Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Gestionale

A.a. 2023/2024

Sessione di Laurea luglio 2024

Q&A Information Retrieval

Progettazione, osservazione ed analisi di una sperimentazione didattica sull'intelligenza artificiale generativa nell'ambito di un corso di Basi di Dati

Relatori:
Prof. Laura Farinetti
Ing. Lorenzo Canale

Candidato:
Edoardo Rinaldi

Sommario

Il presente lavoro di tesi descrive e analizza la prima edizione del progetto didattico “Q&A Information Retrieval” sull’intelligenza artificiale generativa, svolto nell’ambito di un insegnamento di Basi di Dati. Gli studenti coinvolti sono iscritti al terzo anno del corso di laurea in Ingegneria del Cinema e dei Mezzi di Comunicazione del Politecnico di Torino.

L’obiettivo è quello di presentare un metodo di insegnamento che adotta questa tecnologia all’avanguardia per stimolare la consapevolezza degli studenti riguardo all’affidabilità dei modelli linguistici. Infatti, il progetto consiste nello sviluppo di un’applicazione in grado di rispondere a delle domande su un argomento specifico e nella valutazione dell’output ottenuto. Nello specifico, gli studenti hanno svolto le seguenti attività:

- Selezionare un argomento specifico, raccogliere una serie di documenti che servano da base di conoscenza per l’applicazione da sviluppare e definire 10 domande di prova per testarla.
- Creare una prima versione del chatbot a partire dalla LangChain, un pacchetto Python per lo sviluppo di applicazioni che si basano su modelli linguistici.
- Cambiare alcune variabili del modello, confrontare le diverse configurazioni ottenute attraverso delle metriche di valutazione e selezionare la migliore.
- Collegare la configurazione scelta ad un canale Telegram per facilitarne l’utilizzo.
- Valutare le prestazioni del chatbot rispetto ai risultati ottenuti con ChatGPT, che al momento rappresenta il modello di intelligenza artificiale generativa più utilizzato.

I modelli linguistici su cui si basano le applicazioni come ChatGPT negli ultimi mesi hanno dimostrato di essere estremamente utili in vari contesti. Tuttavia, essi sono caratterizzati dal rischio di fornire informazioni inaccurate a causa dell'imprecisione intrinseca degli algoritmi che ne regolano il funzionamento. Per assumere consapevolezza riguardo a questa realtà, agli studenti è stato chiesto specificamente di eseguire una fase di valutazione per mitigare il rischio di disinformazione. L'obiettivo di apprendimento di questa esperienza pratica è quindi duplice: da un lato, esercitarsi con le tecnologie emergenti e, dall'altro, valutare l’affidabilità dei risultati con uno spirito critico.

L’elaborato inizia con una panoramica di alto livello sui modelli conversazionali e il loro utilizzo nell’ambito della formazione universitaria. A seguire verrà introdotta la LangChain e i componenti facenti parte dell’architettura del chatbot. Verranno illustrate le singole fasi dell’esperienza e presentate varie analisi dei risultati ottenuti. Infine, saranno discusse alcune proposte di miglioramento per lo svolgimento di eventuali edizioni future del progetto.

Indice generale

Elenco delle figure

Elenco delle tabelle

1 Introduzione

L'intelligenza artificiale generativa è una recente tecnologia in grado di produrre automaticamente contenuto, sotto forma di testo, immagini, audio, video o altri media, in risposta alle richieste dell'utente. I modelli di intelligenza artificiale generativa recepiscono e processano dei prompt, analizzano e rielaborano i dati in loro possesso, e forniscono risposte strutturate tramite tecniche di elaborazione del linguaggio naturale (NLP).

ChatGPT è il modello di linguaggio generativo più utilizzato tra quelli attualmente esistenti. È stato sviluppato da OpenAI e si basa sull'architettura del Generative Pre-trained Transformer (GPT), un tipo di Large Language Model (LLM). Il GPT è stato allenato su una grande quantità di documenti testuali presenti online e di conseguenza possiede una base di conoscenza generale su molti argomenti di varia natura. I potenziali casi d'uso dei modelli come questo sono molteplici. Solitamente li si utilizzano per rispondere a domande aperte o chiuse, produrre contenuto creativo, tradurre, risolvere esercizi numerici o programmare.

Tuttavia, è importante specificare che tutti i modelli conversazionali rispondono in base ai dati su cui sono stati addestrati e a quelli presenti nei prompt. Non si può dire che essi abbiamo una reale comprensione delle tematiche trattate od una propria consapevolezza. Pertanto, possono generare risposte talvolta errate o inappropriate. È fondamentale che l'utente sia conscio di questi limiti e verifichi sempre il risultato fornito dal modello attraverso un'analisi critica.

Negli ultimi mesi, l'intelligenza artificiale generativa ha ottenuto sempre più attenzione nel mondo della formazione. Un gran numero di articoli scientifici dimostrano che ChatGPT e altri LLM sono stati utilizzati per fare esperimenti in vari contesti didattici. In particolare, facoltà come l'ingegneria sono per natura molto orientate verso le tecnologie all'avanguardia, per cui l'interesse nell'utilizzo di questi modelli come strumento a supporto dell'insegnamento e dello studio sta aumentando.

