

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

Corso di Laurea Magistrale in Ingegneria Informatica



PROGETTO TEXT MINING: CHATBOT PANDAS

a.a. 2023-24

Studenti:

Angelo Andrea Nozzolillo - M63/1602

Pierluigi Montieri – M63/1584

Stefano Angelo Riviello – M63/1592

Luca Migliaccio – M63/1573

Indice

Capitolo 1: Introduzione	3
Capitolo 2: Principi di funzionamento	3
Capitolo 3: Blocchi funzionali	5
Capitolo 4: Diagramma di flusso del chatbot	7
Capitolo 5: Prem AI - Implementazione	8
Capitolo 6: Test & Risultati	10
Capitolo 7: Conclusioni	11
Capitolo 8: Riferimenti	11

Capitolo 1: Introduzione

In generale, i **chatbot** vengono utilizzati per il recupero di informazioni e, in genere, funzionano in base ad alcune regole predefinite e alla corrispondenza delle parole chiave. Ciò può essere visto come una sorta di *insieme di regole if-else* predefinite dagli utenti. Quando un utente inserisce una *query*, il chatbot cerca parole chiave o modelli specifici nella query per identificare la risposta appropriata.

I chatbot tradizionali si basano su una *knowledge base* fissa o su un *database* di risposte predefinite (le risposte vengono inserite manualmente nel database dallo sviluppatore). Quando un utente inserisce una query, il chatbot cerca le **regole** adatte in base alla domanda. Una volta trovata, fornisce la risposta che è associata ad essa. Dunque, non fa alcuna parafrasi o non può eseguire alcuna generazione. Al giorno d'oggi, i chatbot basati su **LLM (Large Language Model)** sono in voga. I chatbot basati su LLM possono essere di due tipi:

1. *Chatbot basati su LLM senza RAG*: LLM, come quelli di OpenAI o Llama, sono addestrati con miliardi di parametri e con enormi quantità di dati testuali. Alcuni di questi modelli sono open-source e possono essere utilizzati gratuitamente, mentre altri no. Possiamo utilizzare il chatbot per il nostro scopo utilizzando l'**API** fornita dall'organizzazione responsabile di questi LLM. Tuttavia, il problema è che, quando un utente pone una domanda, il chatbot risponde direttamente dai dati su cui è stato addestrato, senza considerare alcuna base di conoscenza esterna (funzionerà proprio come ChatGPT).
2. *Chatbot basati su LLM con RAG*: **RAG** sta per **Retrieval-Augmented Generation** e ha due componenti principali: *generazione* e *recupero*. A differenza dei chatbot del caso precedente, qui vengono *utilizzate fonti di dati esterne* come PDF, testi e database come base di conoscenza insieme al modello LLM addestrato. In questo caso, quando un utente fa una domanda, il sistema prima cerca un frammento di testo (**chunk**) simile nella base di conoscenza esterna (*recupero*). Questi frammenti di testo vengono utilizzati come prompt per il modello LLM. Basandosi sul contesto e sulla query dell'utente, il modello LLM può creare una risposta più precisa e creativa, che può essere definita *generazione*. Questo non è possibile con altri tipi di chatbot.

Questo progetto si propone di sviluppare un *chatbot su LLM con RAG* che, a partire da documenti caricati, sia in grado di rispondere correttamente alle domande degli utenti. L'**obiettivo principale** è garantire che il chatbot comprenda il contenuto dei documenti e fornisca risposte pertinenti ed accurate.

