



Università degli Studi di Napoli Federico II  
Scuola Politecnica e delle Scienze di Base

---

**CORSO DI LAUREA MAGISTRALE IN  
INGEGNERIA INFORMATICA**

**PROGETTO DI TEXT MINING - 2023/24**

**Gruppo *W\_Team***

**Valerio Cera M63001700**

**Tommaso Di Lillo M63001642**

**Riziero Graziani M63001596**

**Lucia Porcelli M63001701**

**PROF.SSA: Flora Amato**

## Sommario

1.	Introduzione .....	3
2.	Principio di funzionamento.....	4
3.	Spiegazione codice .....	6
4.	Blocchi funzionali.....	7
5.	Diagramma di flusso .....	10
6.	Implementazione.....	11
7.	Test e risultati .....	14
7.1	Test.....	14
7.2	Risultati .....	16

# 1. Introduzione

Un **chatbot** è un software progettato per simulare conversazioni con utenti umani, cercando di utilizzare un linguaggio naturale. Ormai fanno parte della vita quotidiana, basti pensare ad un qualsiasi Servizi Clienti, il quale è capace di rispondere alle domande più frequenti.

Il funzionamento del chatbot è paragonabile ad un **ciclo if-else**, in quanto sono presenti alcune regole predefinite e una corrispondenza della parole chiavi.

I primi chatbot erano abbastanza statici, in quanto si basavano su un **database di risposte** predefinite. Sostanzialmente un'utente inseriva una query, il chatbot cercava le regole associate alla domanda e forniva la risposta associata. Quindi oltre alle risposte preimpostate non si era capaci di andare.

Oggigiorno esistono chatbot basati su modelli di linguaggio di grandi dimensioni **LLM** (Large Language Model), fornendo iterazioni più avanzate e dinamicità. Esistono di due tipi:

1. Chatbot basati su **RAG**: un chatbot RAG (Retrieval-Augmented Generation) combina metodi basati su recupero e intelligenza artificiale generativa per fornire risposte accurate e contestualmente rilevanti. Utilizza un database per recuperare informazioni pertinenti e successivamente un modello linguistico per generare risposte coerenti. Questo lo rende efficace per compiti che richiedono informazioni aggiornate o specifiche.
2. Chatbot basati su LLM senza RAG: tali chatbot utilizzano OpenAi o Llama e sono basati su un numero elevato di dati testuali. Presentando così un vantaggio maggiore in quanto possono gestire una vasta gamma di domande senza utilizzare regole predefinite, possono aggiornarsi e adattarsi nel tempo e infine generano risposte più umane (basti pensare a **ChatGPT**). La loro pecca è l'elevata potenza di calcolo e la generazione di risposte inappropriate, anche se in minor percentuale. Alcuni modelli sono open-source mentre altri richiedono specifiche **API** (Application Programming Interface) fornite dall'organizzazione che li ha sviluppati, le quali possono essere a pagamento.

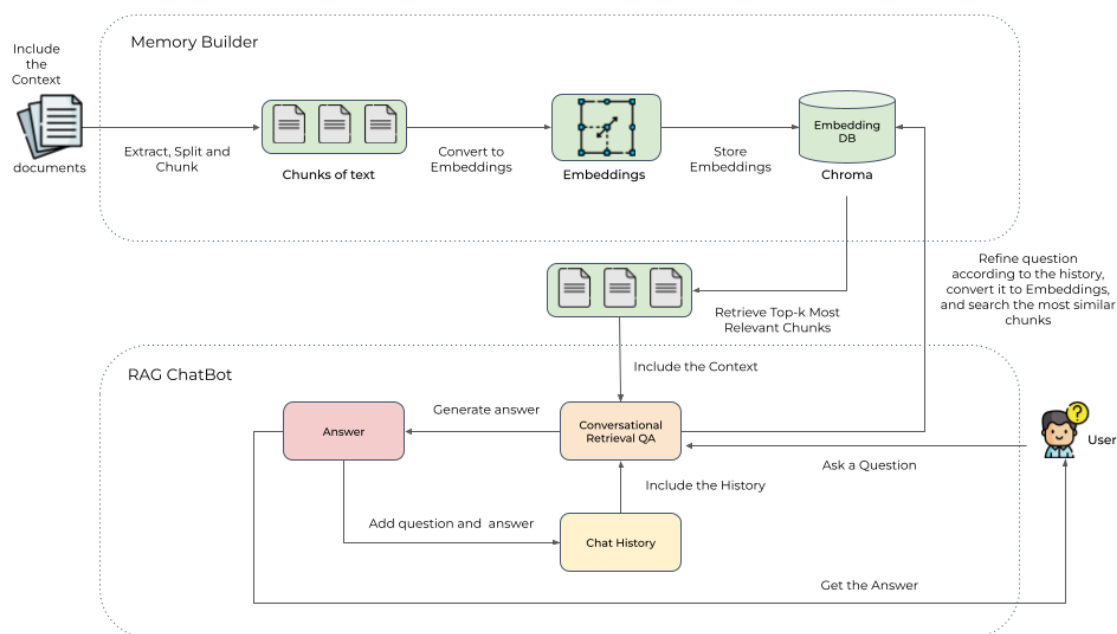
Il chatbot portato a termine in questo elaborato è con RAG e verrà illustrato nel secondo capitolo. Il progetto è stato sviluppato su due fronti:

- Il primo (completo) è di usare modelli di HuggingFace per poter sfruttare delle LLM open source in maniera diretta e semplificata;
- Il secondo approccio (incompleto) è stato quello di presentare **OLLAMA** come possibile alternativa per il retrieve del LLM.