I docenti dell'insegnamento di Basi si dati del corso di laurea in Ingegneria del cinema e dei mezzi di comunicazione del Politecnico di Torino hanno scelto di unirsi a questa tendenza progettando la sperimentazione didattica dal nome "Q&A Information Retrieval". Questa esperienza consiste nel far creare agli studenti un chatbot personalizzato tramite l'utilizzo di alcuni LLM open source e nel farglielo valutare. La motivazione che ha portato a quest'idea risiede nella natura dell'insegnamento di Basi di dati: il contenuto che viene tradizionalmente erogato a lezione offre agli studenti opportunità limitate per allenare competenze come la creatività e il pensiero critico.

L'obiettivo di questo progetto non è quello di valutare l'utilità del chatbot per l'apprendimento, tema già affrontato da molti studi, ma di esporre gli studenti a questa tecnologia, facendogliela toccare con mano e chiedendo loro di misurarne le prestazioni in termini di affidabilità. Tuttavia, poiché non si tratta di studenti di Ingegneria informatica, le attività proposte non sono focalizzate sui dettagli tecnici dei LLM, ma sull'utilizzo di strumenti consolidati che permettono l'assemblaggio dei componenti necessari e sulla valutazione dell'efficacia del chatbot sviluppato.

L'intero processo viene implementato in Python[1]. Per la scrittura del codice vengono utilizzati i Jupyter notebook[2]. Un Jupyter notebook è un'applicazione web che serve a creare e condividere documenti interattivi contenenti testo, codice eseguibile e output. Il nome "Jupyter" deriva dai linguaggi di programmazione principali supportati: Julia, Python e R. In un Jupyter Notebook, si può scrivere codice in celle ed eseguire queste una alla volta, ottenendo il risultato direttamente sotto la cella stessa.

Per eseguire i notebook gli studenti hanno utilizzato Colab[3], un servizio cloud gratuito offerto da Google. Google Colab è particolarmente popolare tra gli studenti, i ricercatori e i professionisti che desiderano eseguire codice Python senza dover gestire l'infrastruttura sottostante.

[1] https://www.python.org/

[2] https://jupyter.org/

[3] https://colab.google/

2 LangChain

Il progetto “Q&A Information Retrieval” prevede che gli studenti creino un chatbot utilizzando la LangChain[4], un pacchetto Python per lo sviluppo di applicazioni alimentate da modelli linguistici. L’obiettivo di questo capitolo è quello di spiegare il funzionamento di questo strumento, mettendone in luce i vari componenti.

Prima di esaminare la struttura dettagliata della LangChain, è importante comprendere le diverse situazioni in cui questo framework viene utilizzato. Le sue applicazioni sono molteplici e possono essere classificate in diverse categorie di complessità:

- **Question & answer (Q&A):** applicazioni in grado di fornire risposte a domande complesse basandosi su informazioni provenienti da fonti specifiche. Sono fondamentali per l'elaborazione di domande e la restituzione di risposte accurate.
- **Chatbot:** applicazioni progettate per mantenere conversazioni prolungate e contestualizzate con gli utenti. Offrono un'interazione più dinamica e naturale rispetto ai tradizionali sistemi di Q&A.
- **Agenti:** applicazioni che agiscono in modo autonomo per determinare quali strumenti utilizzare tra quelli disponibili, al fine di generare una risposta all'utente. Possono integrare una vasta gamma di funzionalità per fornire un servizio completo ed efficace.

Queste categorie rappresentano solo alcune delle molteplici modalità in cui la LangChain può essere utilizzata. In generale, si tratta di uno strumento che si adatta a diversi contesti. In particolare, si presta ad essere utilizzato in ambito accademico in quanto permette di conoscere esattamente le fonti da cui provengono le risposte. Questa caratteristica costituisce un vantaggio per chi è interessato a valutare l’affidabilità delle applicazioni sviluppate.

Il progetto "Q&A Information Retrieval" richiede agli studenti di sviluppare e valutare un'applicazione di Q&A su argomenti specifici. Per raggiungere questo obiettivo, è essenziale utilizzare la tecnica della Retrieval Augmented Generation (RAG), che sfrutta i Large Language Model (LLM) per generare contenuti basati su dati specifici.

Sebbene i LLM siano in grado di gestire una vasta gamma di argomenti, la loro conoscenza è limitata ai dati presenti nel dominio pubblico e fino a una data specifica. Di conseguenza, per costruire applicazioni di intelligenza artificiale in grado di operare su dati provenienti da fonti private o successive alla data limite, è necessario integrare i LLM con altri componenti tipici della RAG. Questa integrazione estende la capacità dei modelli linguistici oltre i limiti dei dati di addestramento, garantendo un'applicazione più versatile e adatta alle esigenze specifiche del contesto d'uso.

Un’applicazione basata sulla RAG si fonda su un’architettura divisa in due macroaree:

- **Indicizzazione:** pipeline per l'assunzione dei dati da una fonte e la loro indicizzazione.
- **Recupero e generazione:** la catena della RAG vera e propria, che riceve la query dell'utente e recupera i dati rilevanti dall'indice, per poi passarli al modello di linguaggio naturale.

Il processo di indicizzazione, rappresentato ad alto livello in figura 2.1, si divide in più step:

- **Caricamento:** i documenti sorgente vengono caricati attraverso un document loader.
- **Suddivisione:** i documenti sorgente vengono suddivisi in porzioni più piccole, chiamate chunk, utilizzando un text splitter.
- **Trasformazione:** i chunk vengono trasformati in vettori tramite un embedding model.
- **Archiviazione:** i vettori vengono memorizzati e indicizzati in un vector store, da cui possono essere recuperati in un secondo momento.

