
Progetto Text Mining

ChatBot

Documentazione
I3Namecciani

Corso di Laurea Magistrale in Ingegneria Informatica
Università degli Studi di Napoli Federico II



UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

**Corso di Laurea Magistrale in Ingegneria
Informatica**

Università degli Studi di Napoli Federico II
<https://www.unina.it>

Titolo:

ChatBot per la Camera dei Deputati Italiana

Tema:

ChatBot - RAG - IA Generativa - Text Minig

Estratto:

In questa documentazione viene presentato lo sviluppo di un ChatBot RAG mediante API, applicato a documentazione riguardante la Camera dei Deputati Italiana.

Anno:

Sessione Estiva 2024

Gruppo:

I3Namecciani

Partecipanti:

Michele Savella
Ciro Scogliamiglio
Antonio Sirignano

Numero pagine: 11

Data:

29 maggio 2024

Il contenuto di questo progetto è disponibile gratuitamente.

Indice

1	Librerie e servizi utilizzate	1
2	Componenti dell'applicazione	2
2.1	Chunking dei documenti	2
2.2	Embedding	3
2.3	Database vettoriale e retriever	3
2.4	LLMs	4
3	Interfaccia grafica	5
4	Test effettuati	7
5	Come far funzionare il chatbot	9
	Bibliografia	11

Capitolo 1

Librerie e servizi utilizzate

Di seguito riportiamo tutti i package ed i servizi utilizzati nel progetto:

- **Langchain:** è un framework di orchestrazione open source per lo sviluppo di applicazioni che utilizzano modelli linguistici di grandi dimensioni. Langchain funge da interfaccia per quasi tutti gli LLM, fornendo un ambiente di sviluppo centralizzato per creare applicazione LLM e integrarle con fonti di dati esterni e workflow di software;
- **Streamlit:** è una libreria open-source di python che facilita la creazione e lo sviluppo di applicazioni web personalizzate con interfacce interattive;
- **langchain-google-genai:** è libreria di integrazione per Langchain per supportare Gemini di Google;
- **Gemini:** è un LLM di cui Google mette a disposizione delle API gratuite (solo in paesi fuori dall'Unione Europea);
- **HuggingFace:** è una piattaforma che offre una libreria vasta di modelli pre-addestrati, una comunità di sviluppatori e ricercatori, una serie di tutorial e documentazione, e una piattaforma cloud che permette di allenare, condividere e usare i modelli in modo semplice e veloce;
- **faiss-cpu:** è una libreria di supporto per permettere il corretto funzionamento del modulo FAISS di Langchain
- **sentence-transformers (SBERT):** è una libreria per utilizzare modelli di embedding allo stato dell'arte
- **python-docx:** è una libreria per leggere, creare, modificare e aggiornare file Word (.docx).

Capitolo 2

Componenti dell'applicazione

Il nostro chatbot è stato sviluppato essenzialmente in due fasi:

- **Fase 1:** in questa fase abbiamo effettuato il chunking dei documenti forniti come base di conoscenza e dopo l'operazione di embedding abbiamo salvato i vettori risultati in un database vettoriale, che nel nostro caso è FAISS;
- **Fase 2:** questa è la fase principale del chatbot, l'utente effettua una domanda riguardante la base di conoscenza, se vi è stata già una precedente conversazione la domanda viene modificata per includere le informazioni della conversazione utilizzando un primo LLM. Successivamente la domanda viene passata al retriever che recupera dal database vettoriale i chunk di documentazioni più "simili" alla domanda (dell'utente o modificata dall'LLM se non è la prima domanda). Successivamente un secondo LLM risponde alla domanda utilizzando come contesto i chunk recuperati dal retriever, la risposta viene fornita in uscita insieme ai documenti utilizzati.

In figura 2.1 vi è riportato uno schema dell'architettura complessiva del chatbot. Vediamo ora i componenti dello schema nel dettaglio.

2.1 Chunking dei documenti

Il chunking è un'operazione che prevede la suddivisione di un testo in diverse parti, dette chunk, utilizzando un criterio di split.