Capitolo 2: Principi di funzionamento

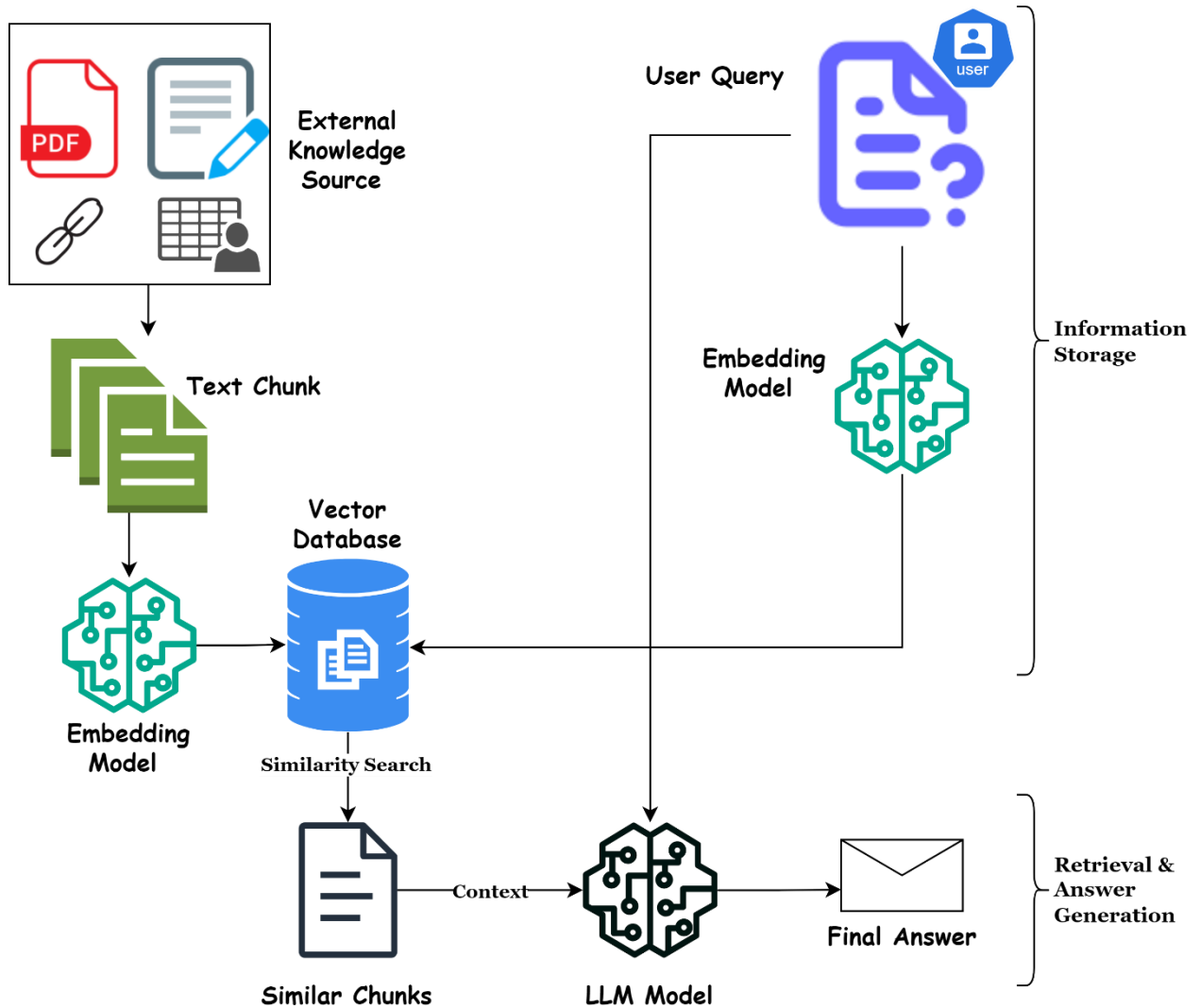
Nel seguente capitolo andiamo ad introdurre il funzionamento della seconda famiglia di chatbot presentati precedentemente, ovvero dei chatbot basati su RAG. Tale *funzionamento* può essere suddiviso in tre differenti punti:

1. **Immagazzinamento dell'informazione**: diversi documenti possono essere utilizzati come fonti di dati esterne. Quando un documento viene caricato per primo utilizzando *RecursiveCharacterTextSplitter*, i documenti di testo vengono suddivisi in *chunk* utilizzando la *sovrapposizione* o la *non sovrapposizione*. Viene quindi utilizzato un **modello di embedding** per creare un **vettore di embedding** di questi chunk per acquisire il significato semantico. Questi vettori vengono quindi memorizzati in set di dati vettoriali utilizzando l'indicizzazione per un recupero rapido e preciso delle informazioni.
2. **Recupero delle Informazioni**: quando un utente inserisce una query testuale, utilizzando lo stesso modello di embedding, questo testo viene convertito in un vettore di embedding e passato al database vettoriale per il recupero delle informazioni. Qui viene utilizzato il concetto di *ricerca per similarità*. Diverse tecniche, come la similarità coseno (*cosin similarity*), la distanza L1 o la distanza

L2, vengono utilizzate per recuperare frammenti di testo contestualmente simili alla query dell'utente. Dopo il recupero, questi frammenti di documenti vengono passati al modello LLM per generare la risposta finale.