[4] https://python.langchain.com/docs/get_started/introduction

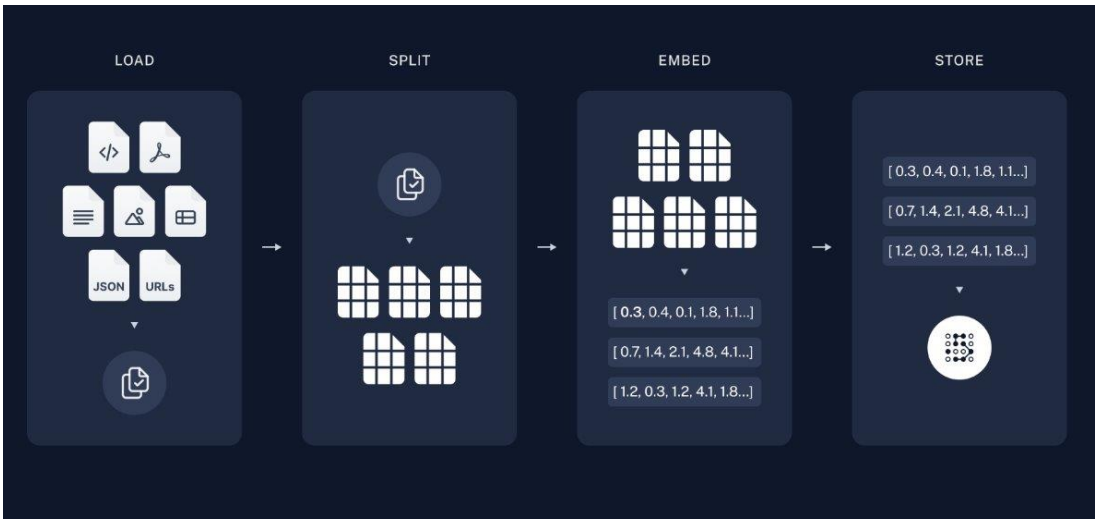


Figura 2.1 - Indicizzazione della RAG

D'altra parte, la catena che comprende recupero e generazione (figura 2.2), si divide in due parti:

- **Recupero:** un retriever ricerca tra i dati presenti nell'archivio quelli che più si avvicinano alla query dell'utente.
- **Generazione:** un LLM riceve un prompt, che comprende i dati recuperati e la domanda dell'utente, e genera una risposta.

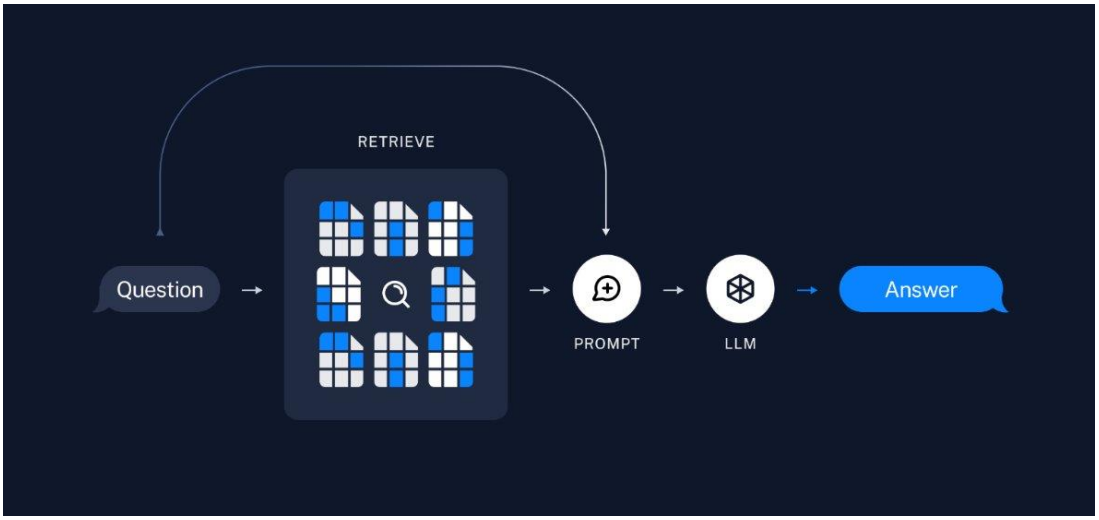


Figura 2.2 - Recupero e generazione della RAG

In sintesi, per formulare le risposte, l'applicazione seleziona tra i dati a disposizione quelli che ritiene più attinenti alla domanda dell'utente e li rielabora attraverso un modello di linguaggio naturale. Di conseguenza, un output poco soddisfacente può derivare dalla presenza di errori in una o entrambe fasi di recupero e generazione.

Nelle prossime sezioni vengono presentati i singoli componenti necessari alla creazione di un esempio di applicazione Q&A basata sulla RAG mediante il pacchetto LangChain.

2.1 Document loader

I modelli conversazionali spesso richiedono l'estrazione di dati da file (es. PDF) e la loro conversione in un formato che i LLM possano utilizzare. Nella LangChain, per caricare i file si utilizza un document loader. Esistono diversi tipi di document loader, specifici per il formato dei documenti che si intende utilizzare. Si possono caricare semplici file di testo, ma anche documenti PDF, pagine web, file CSV, codice sorgente in Python, DOCX, XLSX, PPTX, JSON o Markdown.