I documenti forniti come base di conoscenza sono dei file Word (.docx), quindi abbiamo tentato due vie:

- Come primo approccio abbiamo convertito tutti i file in formato PDF ed abbiamo utilizzato la classe RecursiveCharacterTextSplitter di Langchain per ottenere i chunk;
- Con il secondo approccio abbiamo voluto sfruttare la struttura peculiare dei file word, quindi abbiamo effettuato un chunking "manuale" per dividere i documenti in paragrafi utilizzando la libreria python-docx.

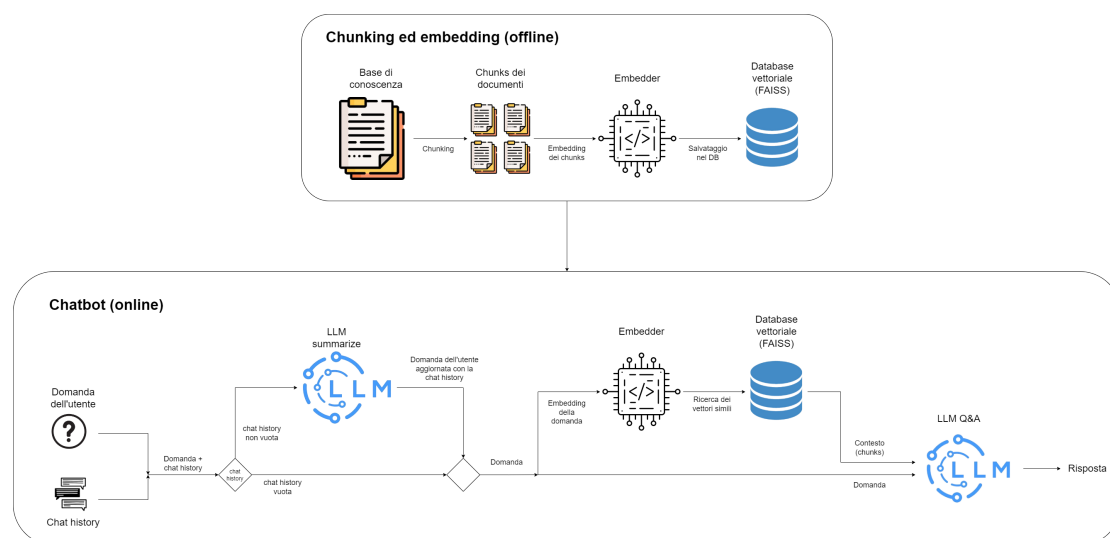


Figura 2.1: Schema realizzativo di Lucio

Il secondo approccio si è rivelato il migliore in quanto divideva perfettamente i paragrafi e permetteva quindi di avere tutte le informazioni in un unico chunk.

2.2 Embedding

L'embedding prevede di trasformare una stringa in un vettore in uno spazio n-dimensionale, questa operazione è necessaria per permettere di calcolare in modo semplice la somiglianza tra documenti diversi.

Come modello di embeddings abbiamo utilizzato quello fornito da Google, in particolare "models/embedding-001" che embedda il testo in vettori di 768 elementi.

2.3 Database vettoriale e retriever

Il database vettoriale è incaricato di memorizzare gli embeddings calcolati, mentre il retriever ci permette di recuperare gli elementi nel database simili ad stringa passata in ingresso (di cui prima viene calcolato l'embedd). Un parametro importante del retriever è il numero di documenti che ritorna la ricerca.

In particolare abbiamo utilizzato FAISS come database vettoriale e quindi il retriever è quello fornito da Langchain per FAISS. Il numero di documenti ritornati dal retriever è variabile ed è modificabile dall'utente.

2.4 LLMs

Il fulcro dell'intero chatbot sono gli LLM. Gli LLM sono modelli linguistici di grandi dimensioni in grado di generare testo. Nel nostro chatbot ne sono utilizzati 2, per scopi molto diversi:

- Il primo modello (LLM1 in figura 2.1) è incaricato di contestualizzare la domanda attuale dell'utente utilizzando la conversazione passata tra utente e chatbot (aggiungendo contesto e dettagli alla domanda attuale). Nel caso in cui sia la prima domanda dell'utente, questo modello non viene utilizzato
- Il secondo modello (LLM2 in figura 2.1) è quello che effettivamente si occupa di rispondere alla domanda dell'utente. Prende in ingresso la domanda (contestualizzata o meno da LLM1) ed il contesto ricavato dal processo di retrieving e risponde utilizzando tutte le informazioni che ha a disposizione.