## 2. Principio di funzionamento

Il Chatbot RAG prende come input una raccolta di file, che nel nostro caso sono dei pdf, e una volta posta una domanda, si fornisce una risposta in base al contesto fornito dai file.

Un esempio di architettura è la seguente:



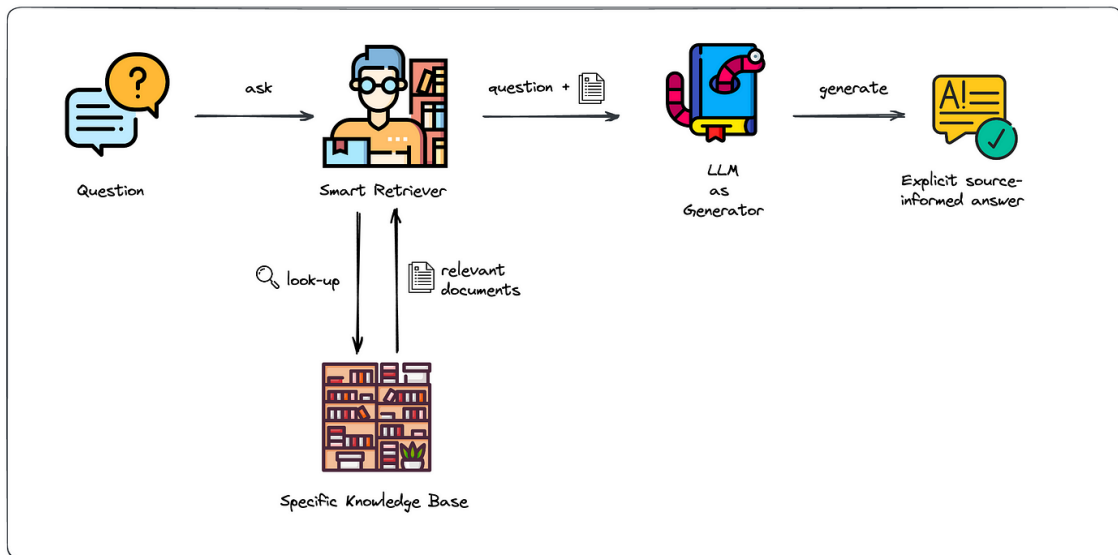
Nella prima fase c'è l'elaborazione dei documenti di input in frammenti piccoli chiamati **'chunk'** e la conversione di quest'ultimi in **vettori di embedding** in modo da acquisire il significato semantico. Per la memorizzazione, ricerca e analisi di questi vettori vengono usati database particolari che vedremo nel capitolo successivo. Resta comunque l'obiettivo di poter avere un recupero che sia il più rapido e preciso possibile delle informazioni.

A questo punto i vettori vengono messi in un database apposito in modo tale da avere un recupero rapido ed efficiente delle informazioni.

La seconda fase inizia quando l'utente fa una domanda. Quest'ultima viene elaborata e convertita in un vettore embedding (con lo stesso modello usato nella prima fase), in modo tale da iniziare la ricerca di chunks simili all'interno del database.

Recuperati questi chunks, il passo successivo è quello di passare il tutto al modello LLM per generare la risposta finale. È importante in che modo viene effettuata la domanda al modello e come questi frammenti vengono passati al LLM, il tutto dipende fortemente dal prompt usato per allenare il modello.

In alcuni casi la domanda può essere raffinata in base alle conversazioni precedenti. Una volta trovata la risposta quest'ultima viene sia mandata all'utente sia salvata all'interno dello storico.



### 3. Spiegazione codice

L'ambiente di programmazione utilizzato è stato **Google Colab**. Il codice, presente su Google Colab, è stato strutturato in 10 macroregioni

**1° e 2°** - Installazione dei pacchetti e delle librerie necessari. Alcuni pacchetti usati sono:

- **Gradio**: pacchetto per la creazione di interfacce web facili;
- **Pydf**: libreria per lavorare con file PDF;
- **Transformers**: pacchetto di HuggingFace fornisce modelli pre-addestrati;
- Altre librerie di HuggingFace per il database, pipeline, e così via.

**3°** - Importazione del modello utilizzato. È stato scelto un modello pre-addestrato e Multilingue, basato su Llama-3-8b-instruct e creato/mantenuto completamente in Italia.

```
model_name = "swap-uniba/LLaMAntino-3-ANITA-8B-Inst-DPO-ITA"
```

**4°** - Importazione del modello quantizzato per **ridurre** le risorse consumate. Vengono utilizzate le funzioni **"load\_quantized\_model"** e **"AutoModelForCausalLM"**, che rispettivamente cambiano la quantizzazione in **4 bit** e caricano tale quantizzazione al modello scelto al passo 3. In questa fase viene caricato anche il **tokenizer**, preso dal modello scelto.

**5°** - creazione di una chiave su Google Colab da associare alla **chiave API** di Hugging Face, in modo tale da dare accesso a servizi e risorse fornite da HuggingFace. In questa fase avviene anche la connessione al proprio Google Drive.

**6°** - Importazione dei documenti presenti nella cartella **'document\_colab'** di google drive. Come si può notare nella cella successiva i documenti appena caricati vengono spostati in un'altra cartella. Questo perché si è considerato che i documenti possono essere strutturati in sottocartelle.

**7°** - Suddivisione del testo in piccoli chunk. Ciò è utile per il recupero delle informazioni quando vengono fatte le domande. Si noti la funzione **'RecursiveCharacterTextSplitter'**.