3. **Generazione della Risposta:** questo è l'ultimo passaggio del chatbot. Dopo il recupero delle informazioni, i vettori di embedding vengono passati al modello LLM. Questo viene utilizzato come prompt per il modello. Il modello è in grado di comprendere il contesto per cui deve generare la risposta. Infine, utilizzando il contesto, il modello generativo LLM genera il risultato finale.

Nella seguente figura è rappresentata una panoramica di alto livello del chatbot basato su RAG.

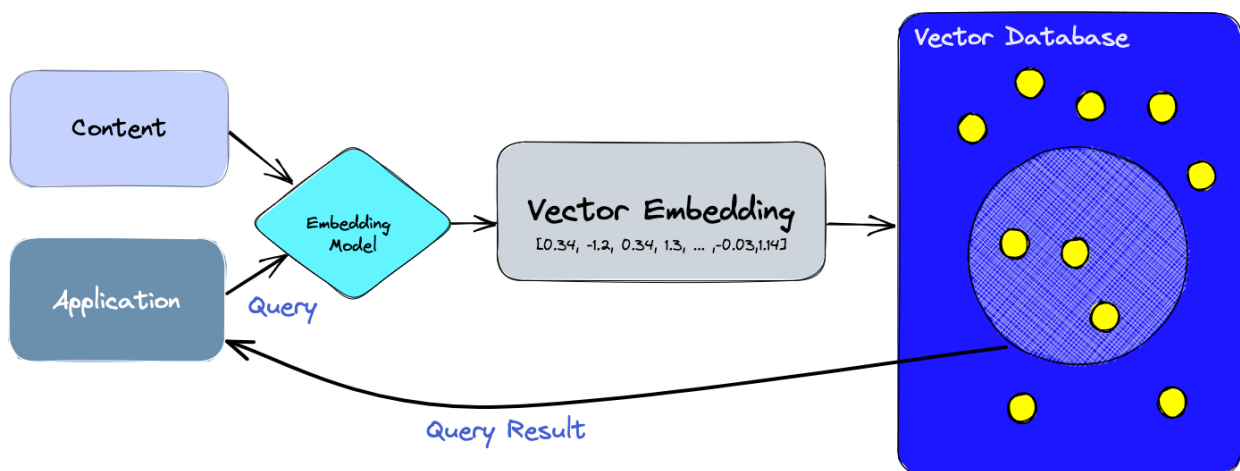


Capitolo 3: Blocchi funzionali

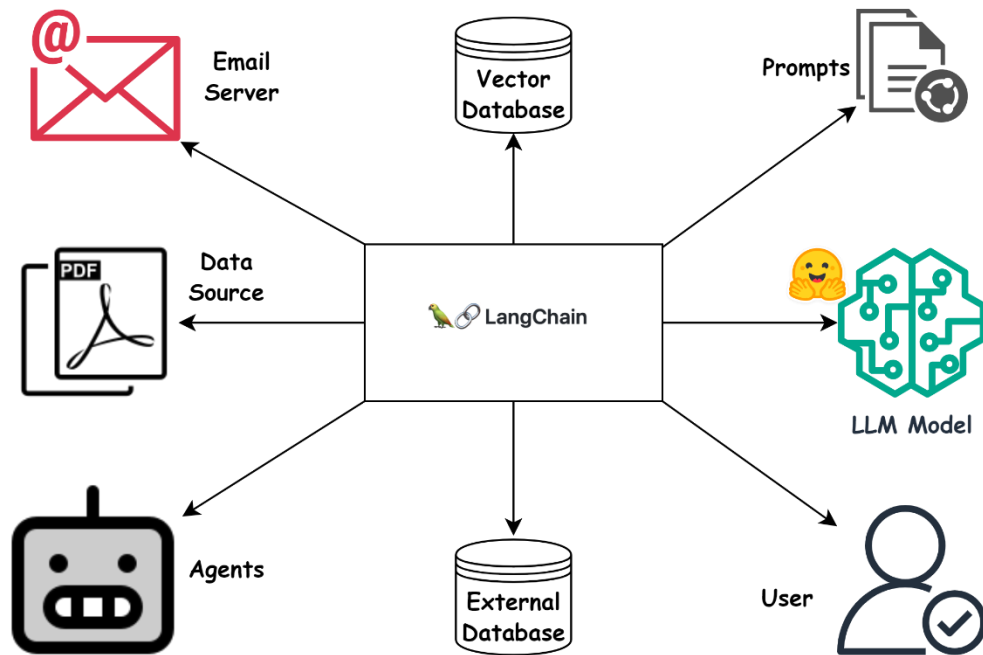
Il chatbot è composto da diversi blocchi funzionali, ognuno con specifiche funzionalità e compiti da svolgere. In questo capitolo, cercheremo di capire il funzionamento di questi singoli elementi del chatbot e come vengono integrati per implementare il chatbot finale funzionante.

