello usato da questo Encoder è cuda, che valuta la potenza della GPU al fine di non sovraccaricare il processore.

3.1.4 function prepare_rag_llm

Il metodo è progettato per preparare un chatbot basato su un modello di linguaggio (LLM) utilizzando una catena di recupero conversazionale.

In particolare il modello è llama3 e ha una memoria relative alle ultime due conversazioni con l'utente, in oltre è specificato come il retriver della risposta si basi sull'ottimizzazione delle migliori tre.

3.1.5 function generate_answer

La seguente funzione è progettata per interagire con un sistema di chatbot precedentemente configurato (tramite prepare_rag_llm) per generare una risposta a una domanda fornita. La funzione utilizza uno stato di sessione (st.session_state) per mantenere il contesto della conversazione e recupera sia la risposta che le fonti dei documenti utilizzati per generare quella risposta.
Si ricordi che è utile conservare i documenti poichè linkati all'atto di risposta.

3.2 main.py

Di seguito sono implementate le funzioni che permettono l'esecuzione del chatbot

3.2.1 Function 'chat_history'

La funzione chat_history si occupa di fornire un elenco di tutte le chat disponibili. Essa identifica la directory chat_history nel percorso corrente del file e enumera i file .txt presenti al suo interno, aggiungendo

"Nuova chat" come opzione predefinita. Questo elenco di chat è essenziale per permettere agli utenti di scegliere quale chat visualizzare o modificare.

3.2.2 Function open_chat_from_file

La funzione open_chat_from_file è responsabile del caricamento della cronologia di una chat da un file specifico. Accetta il parametro name_file, che rappresenta il nome del file della chat da cui caricare la cronologia. Se st.session_state.overwriting è vero e name_file non è "Nuova chat", la funzione apre il file corrispondente nella directory chat_history, legge i messaggi registrati (come "user" seguito dal contenuto del messaggio e "assistant" seguito dalla risposta), e li organizza in una lista di dizionari. Se st.session_state.overwriting è falso, la funzione carica la cronologia della chat direttamente dalla sessione corrente, utilizzando st.session_state.history. Questa funzione è fondamentale per recuperare e mantenere il contesto delle conversazioni passate.

3.2.3 Function write_on_history_file

La funzione write_on_history_file consente di aggiungere nuovi messaggi alla cronologia di una chat esistente. Prende in input il nome del file della chat, la domanda posta dall'utente e la risposta fornita dall'assistente. Utilizzando queste informazioni, apre il file corrispon-

dente nella modalità append ('a'), scrive il ruolo del mittente ("user" per la domanda e "assistant" per la risposta) seguito dal contenuto del messaggio, e chiude il file. Questo processo garantisce che ogni interazione utente-assistente venga registrata correttamente nel file di cronologia della chat.

3.2.4 Function get_base64_of_pdf

Questa funzione converte un file PDF in una rappresentazione in formato Base64. Prende in input il percorso del file PDF, lo apre in modalità lettura binaria ("rb"), legge il contenuto del file, lo codifica in Base64 e restituisce il risultato sotto forma di stringa decodificata UTF-8. È utile quando si desidera incorporare un PDF all'interno di un'applicazione web o convertirlo in un formato che può essere facilmente condiviso tramite dati URL.

3.2.5 Function pdf_download_link

Questa funzione genera un link di download per un file PDF specificato. Utilizza la funzione get_base64_of_pdf per ottenere la rappresentazione Base64 del PDF dal percorso specificato. Successivamente, utilizza st.link_button per creare un pulsante nel framework Streamlit che, quando cliccato, consente all'utente di scaricare il PDF specificato. Il pulsante è configurato per mostrare il nome del file fornito e include i dati del PDF in formato Base64 nel link di download.

3.2.6 Function document_embedding

Questa funzione gestisce l'incorporazione dei documenti per la chat. Utilizza un loader per caricare i documenti da un'origine specifica. Se la directory di destinazione per il salvataggio dei vettori non esiste, viene creato un nuovo database FAISS utilizzando l'encoder e la strategia di distanza coseno per gestire l'incorporazione e l'indicizzazione dei documenti. Se la directory esiste già, la funzione stampa un messaggio di conferma.

3.2.7 Function display_chat_modificato

La funzione display_chat_modificato gestisce l'interfaccia utente della chat all'interno dell'applicazione Streamlit. È progettata per mostrare la cronologia delle conversazioni tra utente e assistente, integrando un modello di linguaggio e gestendo l'interazione dell'utente con l'assistente virtuale.

Utilizza FAISS.load_local per caricare un database di documenti precedentemente salvato. Questo database è stato creato utilizzando l'encoder specificato, che è stato utilizzato per incorporare e indicizzare i documenti testuali. Il parametro allow_dangerous_deserialization=True è utilizzato per consentire il caricamento sicuro del database dal disco.

Carica due immagini per l'utente (user_image) e l'assistente (assistant_image) dalla directory specificata. Queste immagini vengono utilizzate per rappresentare visivamente i messaggi inviati dall'utente

e dall'assistente all'interno dell'interfaccia di chat.

Imposta il titolo dell'applicazione Streamlit come "FORZA NAPOLI SEMPRE (FNS)". Questo titolo appare nella parte superiore della pagina, fornendo un'identità visiva all'applicazione.

Recupera l'elenco delle chat storiche utilizzando la funzione chat_history. Questa lista è visualizzata nella barra laterale dell'applicazione, permettendo agli utenti di selezionare una chat precedente da visualizzare.

Controlla se l'oggetto di sessione conversation è già stato inizializzato. Se non è presente, chiama utils.prepare_rag_llm per preparare un modello di linguaggio conversazionale. Questo modello viene utilizzato per gestire le interazioni utente-assistente durante la sessione di chat.

Inizializza le variabili di sessione overwriting e not_saved_chat se non sono già presenti. Queste variabili tengono traccia dello stato della chat corrente, in particolare se la chat è in corso di sovrascrittura o se è stata salvata.

Utilizza un ciclo per iterare attraverso la cronologia della chat memorizzata nella sessione (st.session_state.history). Per ogni messaggio nella cronologia, determina se il messaggio è stato inviato dall'utente o dall'assistente e visualizza il messaggio corrispondente utilizzando st.chat_message, impostando l'avatar appropriato (immagine dell'utente o dell'assistente).

Fornisce all'utente la possibilità di fare domande attraverso un in-

put di chat (st.chat_input). Quando un utente invia una domanda, questa funzione aggiunge la domanda alla cronologia della chat e utilizza il modello di linguaggio preparato per generare e visualizzare una risposta appropriata. Le risposte dell'assistente vengono visualizzate utilizzando st.chat_message e aggiunte alla cronologia della chat.

Per ogni risposta fornita dall'assistente, la funzione genera anche dei link ipertestuali per i documenti sorgente associati. Questi documenti sono caricati dalla directory "TextMiningPdf" e resi disponibili per il download tramite pdf_download_link.

Utilizza st.expander per espandere e mostrare ulteriori dettagli sulla cronologia della chat e sulle informazioni sorgente associate. Questa sezione fornisce una vista più dettagliata delle interazioni tra l'utente e l'assistente, inclusi i documenti consultati durante la chat.