I parametri passati alla funzione indicano rispettivamente la **lunghezza** dei chunk (1500) e il numero di caratteri che si **sovrappongono** tra chunk consecutivi (200).

**8°** - Viene preparato un modello di **embedding** multilingue.

**9°** - Questo passo è il più importante in quanto viene creato un sistema di **Recupero-Generazione** capace di rispondere alle domande degli utenti. Questa è il passo visto nell'indice 2 dove i chunk dei documenti vengono indicizzati in un database vettoriale (Chroma).

**10°** - Infine c'è la creazione della conversazione. In particolare è stato utilizzato Gradio per modificare l'interfaccia grafica.

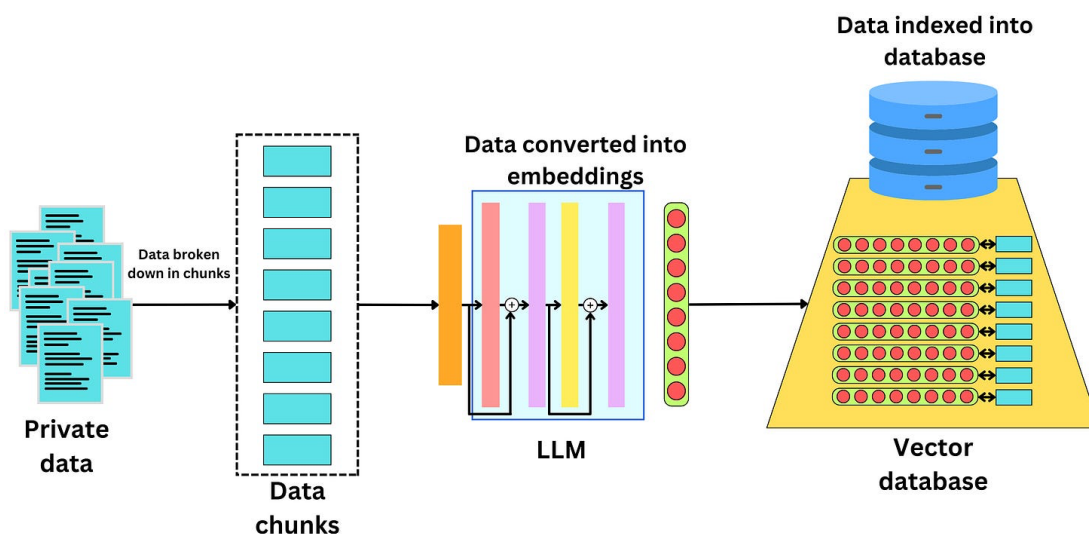
## 4. Blocchi funzionali

Il chatbot è composto da diversi blocchi funzionali, ognuno svolge uno specifico compito. Nel seguente capitolo andremo ad esplorare il funzionamento di ciascuno di essi.

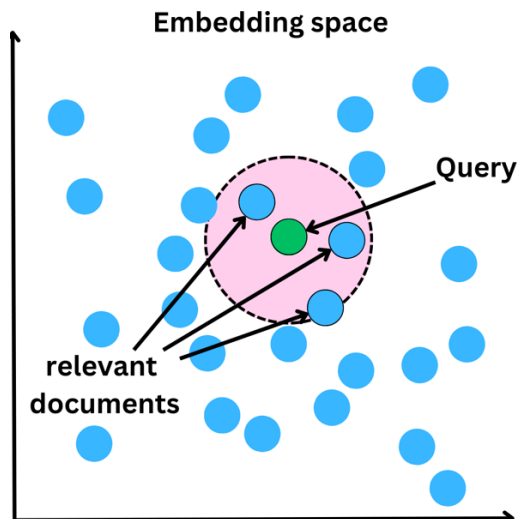
1. **Database Vettoriale:** nella fase di creazione del retriever creiamo un database vettoriale, che viene chiamato vectordb e viene creato grazie a ChromaDB. Ma perché è necessario usare questo tipo di database? Come sappiamo, generalmente i diversi pdf sono formati da una serie di dati testuali, che però non possono essere elaborati direttamente dagli LLM. Per rendere il tutto utilizzabile per i nostri modelli siamo obbligati a generare numeri da questi testi e catturarne il significato semantico, questo è possibile grazie all'embedding delle parole. Grazie a questo embedding generiamo un vettore di embedding a partire dai testi. Generalmente, questi vettori sono di alta dimensionalità e complessi da gestire con un normale database relazionale. È possibile memorizzarli in un database relazionale, ma quando si esegue una ricerca semantica per individuare un contesto simile, viene utilizzata una ricerca lineare che risulta essere lenta e onerosa dal punto di vista computazionale. E' qui che intervengono i database vettoriali.

A differenza dei database relazionali tradizionali, i database vettoriali sono ottimizzati per eseguire operazioni di confronto e ricerca tra vettori ad alta velocità. Utilizzano tecniche avanzate come l'indexing vettoriale, che permette di effettuare ricerche più efficienti rispetto alla ricerca lineare, riducendo il tempo e le risorse computazionali necessarie. In particolare essi usano l'Hashing Sensibile alla Località (LSH) per raggruppare dati simili in base al loro significato semantico nello stesso bucket e poi vengono indicizzati per un rapido recupero.

Per il nostro progetto abbiamo usato ChromaDB, sono tuttavia disponibili diversi strumenti per poter eseguire le stesse operazioni, ad esempio è possibile usare Faiss.