1. **Database Vettoriale (chunk_embeddings.pkl)** → in generale, i dati testuali non possono essere elaborati direttamente dai LLM. Per alimentare questo testo non strutturato, è necessario generare dei numeri a partire da esso. Sarebbe utile se potessimo generare numeri da questi testi e catturarne il significato semantico. Qui entra in gioco l'**embedding delle parole**, che genera un **vettore di embedding** a partire dai testi. Di solito, questi vettori sono altamente dimensionali e difficili da gestire con un normale database relazionale. Possiamo memorizzarli in un database relazionale, ma quando eseguiamo una ricerca semantica per trovare un contesto simile, verrà utilizzata una *ricerca lineare* che è *dispendiosa* in termini di tempo e *costosa* dal punto di vista computazionale.

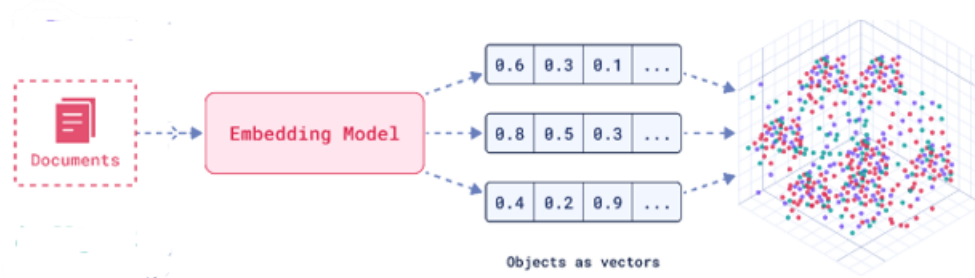
Per risolvere questo problema, vengono introdotti i **database di vettori** che utilizzano l'*Hashing Sensibile alla Località (LSH)* per raggruppare dati simili in base al loro significato semantico nello stesso bucket e indicizzarli per un rapido recupero. La figura successiva mostra il *principio di funzionamento ad alto livello del database di vettori*. Durante il lavoro con i modelli LLM, sono disponibili diversi database di vettori come Chroma DB e Faiss. In questo progetto, abbiamo utilizzato **Faiss** (indirettamente) per creare un database di vettori e indicizzare i dati localmente sulla macchina locale per una ricerca semantica e un recupero delle informazioni veloci.



2. **LangChain** → è un framework che viene utilizzato per l'accelerazione e la semplificazione dello sviluppo di applicazioni utilizzando diversi LLM. Viene utilizzato per connettere il modello linguistico con diverse origini di dati esterne. Se consideriamo la *connessione di rete hub and spoke*, qui LangChain funziona come un hub che collega altri spoke che potrebbero essere di tipo diverso. Se abbiamo un database aziendale e vogliamo utilizzare LLM per generare una query basata sull'input testuale dell'utente e inviare la risposta in base alla query dobbiamo collegare l'LLM al Database e l'User Prompt a LLM. Questa integrazione può essere effettuata utilizzando LangChain. Ci sono altri framework come LangChain come **LlamaIndex** che possono essere utilizzati anche per svolgere questo compito. In questo progetto, abbiamo utilizzato il frame LangChain per creare connessioni con diverse risorse. Nella figura seguente sono mostrati i diversi casi d'uso di LangChain e il modo in cui connette risorse diverse. In particolare, si può osservare come LangChain sia un framework multiuso e possa essere utilizzato per integrare diverse fonti per l'esecuzione di un compito specifico.



3. **Modello di Embedding** → lo **scopo** di un modello di embedding è quello di *generare il vettore di embedding in base al blocco di input fornito*. Sono disponibili diversi modelli di embedding di parole pre-addestrati. Abbiamo usato *text-embedding-3-large* di **Prem AI** come modello di embedding. Sia per le query utente che per le origini esterne, viene utilizzato lo stesso modello. Nella figura illustrata di seguito viene mostrata una panoramica generale del processo di embedding.