Qualunque sia il formato del file, ogni sua pagina viene associata ad un oggetto Document, che contiene il testo estratto nell'attributo 'page_content' e una serie di metadati nell'attributo 'metadata'. Questi ultimi sono contenuti in un dizionario nella forma 'chiave: valore' e contengono dettagli sul contenuto. Ad esempio, i metadati 'source' e 'page' rappresentano il nome e la pagina del file di origine, ma ne esistono altri come il nome dell'autore o la data di pubblicazione.

Le informazioni contenute negli oggetti Document vengono utilizzate dal LLM per generare le risposte desiderate. Spesso i Document vengono immagazzinati in un vector store per un utilizzo successivo.

2.2 Text splitter

Il text splitter riveste un ruolo cruciale nella realizzazione di un'applicazione basata sulla RAG. Come suggerisce il nome, la sua capacità è quella di suddividere un singolo oggetto Document in altri più piccoli. Questa operazione è importante in quanto permette ai componenti a valle di concentrarsi esclusivamente sulle parti più rilevanti, escludendo quelle superflue al raggiungimento dell'obiettivo dell'utente.

Infatti, molto spesso nei modelli di elaborazione linguistica che si appoggiano ad un retriever vi è la necessità di recuperare dati da una vasta collezione di documenti estesi prima di fornire una risposta. In questo contesto, lo splitter agevola la ricerca delle informazioni più importanti e favorisce la generazione di output che meglio soddisfano le richieste dell'utente.

Inoltre, il text splitter contribuisce all'archiviazione delle informazioni che costituiscono la base di conoscenza del modello linguistico. Si è anticipato che i testi vengono trasformati in vettori numerici da un modello di embedding prima di essere memorizzati. Questi modelli hanno un vincolo sulla lunghezza massima della sequenza che possono elaborare, espressa in termini di token, ovvero di unità di testo. Nel caso in cui un testo superi tale limite, esso viene automaticamente troncato per farlo rientrare nella dimensione consentita.

Il rischio di perdere informazioni importanti viene mitigato nel momento in cui si utilizza un text splitter per dividere il testo in segmenti più brevi. Così facendo, viene rispettato il vincolo sui token e ogni segmento può essere rappresentato separatamente senza che si debba rinunciare a parti rilevanti del testo originale.

Esistono diversi tipi di text splitter, ciascuno caratterizzato dalla granularità con cui viene effettuata la suddivisione del testo: alcuni si basano sui caratteri, altri tokenizzano il testo in parole, e altri ancora fanno lo stesso con le frasi. Ad ogni modo, tutti i text splitter condividono due parametri fondamentali: la lunghezza del singolo chunk e il cosiddetto overlap, cioè il numero di token che devono sovrapporsi tra due chunk successivi.

L'overlap è particolarmente utile nei casi in cui la separazione avvenga a metà parola o a metà frase. Sebbene la memorizzazione di più token possa comportare una certa ridondanza, questa caratteristica contribuisce ad evitare la perdita di informazioni durante il processo di suddivisione del testo.

2.3 Embedding model

LangChain dispone di un'interfaccia standard per l'utilizzo di embedding model messi a disposizione da soggetti come OpenAI[5] e Hugging Face[6].

Gli embedding sono rappresentazioni numeriche di dati non strutturati, come testo o immagini, in uno spazio vettoriale. Queste rappresentazioni consentono di catturare le caratteristiche significative degli oggetti in un formato che può essere compreso ed elaborato da algoritmi di machine learning.

Gli embedding sono progettati in modo tale che ad oggetti simili corrispondano vettori vicini nello spazio vettoriale. Di conseguenza, i modelli di apprendimento automatico possono comprendere le relazioni semantiche tra i dati e compiere azioni intelligenti basandosi su di queste. È chiaro che maggiore è il numero di dimensioni del vettore, tanto più la rappresentazione sarà attinente al significato dell'oggetto.

Per rendere più chiaro il concetto di rappresentazione semantica del testo si propone il seguente esempio, in cui vengono messe a confronto tre frasi:

- "La musica rende felici le persone".
- "I brani musicali portano gioia alla gente".
- "Le montagne sono alte e coperte di neve".

Nell'ottica di voler catturare la similarità semantica tra di esse, le prime due dovrebbero avere vettori molto vicini nello spazio vettoriale, e alla terza dovrebbe essere associato un vettore molto distante. Questo risultato è ottenibile solamente mediante l'utilizzo di vettori densi, cioè ad alta dimensionalità (es. > 784 dimensioni) e ricchi di informazioni per ogni dimensione.

Questi vettori catturano il significato semantico del testo che rappresentano, a differenza dei vettori cosiddetti "sparsi", che consentono di effettuare confronti solamente sulla base della sintassi. Ad esempio, si considerino queste due frasi:

- "Edoardo è corso via dal delfino verso la giraffa".
- "Edoardo è corso via dalla giraffa verso il delfino".

Se si rappresentasse ciascuna di esse mediante un vettore sparso, si otterrebbe una corrispondenza quasi perfetta, perché nonostante il significato sia diverso, le frasi sono composte dalla stessa sintassi.

Pertanto, per codificare la relazione tra le parole e il significato complessivo di un testo è necessario utilizzare vettori densi. Infatti, essi costituiscono una tecnica comunemente utilizzata in vari campi, tra cui il Natural Language Processing (NLP), in cui l'obiettivo è rendere il linguaggio umano comprensibile ai computer.