In particolare i due modelli svolgono compiti molto diversi, quindi anche i parametri usati per impostarli sono diversi. Ad esempio la temperatura di LLM1 è più bassa di LLM2, in quanto il primo deve solo riassumere delle informazioni e non elaborare una risposta in modo "creativo".

Capitolo 3

Interfaccia grafica

L'interfaccia grafica di Lucio è molto semplice e funzionale.

Al centro vi è la chat che l'utente ha con Lucio. In basso vi è una casella di testo nel quale

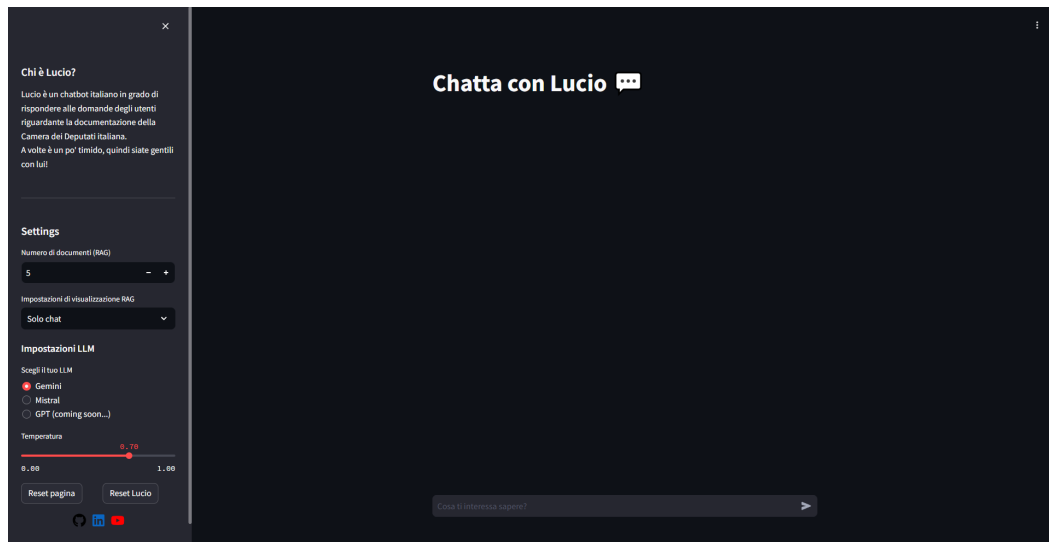


Figura 3.1: Interfaccia

inserire la domanda.

La toolbar sulla sinistra da una prima descrizione di cosa fa il chatbot e su quale documentazione è stato addestrato.

Al di sotto di essa abbiamo le impostazioni: vi è la possibilità di scegliere quale modello utilizzare, tra Gemini (VPN in USA necessaria), Mistral e GPT (chiave a pagamento); vi è la possibilità di scegliere il numero di chunks su cui fare il retriever e scegliere con quale modalità visualizzare la chat:

- **Solo Chat** permette la visualizzazione della semplice chat con Lucio

- **Visualizzazione documenti del retriever** permette la visualizzazione dei documenti dai quali viene effettuato il retriever
- **Visualizzazione del contenuto dei documenti** permette la visualizzazione del contenuto dei documenti recuperati

Inoltre tramite lo slider è possibile settare la temperatura del modello scelto. Infine vi sono due tasti:

- **Reset Pagina** riavvia completamente il sistema;
- **Reset Lucio** resetta semplicemente la chat history in modo da poter iniziare una nuova conversazione.

Capitolo 4

Test effettuati

Di seguito riportiamo alcuni test che abbiamo fatto con i documenti:

- C'è qualcuno che si chiama Bloise? Se sì, puoi darmi informazioni su di lui?
 - Sì, c'è una persona di nome Bloise nella documentazione. Si chiama Bloise Generoso ed è un avvocato che si occupa di assistenza alle imprese in materia tributaria e penale, compliance aziendale e corretta gestione delle imprese.
- Dov'è domiciliato Luca Fontana?
 - Luca Fontana è domiciliato professionalmente in Palazzo Valadier - Piazza Del Popolo n°18, 00187 Roma (RM).
- Dove si trova Associazione Commercialisti Cattolici?
 - Piazza Duomo n°16, 20122 Milano (MI)
- Che attività svolge COMMUNITY SRL?
 - COMMUNITY SRL svolge rappresentanza di interessi per Lidl Italia, leader della grande distribuzione organizzata, in Italia dal 1992. L'attività di rappresentanza di interessi riguarda gli ambiti della competenza della commissione attività produttive e della commissione ambiente, con riferimento ai temi dell'economia circolare e della grande distribuzione.
- Chi è Antonio Sirignano?
 - Non posso rispondere alla tua domanda, poiché il testo fornito non contiene informazioni su Antonio Sirignano.