4. **Divisione ricorsiva del testo** → come criterio per la suddivisione del testo abbiamo utilizzato un insieme di criteri, indicati come segue: `["\n\n", "\n", " ", ":", ":", "\u200b", "\uff0c", "\u3001", "\uff0e", "\u3002", ""]`. Questo processo cerca di tenere insieme tutti i paragrafi (frasi e parole) il più a lungo possibile, poiché essi sembrerebbero genericamente essere i pezzi di testo semanticamente correlati più forti. Qui abbiamo usato *RecursiveCharacterTextSplitter* di **LangChain** con quattro parametri: dimensione del chunk, dimensione della sovrapposizione, funzione per calcolare la lunghezza del testo e un parametro per gestire i separatori (oltre che la lista di separatori indicata precedentemente).

Capitolo 4: Diagramma di flusso del chatbot

Nel seguente capitolo illustriamo i passi dell'intero **diagramma di flusso** del nostro chatbot.

1. Inizializzazione dell'ambiente

- Imposta la variabile d'ambiente **PREMAI_API_KEY**.
- Definisci il modello e inizializza l'oggetto **PremAIEmbeddings** con il modello e l'ID del progetto.

2. Script per creare Embeddings dei chunk

- Definisci il modello e inizializza l'oggetto **PremAIEmbeddings**.
- Definisci il percorso della directory e leggi tutti i file .txt.
- Definisci un text splitter usando *RecursiveCharacterTextSplitter*.
- Dividi i documenti in chunk.
- Genera embeddings per ogni chunk.
- Salva i chunk e i loro embeddings in **chunk_embeddings.pkl**.
- Stampa un messaggio di conferma.

3. Caricamento dei dati

- Apri e carica il file *chunk_embeddings.pkl* che contiene *all_chunks* e *chunk_embeddings*.

4. Definisci la Funzione di Supporto

- Funzione *find_most_similar_chunks*: trova i chunk più simili a una query basata sulla similarità coseno.

5. Inizializzazione dell'App Streamlit

- Configura le impostazioni della pagina Streamlit e lo stile.
- Mostra il titolo e il sottotitolo della pagina.
- Inizializza la lista dei messaggi nello stato della sessione di Streamlit se non già presente.

6. Interazione con l'utente

- Mostra una casella di input per la query dell'utente.
- Se viene fornita una query:
 - Mostra uno spinner di caricamento.
 - Crea l'embedding della query dell'utente usando **embedder**.
 - Trova i chunk più simili alla query.
 - Combina il testo dei chunk più simili.
 - Inizializza il modello **ChatPremAI**.
 - Crea messaggi di sistema e messaggi "umani".
 - Invoca il modello di chat per ottenere una risposta.
 - Aggiungi la query dell'utente e la risposta del chatbot ai messaggi dello stato della sessione.

7. Visualizza messaggi

- Itera sui messaggi nello stato della sessione e visualizzali in ordine inverso, differenziando tra messaggi dell'utente e del chatbot.