Gli embedding vengono calcolati da un embedding model, detto anche sentence transformer. Ogni embedding model si fonda su un'architettura neurale, diversa a seconda della natura del problema, della disponibilità dei dati e della complessità delle relazioni che si desidera catturare. Quando si ha a che fare col trattamento del testo, alcuni dei tipi di reti neurali più utilizzati sono i seguenti:

- **Reti neurali ricorrenti (RNN):** tra le prime architetture utilizzate per il trattamento del testo sequenziale, queste reti sono progettate per gestire dati sequenziali e mantengono uno stato interno che viene aggiornato ad ogni passaggio temporale. Le RNN sono capaci di catturare relazioni a lungo termine nelle sequenze di testo grazie alla loro capacità di mantenere una memoria temporale. Tuttavia, possono soffrire di problemi come l'incapacità di catturare relazioni a lungo raggio.
- **Reti neurali convoluzionali (CNN):** reti ampiamente utilizzate per l'elaborazione di immagini, ma possono essere adattate anche per il trattamento del testo. Applicano filtri convoluzionali, cioè matrici di pesi, su una finestra mobile di parole nel testo per estrarre caratteristiche locali.

[5] https://openai.com/

[6] https://huggingface.co/

Possono catturare pattern locali nei dati, ma possono avere difficoltà a catturare relazioni a lungo termine nei testi sequenziali.

- **Reti neurali trasformanti (Transformer):** queste reti rappresentano un'innovazione significativa nel trattamento del testo sequenziale. Si basano su meccanismi di auto-attenzione per catturare le relazioni contestuali tra le parole in una frase. Questo permette loro di catturare relazioni a lungo termine nei testi e di superare molte delle limitazioni delle RNN e delle CNN. Negli ultimi anni, i modelli Transformer (es. BERT, GPT) hanno dominato il campo del Natural Language Processing (NLP) grazie alla loro capacità di catturare relazioni complesse e a lungo termine.

La capacità dell'embedding model di rappresentare la semantica del testo svolge un ruolo cruciale nella determinazione del successo del processo di recupero, in cui il LLM invece non è coinvolto.

Nella LangChain, per ogni chunk derivante dalla funzione di split viene calcolato un vettore che riflette la semantica della porzione di testo rappresentata. Quest'informazione è molto utile quando si vogliono fare confronti tra testi, come avviene durante una ricerca semantica.

2.4 Vector store

Solitamente, gli embedding vengono inseriti all'interno di un vector store per evitare che sia necessario calcolarli nuovamente. Un vector store è un'infrastruttura progettata per memorizzare e recuperare vettori di dati non strutturati. In quanto tale, consente un accesso efficiente agli embedding pre-calcolati, ottimizzando così le prestazioni dell'applicazione.

A differenza dei database tradizionali (es. database SQL), in cui la ricerca dei dati avviene tipicamente cercando corrispondenze esatte o parziali, i vector store utilizzano il metodo della nearest neightbor search (ricerca del vicino più vicino). Praticamente, a fronte di un vettore di query si ottengono i vettori memorizzati più simili in termini di distanza euclidea o similarità del coseno.

Un database di questo tipo è particolarmente utile quando si lavora con dati talmente complessi che la ricerca di corrispondenze non è praticabile. Infatti, i vector store vengono comunemente utilizzati per trovare immagini simili, documenti di testo correlati o elementi audio con caratteristiche simili.

Pertanto, quando l'utente fa una domanda all'applicazione, il vector store calcola il vettore associato alla query e avvia il recupero di quelli più simili attraverso una ricerca semantica. Praticamente, calcola le distanze tra il vettore della query e tutti quelli memorizzati, li posiziona in ordine decrescente di similarità e restituisce ai componenti a valle i primi della lista.

2.5 Retriever

In generale, non è detto che il vector store sia utilizzato per effettuare la ricerca semantica e recuperare i chunk più significativi. Talvolta ha il solo compito di conservare gli embedding in maniera ordinata, mentre il resto viene fatto da un retriever.

Un retriever è un'interfaccia che restituisce una lista di oggetti Document a partire da una query non strutturata. In quanto tale, esso non deve essere in grado di memorizzare gli embedding, ma solo di recuperarli all'occorrenza.

Come anticipato, i vector store possono essere utilizzati come retriever, ma esistono altri tipi di retriever che sono specializzati nella ricerca semantica.

2.6 LLM

Il LLM è l'ultimo e il più importante componente della catena di RAG. Esso riceve in input un prompt costituito dagli embedding recuperati dal retriever e dalla domanda dell'utente ed elabora le porzioni di testo corrispondenti per formulare la risposta da restituire in output.

I LLM sono caratterizzati dalla capacità di elaborazione del linguaggio naturale, che acquisiscono mediante un addestramento su vasti dataset di testo provenienti da una grande varietà di fonti. Grazie ad architetture neurali complesse, come i Transformer, essi sono in grado di catturare il contesto delle parole e delle frasi, in modo tale da generare testo coerente e rilevante.

LangChain non utilizza LLM propri, ma fornisce un'interfaccia standard per interagire con una grande varietà di LLM messi a disposizione da soggetti come OpenAI e Hugging Face.