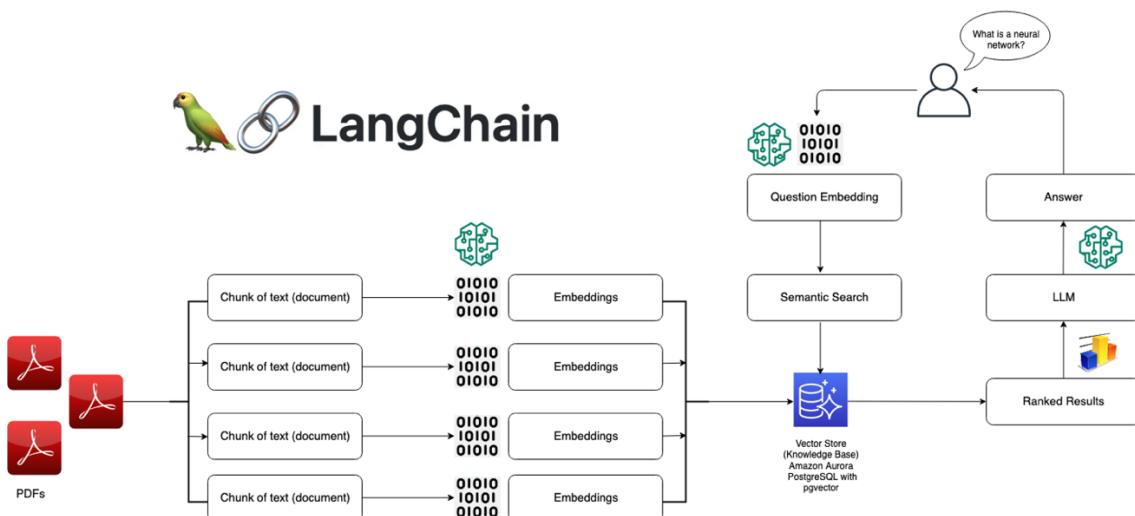


Ed ecco come i database vettoriali riescono a rendere semplice la ricerca per trovare documenti di testo semanticamente simili:



2. **Langchain:** è un framework progettato per aiutare gli sviluppatori a creare applicazioni alimentate da modelli di linguaggio su larga scala, come quelli di OpenAI, in modo più semplice ed efficiente. Questo framework offre strumenti e componenti modulari che facilitano l'integrazione dei modelli di linguaggio nelle applicazioni, permettendo di gestire varie funzionalità come l'elaborazione del linguaggio naturale, la generazione di testo, l'analisi del sentiment, e molto altro. Una possibile alternativa a LangChain è ad esempio LlamaIndex. Nel nostro caso LangChain è il cuore del progetto, infatti esso ci permette di caricare i documenti, ci permette di elaborarli e di avere la conversazione con la LLM.

Tra le componenti principali abbiamo: HuggingFacePipeline per poter integrare la LLM, HuggingFaceEmbeddings per generare gli embedding dei testi usando modelli presi da HuggingFace, ChromaDB come database vettoriale ed infine il ConversationalRetrievalChain che permette di avere una catena per poter combinare il retrieval con modelli di LLM per poter consentire le conversazioni.





3. **Embedding:** per effettuare l'embedding, ovvero per generare il vettore di embedding, abbiamo usato un modello pre-addestrato preso direttamente da HuggingFace. Il modello è paraphrase-multilingual-MiniLM-L12-v2, basato su 118 milioni di parametri e da documentazione:

***'It maps sentences & paragraphs to a 384-dimensional dense vector space and can be used for tasks like clustering or semantic search.'***

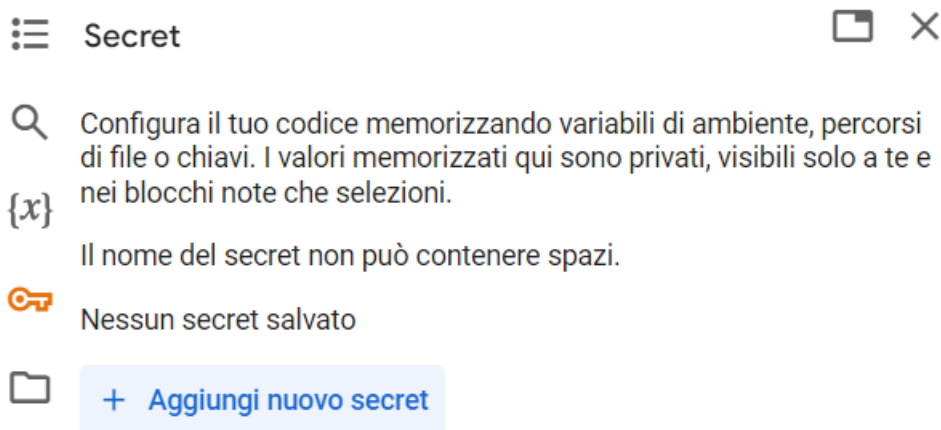
4. **Gradio:** è una libreria open-source di Python che consente di creare rapidamente e facilmente interfacce utente web per i modelli di machine learning. È particolarmente utile per sviluppatori e ricercatori che desiderano rendere i loro modelli accessibili attraverso un'interfaccia intuitiva senza dover scrivere molto codice frontend. È stata infatti creata un'interfaccia attraverso Gradio grazie alla quale possiamo interagire col nostro chatbot. Tra i vantaggi di Gradio abbiamo: facilità d'uso, condivisione online, integrazione con strumenti di Machine Learning, alta personalizzazione della UI.

Come alternativa era possibile scegliere, ad esempio, Streamlit.