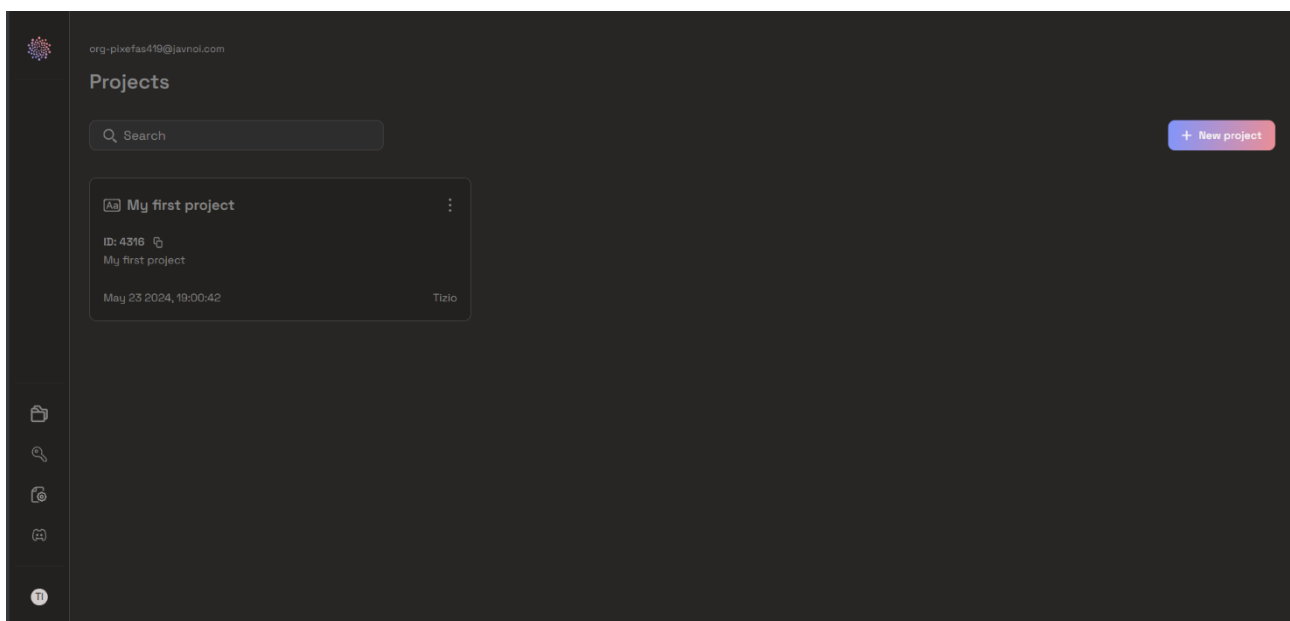
Capitolo 5: Prem AI- Implementazione

Il sito **PREM AI** rappresenta una piattaforma innovativa focalizzata sull'intelligenza artificiale (IA) e il machine learning (ML). La sua missione principale è quella di rendere accessibili e comprensibili le tecnologie avanzate di IA a un pubblico ampio, che spazia dai neofiti agli esperti del settore.

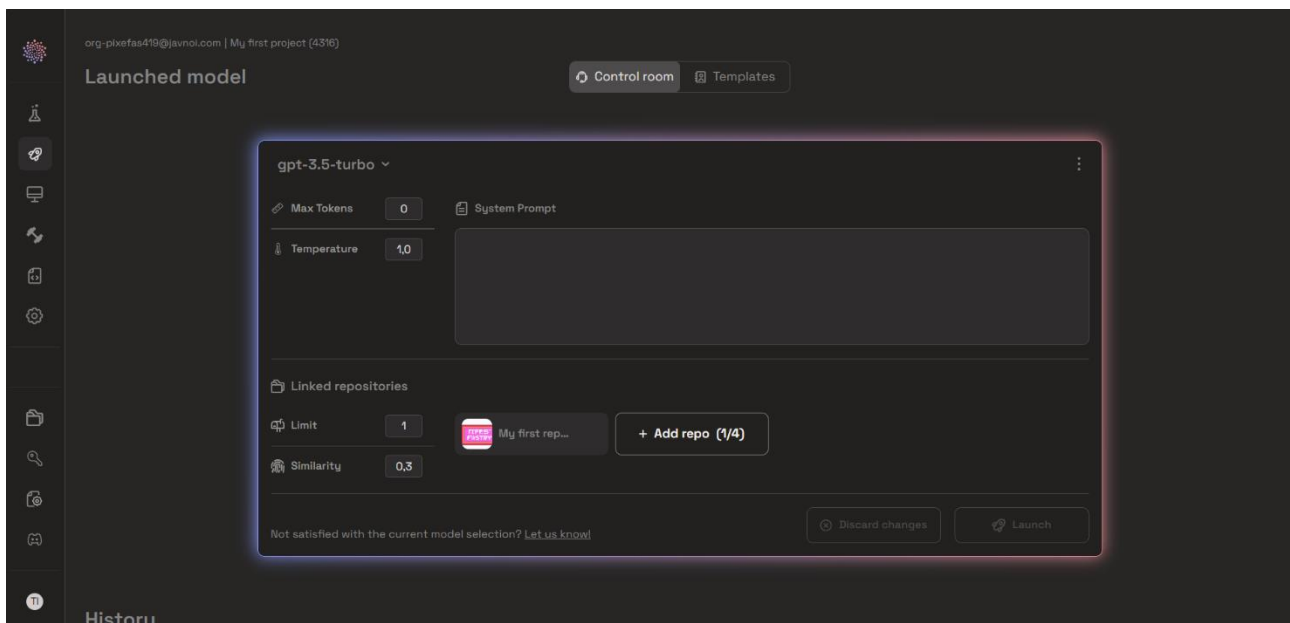
Tra le caratteristiche principali vi è la possibilità di accedere ad un'ampia collezione di API e librerie per facilitare l'integrazione delle tecnologie di IA nelle applicazioni.