3 Esercitazioni in laboratorio

In questo capitolo verranno presentate le attività operative essenziali per lo sviluppo e la valutazione dell'applicazione di Q&A, anche detta chatbot seppur impropriamente (cfr. § 2). Al fine di facilitare la replicazione del progetto, nell'esposizione si è deciso di utilizzare il tempo presente, come se le prossime sezioni facessero parte di un manuale destinato ai docenti universitari che desiderano introdurre all'interno del proprio insegnamento un'iniziativa analoga a quella presentata in questo documento. Inoltre, dal momento che lo sviluppo avviene in linguaggio Python, gli script da utilizzare saranno riportati con opportuni commenti.

Prima di entrare nel dettaglio dei singoli step, viene fornita una panoramica sulla struttura dell'esperienza didattica, così come è stata progettata nella sua prima versione.

Il progetto è composto da 5 fasi, e per ognuna di esse è prevista un'esercitazione in laboratorio in cui viene introdotto il notebook contenente le istruzioni e il codice Python di partenza, e lasciato agli studenti del tempo per avviare il lavoro e fare eventuali domande ai docenti o ai collaboratori del corso. Le esercitazioni in laboratorio sono precedute da una prima lezione introduttiva finalizzata ad una presentazione completa del lavoro da svolgere e degli strumenti tecnici che verranno impiegati.

Di seguito sono elencate le fasi del progetto, ognuna delle quali verrà esposta in una delle sezioni successive:

- Preparazione del chatbot;
- Sviluppo della configurazione iniziale;
- Confronto tra configurazioni diverse;
- Integrazione con interfaccia Telegram;
- Valutazione finale del chatbot.

Per ognuna di esse gli studenti devono consegnare uno o più elaborati intermedi entro una data di scadenza. Viene data circa una settimana tra una consegna e l'altra, ma questo intervallo di tempo può essere liberamente modificato in base alle esigenze del corso.

3.1 Preparazione del chatbot

In questa prima fase gli studenti fanno pratica con i Jupyter notebook e iniziano a familiarizzare con i primi componenti dell'architettura RAG. Le attività previste sono le seguenti:

- Installazione delle librerie utili;
- Creazione del gruppo di lavoro;
- Impostazione del canale Telegram;
- Verifica della documentazione raccolta;
- Definizione delle domande e delle risposte di prova.

3.1.1 Installazione delle librerie utili

Quando si crea un notebook, è buona norma installare tutti i pacchetti necessari fin da subito. In questo caso, il codice è fornito dai docenti, e gli studenti devono solo eseguirlo.

Per l'installazione viene utilizzato pip, un sistema di gestione dei pacchetti scritto in Python. Pip crea una connessione con il Python Package Index, un archivio di pacchetti software presente in rete.

Vengono riportati di seguito i pacchetti utilizzati:

```
!pip install langchain  # Libreria per lo sviluppo di applicazioni basate sui LLM

!pip install unstructured     # Libreria per l'elaborazione di dati non strutturati (es. documenti testuali)

!pip install pypdf      # Libreria per la gestione di file PDF
```

3.1.2 Creazione del gruppo di lavoro

Gli studenti si organizzano in gruppi di due, nominano un responsabile per le consegne e scelgono il turno di laboratorio preferito per la loro squadra. Tutte le informazioni vengono inserite in una tabella con le seguenti colonne:

- **Cognome e Nome 1:** nominativo del responsabile.
- **Numero Matricola 1:** numero di matricola del responsabile.
- **Cognome e Nome 2:** nominativo dell'altro studente.
- **Numero Matricola 2:** numero di matricola dell'altro studente.
- **Turno laborarorio:** slot tra quelli disponibili in base all'orario del corso. Se i turni non sono bilanciati, i docenti intervengono con degli spostamenti utilizzando un metodo imparziale.

Tale tabella è presente in un foglio di calcolo condiviso nominato "GRUPPI DI LAVORO", che verrà richiamato diverse volte nelle prossime sezioni.

3.1.3 Impostazione del canale Telegram

Gli studenti devono creare su Telegram un bot non funzionante e salvare il token fornito, che sarà collegato con la LangChain in un secondo momento. Successivamente è richiesto che compilino le seguenti colonne mancanti della tabella presente nel file "GRUPPI DI LAVORO":

- **Nome del chatbot:** scelto in modo tale che sia rappresentativo dei documenti che il LLM utilizza per rispondere alle varie domande.
- **Link al chatbot:** collegamento all'URL ottenuto al momento della creazione del bot su Telegram.
- **Descrizione del chatbot:** la stessa inserita su Telegram al momento della creazione.

Il canale generato viene messo da parte e ripreso nella fase 4, quando bisognerà integrarlo al chatbot.

3.1.4 Verifica della documentazione raccolta

Il chatbot è progettato per rispondere a domande su un argomento scelto dagli studenti. Questo argomento può appartenere a qualsiasi dominio, come storia, scienza, attualità o cinema, ma si raccomanda di evitare argomenti troppo generici. Ad esempio, concentrarsi sul tema delle "centrali eoliche" può essere considerata una scelta valida, mentre "fonti di energia rinnovabile" potrebbe essere troppo generico e portare a risposte meno soddisfacenti.

Questo requisito è motivato dal fatto che la documentazione utilizzata come base di conoscenza dal LLM è limitata, e di conseguenza è più difficile ottenere risposte accurate e complete su temi molto vasti. Infatti, si richiede che gli studenti selezionino da 3 a 6 documenti, di cui almeno un PDF e una pagina web. La lingua di questi documenti può essere qualsiasi, poiché più avanti è previsto un passaggio di traduzione automatica in inglese.