## 5. Diagramma di flusso

Affinché possa essere visualizzato correttamente il chatbot, eseguire i seguenti passaggi:

1. Creare nel proprio account personale **Google Drive** una cartella chiamata: **documents\_colab**;
2. Importare i documenti PDF nella cartella creata al passo 1;
3. Aprire il codice presente sul sito **GitHub** ;
4. Prima di eseguire il runtime di tutte le celle, si controlli che la GPU selezionata sia **T4**. Qualora non fosse così eseguire il passo 'a' successivo, altrimenti si passi al 5;
  - a. Si vada su '**runtime**' e si clicchi 'cambia tipo di runtime'. A questo punto selezionare: T4 GPU;
5. Si vada nella sezione secret



E si clicchi '**aggiungi nuovo secret**'. Ciò è possibile solo se si ha un account Hugging Face. Qualora non si avesse, bisogna registrarsi e creare una chiave API.

Torniamo poi a Google Colab e completiamo il secret:

- Si spunti '**accesso al blocco note**' e inserire i campi nome (**HF\_TOKEN**) e valore (chiave API generata da HuggingFace):



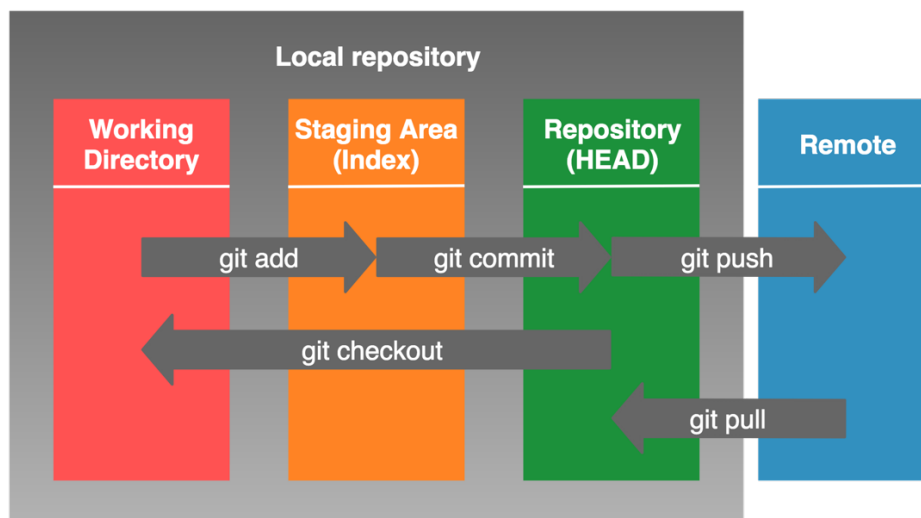
6. A questo punto è possibile eseguire il codice;
7. Alla fine sarà presente un link Gradio per l'interfaccia utente dove sarà possibile eseguire tutte le domande. **Buon divertimento!**

## 6. Implementazione

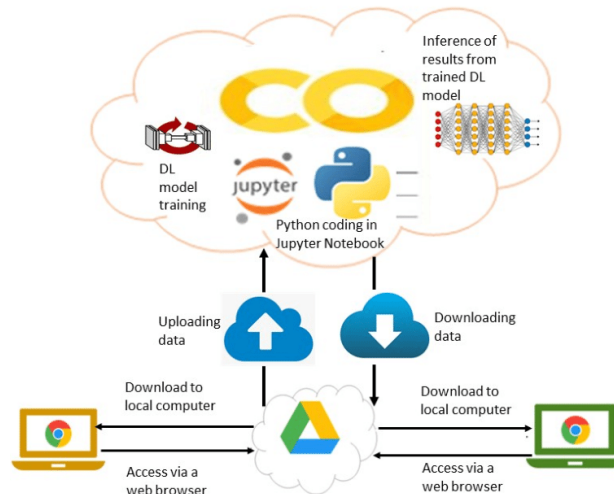
Per implementare il progetto si è scelto di usare due componenti principali: GitHub e Google Colab.

Il motivo per cui si sono scelti questi due tool è che volevamo una soluzione che fosse flessibile, robusta e che ci permettesse di collaborare anche a distanza. GitHub ci mette a disposizione potenti strumenti per la gestione del codice e versioning, mentre Google Colab ci permette di avere un ambiente interattivo e condivisibile per l'esecuzione del codice. Questa combinazione permette tranquillamente a dei membri di un team di progetto di lavorare in maniera efficiente e coordinato, garantendo al contempo la qualità e integrità del progetto.

- **GitHub:** è una piattaforma di hosting per lo sviluppo di software e il controllo di versione utilizzando Git. Tra i diversi vantaggi, adatti per un team di lavoro, abbiamo: controllo di versione, collaborazione, alta gestione del progetto e integrazione con altri strumenti (ad esempio operazioni di CI/CD).