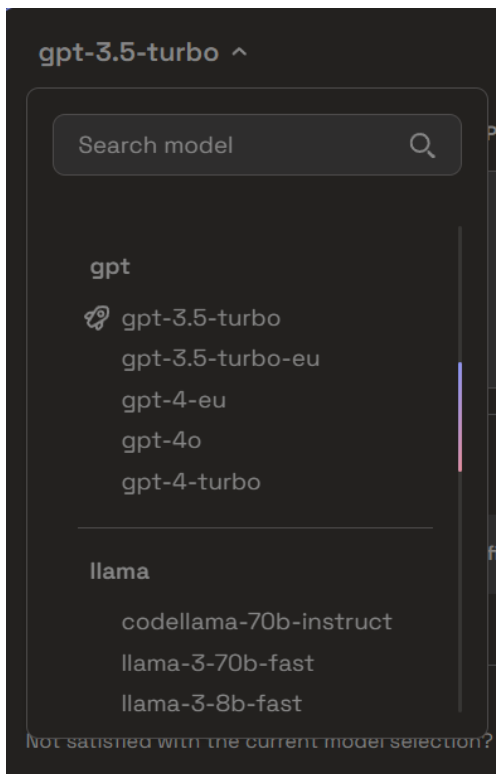
L'**obiettivo principale** di PREM AI è quello di abbattere le barriere all'entrata nel mondo dell'intelligenza artificiale, fornendo risorse accessibili e strumenti potenti a chiunque sia interessato a esplorare questo campo. Offre un ecosistema completo per l'apprendimento, lo sviluppo e la collaborazione, rendendo la tecnologia di intelligenza artificiale più accessibile e comprensibile.

Per utilizzare PREM AI abbiamo creato nella main page il nostro progetto, il quale ha il proprio project ID (4316).



Abbiamo settato nel nostro progetto due parametri: *temperature* e *similarity*. Essi sono parametri chiave che possono influenzare la capacità di un modello di simulazione di generare risultati simili ai dati reali, migliorando così la calibrazione e l'affidabilità del modello stesso. La prima viene utilizzata per trovare la

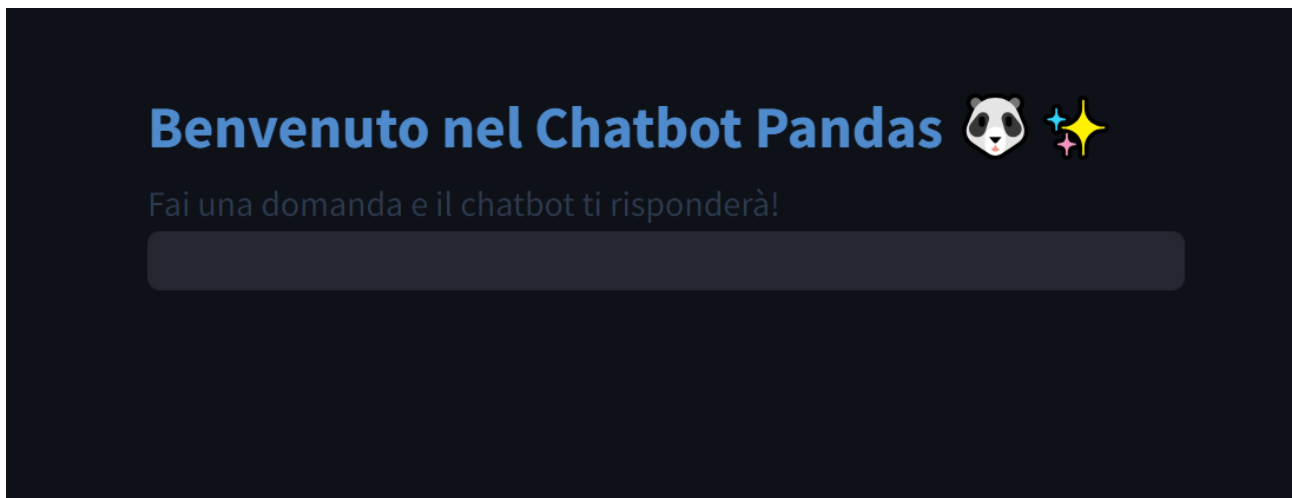




miglior corrispondenza o similitudine tra due insiemi di dati, mentre la seconda determina quanto il risultato della simulazione cambia in risposta a variazioni nei parametri di input. Notiamo che una sensibilità elevata può indicare che il modello è molto reattivo a piccoli cambiamenti, il che può influenzare la misura di similarità se i dati di input sono soggetti a rumore o incertezza.