Per le pagine web è sufficiente controllare che il caricamento avvenga correttamente. In questa prima fase del progetto viene fornito un codice per la verifica dei documenti, che gli studenti devono modificare sostituendo gli URL d'esempio con quelli raccolti. Il caricamento vero e proprio, invece, verrà effettuato nella fase successiva.

Si riporta il codice in questione:

```
from langchain.document_loaders import UnstructuredURLLoader

# Definisce una lista di URL da cui scaricare il contenuto d'interesse (sostituire con i propri URL)
urls = [
      "https://ourworldindata.org/artificial-intelligence",
      "https://ourworldindata.org/causes-of-death",
      "https://ourworldindata.org/terrorism"
]
# Crea un oggetto 'loader' di tipo UnstructuredURLLoader, specificando gli URL da caricare
loader = UnstructuredURLLoader(urls=urls)
# Carica il contenuto dalle URL specificate all'interno di una lista
documents = loader.load()
# Itera sui documenti ottenuti dalle URL
for doc in documents:
      # Estrae il contenuto della pagina
      page_content = doc.page_content
      # Ottiene la fonte del documento dai metadati
      source = doc.metadata["source"]
      # Estrae le prime 100 parole dal contenuto della pagina
      first_words = " ".join(page_content.split()[:100])
      # Stampa le prime 100 parole della pagina insieme alla sua fonte
      print(f"Prime 100 parole della pagina {source}:\n{first_words}")
      print("\n\n")
```

Il caricamento è andato a buon fine se per ogni pagina web vengono stampate le prime 100 parole.

Potrebbe capitare che alcune pagine non funzionino correttamente o che l'header non venga rimosso, ad esempio se è troppo esteso. In questi casi gli studenti devono scegliere altre pagine oppure cercare soluzioni alternative consultando la documentazione della LangChain[7].

Per caricare i PDF bisogna salvare i documenti su GitHub. I singoli gruppi devono creare un account e un repository raggiungibile dal link https://github.com/username/files_for_chatbot, dove l'username è quello dell'account. Al suo interno dev'essere creata una cartella chiamata "PDF", e in essa vanno inseriti i documenti selezionati.

A questo punto bisogna importare i documenti nel notebook attraverso la classe PyPDFLoader. I PDF vengono caricati una pagina per volta, e per verificare il corretto funzionamento del codice è prevista la stampa delle prime 100 parole delle prime 3 pagine. Anche in questo caso, per aiutarsi gli studenti possono consultare la documentazione della LangChain.

[7] https://python.langchain.com/docs/modules/

Si riporta l'esempio fornito dai docenti, da utilizzare come base di partenza:

```
# Copia in locale il repository (sostituire con il proprio URL)
!git clone https://github.com/username/files_for_chatbot.git
# Controlla che tutti i file siano presenti
!ls files_for_chatbot
from langchain.document_loaders import PyPDFLoader
# Importa il modulo integrato os per l'interazione con il sistema operativo
import os
# Definisce il percorso della cartella contenente i file PDF da elaborare
path_to_PDF_files = "files_for_chatbot/PDF/"
# Ottiene una lista con i nomi dei file presenti nella cartella specificata
files = os.listdir(path_to_PDF_files)
# Crea una lista vuota per contenere i documenti estratti dai file PDF
documents = list()
# Itera sulla lista dei file
for f in files:
      # Crea un oggetto 'loader' di tipo PyPDFLoader a partire dalla
      concatenazione tra il nome della cartella e quello del file corrente
      loader = PyPDFLoader(path_to_PDF_files + f)
      # Crea un oggetto Document per ogni pagina del file corrente e li
      aggiunge alla lista documents
      documents += loader.load_and_split()
# Vengono visualizzati solo 3 documenti poiché il file d'esempio è un PDF
con almeno 200 pagine e ogni pagina viene salvata come un documento
for doc in documents[:3]:
      # Estrae il contenuto della pagina
      page_content = doc.page_content
      # Ottiene il numero della pagina dai metadati
      source = doc.metadata["source"]
      # Estrae le prime 100 parole dal contenuto della pagina
      first_words = " ".join(page_content.split()[:100])
      # Stampa le prime 100 parole della pagina insieme alla sua fonte
      print(f"Prime 100 parole della pagina {source}:\n{first_words}")
      print("\n\n")
```

Ai gruppi non rimane altro che modificare il codice sostituendo l'URL del proprio repository GitHub. Dopodiché, il codice può essere utilizzato per il caricamento nella fase successiva.

3.1.5 Definizione delle domande e delle risposte di prova

L'obiettivo del progetto è quello di valutare la capacità del chatbot di fornire risposte corrette. A tal proposito, gli studenti devono definire una lista di 10 domande a cui il chatbot dovrebbe essere in grado di rispondere a partire dai documenti caricati. Inoltre, per evitare che in fase di valutazione essi siano portati a sovrastimare la prestazione del proprio modello, si richiede che essi scrivano in anticipo le risposte corrette.

Come i documenti, anche domande e risposte possono essere scritte in qualsiasi lingua, perché più avanti è presente un passaggio per la traduzione automatica del testo in inglese. In questa fase è importante fare attenzione che nei documenti selezionati siano presenti le informazioni sufficienti per rispondere alle domande, e che anche le relative risposte siano scritte considerando questo aspetto.