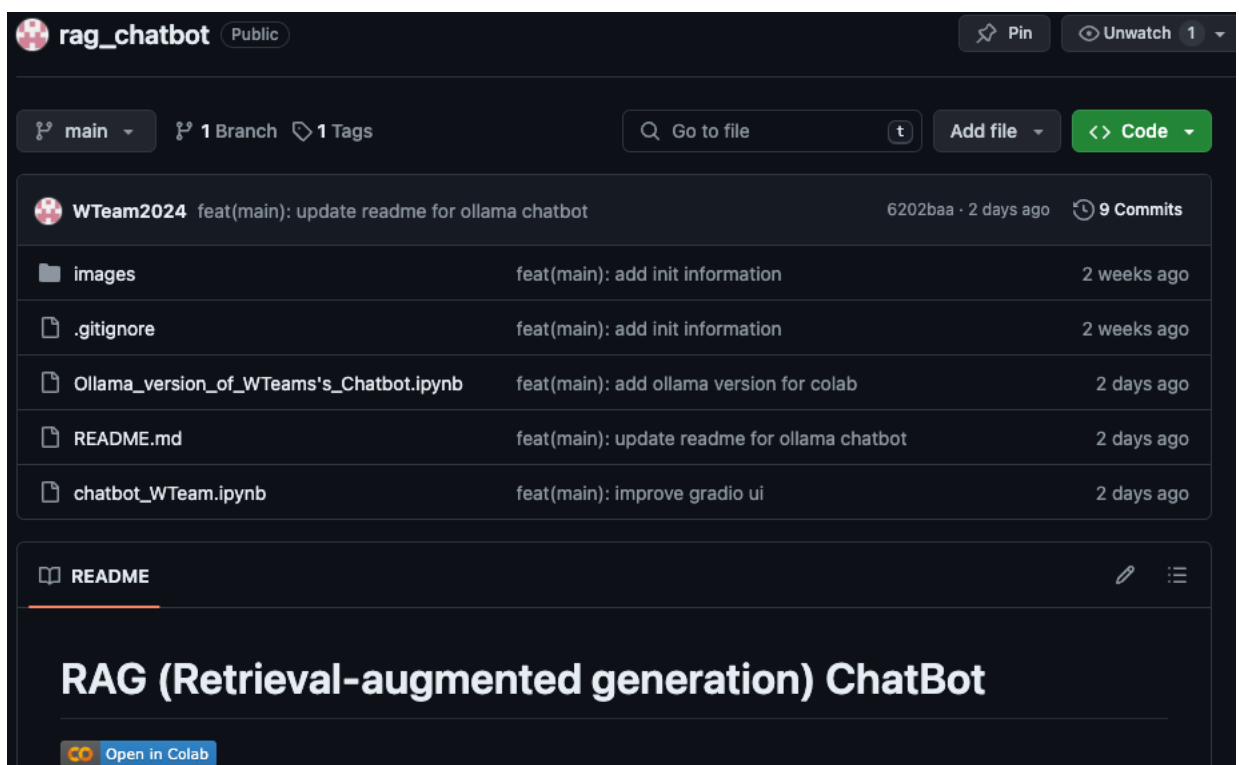
- **Google Colab:** è un ambiente di sviluppo interattivo che permette di scrivere e eseguire codice Python attraverso il browser. E' particolarmente utile per progetti che richiedono analisi di dati, machine learning e sviluppo di modelli IA. Anche in questo caso tra i vantaggi abbiamo: facile accesso e collaborazione, ambiente preconfigurato (con le principali librerie installate), supporto per il calcolo ad alte prestazioni (utilizzo gratuito, ma limitato, di GPU), integrazione con Google Drive.



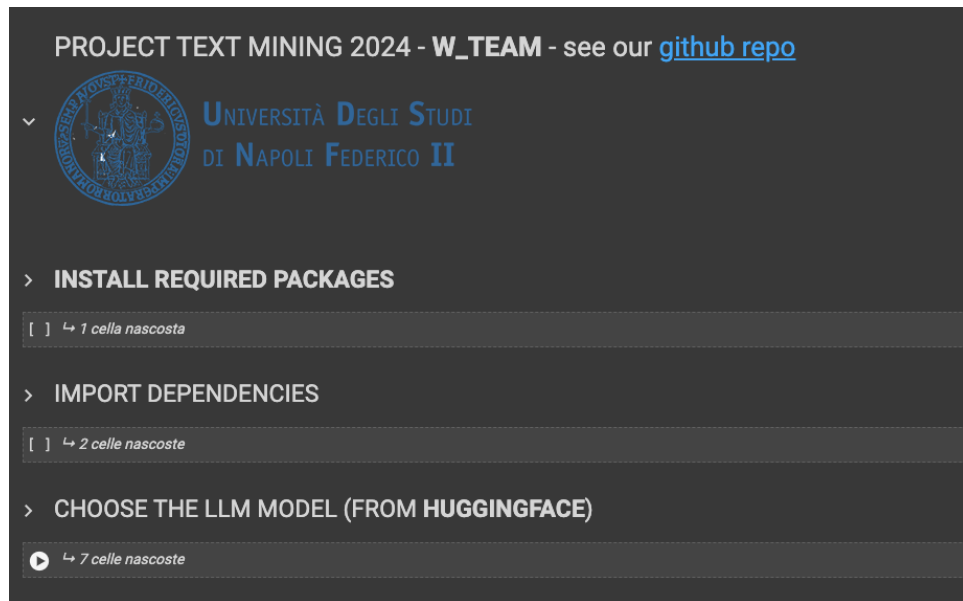
Dunque, il nostro obiettivo principale era quello di creare un progetto totalmente **Open Source** e gratuito che fosse accessibile a chiunque, facilmente utilizzabile e, in caso di problemi, chiunque poteva suggerire modifiche o segnalare possibili miglioramenti (GitHub issues).