Abbiamo testato qual era il modello più adatto facendo un compromesso tra performance e risorse richieste, ed alla fine abbiamo optato per utilizzare *gpt-3.5-turbo*. Infine, abbiamo integrato tale modello all'interno della nostra applicazione effettuando le chiamate API.

Il chatbot è ospitato su **Streamlit**, una potente libreria di Python per ospitare e creare applicazioni interattive. L'interfaccia si presenta come mostrato di seguito:



N.B. Per ulteriori chiarimenti sulle fasi seguite per sviluppare l'applicazione, si faccia riferimento al *capitolo 4*.

Capitolo 6: Test & Risultati

A scopo dimostrativo, abbiamo considerato una serie di domande realizzate a partire dal *dataset* “*RegistroCameraDeputati*”, contenente 12 file con estensione .docx. Di seguito riportiamo due *casi d’uso* della nostra applicazione:

- *Domanda:* chi è Antonucci Marco?

Benvenuto nel Chatbot Pandas 🐼 ✨

Fai una domanda e il chatbot ti risponderà!

chi è Antonucci Marco?

Antonucci Marco è un rappresentante di interessi che si è accreditato per l'Ordine dei Tecnici Sanitari di Radiologia Medica e delle Professioni Sanitarie Tecniche delle Province di Firenze, Arezzo, Prato, Pistoia, Lucca e Massa Carrara. Si occupa di monitoraggio legislativo e attività propositiva in ambito delle professioni intellettuali e sanitarie, con particolare riguardo alla responsabilità professionale e alle politiche sanitarie. La sua data di accredito è il 16/06/2023 con data termine attività il 31/12/2024.

chi è Antonucci Marco?

- *Domanda:* quando si è accreditato Antonucci Marco?

Benvenuto nel Chatbot Pandas 🐼 ✨

Fai una domanda e il chatbot ti risponderà!

quando si è accreditato Antonucci Marco?

Antonucci Marco si è accreditato il 16/06/2023.

quando si è accreditato Antonucci Marco?

Antonucci Marco è un rappresentante di interessi che si è accreditato per l'Ordine dei Tecnici Sanitari di Radiologia Medica e delle Professioni Sanitarie Tecniche delle Province di Firenze, Arezzo, Prato, Pistoia, Lucca e Massa Carrara. Si occupa di monitoraggio legislativo e attività propositiva in ambito delle professioni intellettuali e sanitarie, con particolare riguardo alla responsabilità professionale e alle politiche sanitarie. La sua data di accredito è il 16/06/2023 con data termine attività il 31/12/2024.

chi è Antonucci Marco?

N.B. è importante notare come, nella medesima chat, vengano mantenute le domande/risposte effettuate precedentemente.

Capitolo 7: Conclusioni

I chatbot basati su RAG sono senza dubbio una delle innovazioni più importanti, nonché strumenti che possono essere utilizzati da varie aziende di diversi domini. Qui, nel nostro caso, abbiamo utilizzato LLM open source che sono gratuiti. Tuttavia, questi tipi di chatbot basati su LLM necessitano di un'elevata potenza di calcolo per funzionare in modo fluido ed efficace. Nella maggior parte dei casi, gli LLM a pagamento come OpenAI e Google Gemini funzionano meglio dell'LLM falcon-7b. È possibile utilizzare la stessa architettura e lo stesso flusso di lavoro utilizzando vari LLM e modelli di incorporamento per creare chatbot basati su rag e verificare quale modello funziona in base alle proprie esigenze specifiche.

Capitolo 8: Riferimenti

- <https://docs.premai.io/introduction>
- <https://www.langchain.com/>
- <https://streamlit.io/>
- <https://github.com/nozzolillo01/PANDAS.git>
- <https://docs.premai.io/get-started/supported-models>
- https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/