Per quanto riguarda le conversazioni:

- Che attività svolge Ital Communications S.r.l. ?

- Ital Communications S.r.l. svolge rappresentanza di interessi per Auditel, la società che monitora e pubblica gli ascolti televisivi.
- Dove si trova?
 - Via Antonio Gramsci n°36, 00197 Roma (RM)

Capitolo 5

Come far funzionare il chatbot

L'archivio consegnato insieme a questa documentazione contiene i seguenti file, contenuto nella cartella *embedding*:

Il codice con cui è stato effettuato l'embedding è, invece, contenuto nella cartella *embedding* che contiene:

- Cartella *docx*: cartella contenente i documenti della base di conoscenza:
 - Reg. persone fisiche Abbate-Desiderio
 - Reg. persone fisiche Ferrante-Iezzi
 - Reg. persone fisiche Italiano-Pascarella
 - Reg. persone fisiche Pedicini-Zorzi
 - Registro cat. 117, imprese, gruppi di imprese e aziende
 - Registro cat. Altro
 - Registro cat. associazioni di categoria o di tutela di interessi diffusi
 - Registro cat. associazioni di consumatori riconosciute
 - Registro cat. associazioni professionali
 - Registro cat. organizzazioni datoriali e sindacali
 - Registro cat. organizzazioni non governative
 - Registro cat. soggetti specializzati
- *embedding.py*: contiene il codice da utilizzare, in locale o su colab effettuare embedding e salvare i risultati in un database vettoriale

L'intero codice del chatbot è contenuto nella cartella *chatbot* che contiene:

- Cartella *faiss_db*: contiene i file del database vettoriale (FAISS)
- *dashboard_colab.ipynb*: contiene il codice da utilizzare per un'esecuzione su colab che sfrutta npx per creare un tunnel tra la macchina colab ed il nostro computer.

- *dashboard_locale.py*: contiene il codice da utilizzare per un'esecuzione in locale (necessaria VPN USA per Gemini)
- *requirements.txt*: file contenente tutti i package necessari per eseguire il codice in locale

I passi da effettuare per eseguire su Colab la dashboard e quindi il chatbot sono:

- Inserire nei file *dashboard_colab.ipynb* le key di Gemini, HuggingFace ed OpenAI
- Caricare il notebook su Colab
- Caricare l'archivio (.zip) contenente il database vettoriale in cui sono memorizzati gli embedding dei documenti della base di conoscenza
- Eseguire tutte le celle del notebook
- Copiare l'indirizzo IP riportato nella prima riga di output dell'ultima cella e copiarlo nel campo password una volta aperto il link presente nell'output dell'ultima cella del notebook.

Mentre i passi da effettuare per eseguire in locale la dashboard e quindi il chatbot sono:

- Installare tutti i requisiti con `pip install -r requirements.txt`
- Inserire nei file *dashboard_locale.py* le key di Gemini, HuggingFace ed OpenAI
- Attivare VPN negli Stati Uniti se si vuole usare Gemini (a breve Gemini sarà disponibile in Italia)
- Eseguire il file *dashboard_locale.py* con il comando `streamlit run dashboard_locale.py`

Bibliografia

- [1] *Streamlit • a Faster Way to Build and Share Data Apps*. URL: <https://streamlit.io/>.
- [2] *LangChain Docs*. URL: <https://python.langchain.com/v0.2/docs/introduction/>.
- [3] *Faiss Docs*. URL: <https://faiss.ai>.
- [4] *python-docx*. URL: <https://python-docx.readthedocs.io/en/latest/>.
- [5] *Build with the Gemini API*. URL: <https://ai.google.dev/gemini-api>.