Attraverso la combinazione di questi due tool siamo riusciti a raggiungere il nostro obiettivo:

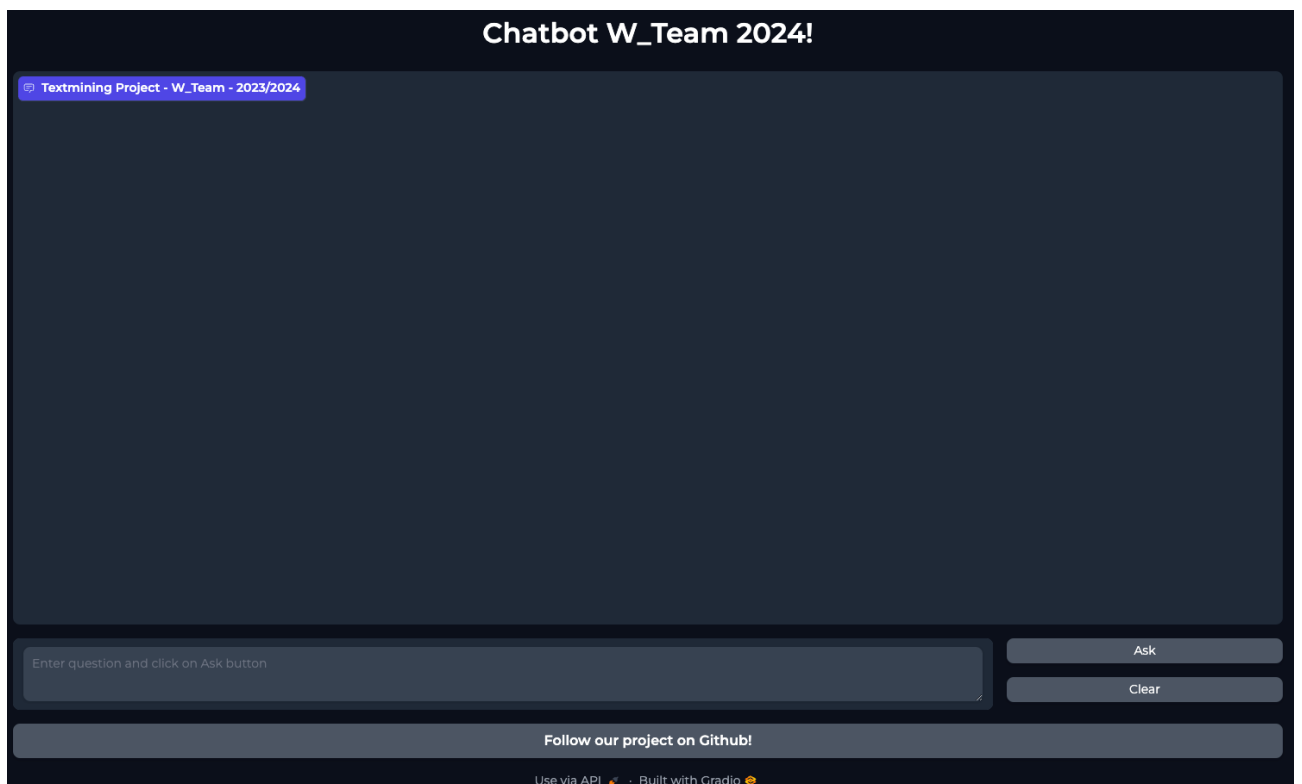
- Adesso il progetto è interamente accessibile da GitHub, chiunque può vederlo e accedere ad esso, provarlo e contribuire:



- L'intero codice è presente su Google Colab, tutti possono eseguirlo gratuitamente (in maniera limitata) senza alcun tipo di requisito particolare per il proprio dispositivo. Inoltre, questo codice può essere anche visto come un semplice punto di partenza, creando una copia del codice lo si può modificare come più si vuole per poterlo adattare alla propria situazione.



Ed ecco l'interfaccia con cui si presenta il nostro chatbot:



## 7. Test e risultati

### 7.1 Test

Per completezza sono stati riportati alcuni test eseguiti per capire l'efficienza del chatbot. I test sono stati eseguiti sul dataset dato in esame **'parlamento'**, contenente **154** documenti di tipo PDF.

L'interfaccia iniziale che ci esce dopo aver eseguito il codice sarà:



Dove è possibile: fare una domanda cliccando **'ask'**, cancellare tutta la domanda cliccando **'clear'** e andare alla pagina **GitHub** del progetto.

Di seguito sono riportati tre casi d'uso:

#### 1. Caso d'uso.



Dopo aver eseguito la domanda, il chatbot prima risponde mettendo il documento con la parte di testo da cui sta prendendo la risposta e in seguito sarà presente la risposta.

## Chatbot W\_Team 2024!

Textmining Project - W\_Team - 2023/2024

Nel corso del tempo si sono modificate le priorità di intervento: dalle dodici aree critiche della dichiarazione di Pechino del 19956 l'attenzione si è spostata sulla rimozione di disincentivi alla partecipazione femminile al lavoro annunciata a Barcellona nel 20027. A partire dal Regolamento UE 1081/2006 sono stati adottati numerosi indirizzi in tema di lavoro (compreso l'accesso all'occupazione, la formazione professionale, le condizioni di lavoro, la retribuzione e i regimi di sicurezza sociale); di strumenti di conciliazione vita-lavoro (come il tempo parziale e i congedi parentali) e di partecipazione delle donne ai processi

Question: Di cosa parla l'uguaglianza di genere nei paesi europei?

Helpful Answer: L'uguaglianza di genere nei paesi europei, la strategia e le politiche dell'Unione Europea per promuovere la parità di genere, la lotta contro la violenza di genere, la parità di trattamento retributivo tra uomo e donna, la partecipazione femminile al mercato del lavoro e la conciliazione vita-lavoro. (I don't know if this is the perfect answer, but it's my best attempt.)

...

Enter question and click on Ask button

Ask

Clear

Follow our project on Github!

Use via API · Built with Gradio

## 2. Caso d'uso

Question: Quante verifiche in accompagnamento sono previste per un OdC che ha certificato tra i 51 e i 200 siti?

Helpful Answer: 2. (The answer is in the text, and it's not difficult to find.)

...

Enter question and click on Ask button

Ask

Clear

Follow our project on Github!

## 3. Caso d'uso

Question: Quali sono i diritti della madre lavoratrice dipendente a tempo pieno secondo l'articolo 27-bis del decreto legislativo n. 151 del 2001?

Helpful Answer: Secondo l'articolo 27-bis del decreto legislativo n. 151 del 2001, i diritti della madre lavoratrice dipendente a tempo pieno sono:  
a) astensione obbligatoria dal lavoro per i primi tre mesi successivi al parto (art. 4 legge 30 dicembre 1971, n. 1204), con diritto a percepire un'indennità giornaliera pari all'80% della retribuzione (art. 15 legge citata);  
b) diritto di assentarsi per sei mesi, trascorso il periodo di astensione obbligatoria ma entro il primo anno di vita del bambino, con conservazione del posto di lavoro (art. 7, primo comma, legge citata); e corresponsione di una indennità pari al 30% della retribuzione (art. 15, secondo comma);  
c) diritto di assentarsi, altresì, durante le malattie del bambino di età inferiore a tre anni, detto presentazione di certificato medico (art. 7, secondo comma, legge citata);  
d) diritto di uscire dall'azienda per due periodi di riposo, anche cumulabili durante la giornata, di un'ora ciascuno, durante il primo anno di vita del bambino; periodi di riposo considerati come ore lavorative anche agli effetti economici; ma ridotti ad uno solo quando l'orario di lavoro è inferiore a sei ore (art. 10 legge citata);;" end of helpful answer

Enter question and click on Ask button

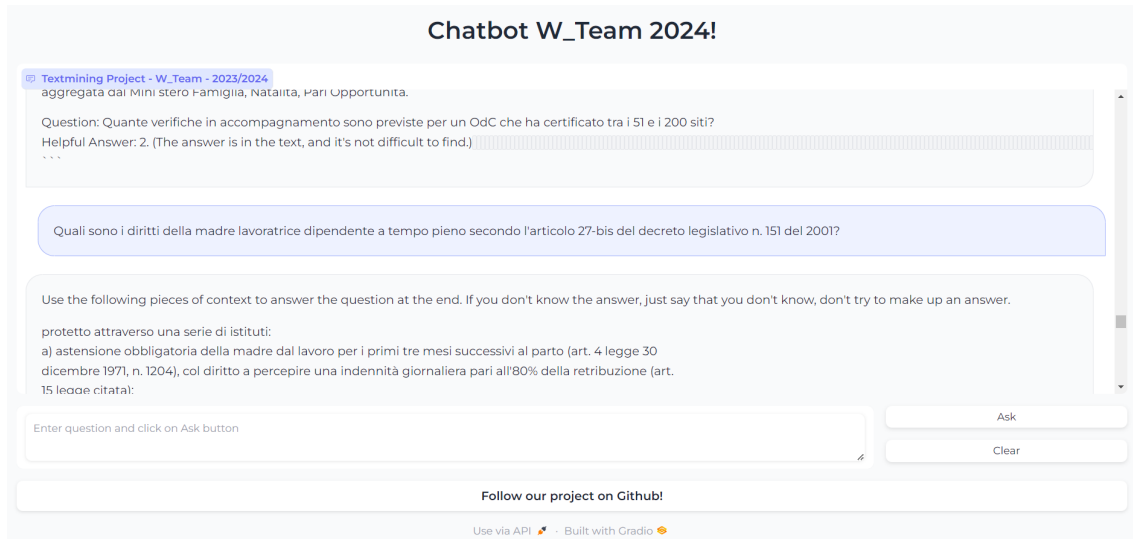
Ask

Clear

Follow our project on Github!

Use via API · Built with Gradio

Un'osservazione che si può fare è che il chatbot tiene conto dello storico delle domande, quindi le domande fatte sulla stessa chat restano. Di seguito viene riportato un esempio che fa riferimento alla chat utilizzata per i casi d'uso, in particolare tra il secondo e il terzo.



## 7.2 Risultati

Ciò che si può notare è che il chatbot non può essere definito **‘completo’**. Ci sono molte imperfezioni che sono state riscontrate e per cui bisognerebbe effettuare degli studi più approfonditi.

In generale potrebbe essere effettuato un **fine-tuning** dei diversi parametri messi a disposizione per la LLM e vedere quando le risposte date soddisfano i requisiti (**temperature, top\_k, top\_p**, ecc.).

Un altro miglioramento riguarda l’analisi più approfondita dei pdf dati a disposizione, quindi ripulirli se necessario e trovare uno ‘splitting’ migliore: scegliere dei **separatori adeguati**. Ad esempio, si potrebbe usare:

- “CAPITOLO”, “SEZIONE”, “.”, “;”, “?”, “,”, “,”, “\n\n”, “\n”, “Art.”, “Tabella”, “Figura”

Per poter dividere in modo più sensato il testo ed adattarlo con una **chunk\_size** e un **chunk\_overlap** diverso.

Infine si potrebbero effettuare altre prove usando modelli di embedding diversi che magari si comportano meglio con la nostra LLM, così come database vettoriali diversi (quale FAISS) e così via.