Domande e risposte vanno inserite in una tabella di due colonne presente in un foglio di calcolo messo a disposizione dai docenti. I gruppi devono scaricare una copia del file in formato CSV e aggiungerlo al proprio repository GitHub, sullo stesso livello della cartella "PDF".

3.1.6 Indicazioni per la consegna

Una volta completati tutti i passaggi, i gruppi devono scaricare il notebook nel formato .ipynb e inserirlo in una cartella con lo stesso nome scelto per il chatbot. Tale cartella dev'essere compressa e caricata sul portale della didattica entro la data di consegna.

3.2 Sviluppo della configurazione iniziale

La seconda fase del progetto consiste nello sviluppo di un caso completo e funzionante di LangChain su un Jupyter Notebook. L'obiettivo è che ciascun gruppo costruisca la propria applicazione a partire dai documenti e dalle domande preparati, scegliendo un text splitter, un embedding model e un LLM. Questa prima versione del chatbot rappresenta una base per quelle che verranno prodotte nella fase successiva, e verrà utilizzata come elemento di confronto e valutazione.

I gruppi devono svolgere le seguenti attività:

- Installazione delle librerie utili;
- Caricamento dei documenti;
- Suddivisione dei documenti in chunk;
- Definizione dell'embedding model;
- Creazione del vector store;
- Definizione del LLM;
- Definizione del retriever;
- Test del chatbot.

Poiché questa è la prima volta in cui gli studenti effettivamente hanno a che fare con i vari componenti della LangChain, si consiglia di esortare gli studenti a supportare lo sviluppo con i contenuti presentati nella lezione introduttiva.

3.2.1 Installazione delle librerie utili

La prima cella del notebook contiene i seguenti comandi per l'installazione di alcuni pacchetti necessari allo svolgimento delle attività appartenenti a questa fase:

```
!pip install pypdf      # Libreria per la gestione di file PDF
!pip install langchain  # Libreria per lo sviluppo di applicazioni basate sui LLM
!pip install nltk # Libreria per il processamento del linguaggio naturale
```

```
!pip install huggingface_hub  # Libreria per l'interazione con Hugging Face Hub, una piattaforma per la condivisione di modelli di machine learning open source

!pip install sentence_transformers  # Libreria per la rappresentazione vettoriale del testo

!pip install chromadb==0.4.15 # Libreria per l'archiviazione degli embedding

!pip install tiktoken   # Libreria per la conversione del testo in token

!pip install peft # Libreria per l'adattamento di grandi modelli pre-addestrati alle applicazioni a valle

!pip install googletrans==3.1.0a0   # Libreria per l'implementazione dell'API di Goodle Translate
```

I gruppi non devono fare altro che lanciare il codice per avere a disposizione sul proprio ambiente di sviluppo le classi e le funzioni richieste dai prossimi passaggi.

3.2.2 Caricamento dei documenti

In un'apposita cella del notebook, i gruppi devono inserire il codice che hanno preparato nell'attività di verifica della documentazione raccolta della fase precedente (cfr. § 3.1.4). In questo modo, tutti i documenti vengono caricati pagina per pagina all'interno del framework. Per fare ciò può essere utile consultare la documentazione della LangChain sui document loaders[8].

3.2.3 Suddivisione dei documenti in chunk

Le singole pagine che costituiscono la base di conoscenza del chatbot devono essere separate in porzioni più piccole attraverso un text splitter[9]. Il notebook contiene un esempio di text splitter personalizzato che suddivide il testo in gruppi di frasi. Questo metodo si rivela particolarmente utile quando si vogliono analizzare testi lunghi oppure lavorare in modo più gestibile con alcune parti. Essenzialmente, il testo viene suddiviso tramite una funzione che restituisce una lista di gruppi di frasi a partire dai seguenti parametri:

- **text (str):** testo da suddividere.
- **n (int):** dimensione del chunk in termini di frasi, impostata a 3 di default.
- **overlap (int):** frasi da sovrapporre tra due chunk successivi, impostato a 2 di default;
- **lang (str):** lingua in cui è scritto testo, impostata all'italiano di default.

A meno che non vengano specificati altri valori, il testo in input viene suddiviso in gruppi di 3 con un overlap di 2. Ciò significa che il chunk0 è composto da frase0, frase1 e frase2, il chunk1 da frase1, frase2 e frase3, e così via. Un overlap così elevato rispetto alla dimensione del chunk implica sicuramente che venga caricato molto più testo di quello contenuto nei documenti, ma visto che si tratta di dimensioni relativamente ridotte, ciò non dovrebbe essere troppo oneroso dal punto di vista computazionale.

Le linee seguenti servono a definire il text splitter appena descritto:

```
# Importa il pacchetto natural language toolkit per effettuare operazioni sul testo

import nltk
```

[8] https://python.langchain.com/docs/modules/data_connection/document_loaders/

[9] https://python.langchain.com/docs/modules/data_connection/document_transformers/

```
nltk.download('punkt')
# Definisce la funzione di split
def split_text_into_groups(text, n=3, overlap=2, lang="italian"):
      # Suddivide il testo in frasi sulla base della lingua specificata
      sentences = nltk.sent_tokenize(text, language=lang)
      # Inizializza una lista per i chunk
      groups = []
      # Itera sul range che va da 0 all'indice dell'ultima frase muovendosi
      di n-overlap frasi
      for i in range(0, len(sentences), n - overlap):
            # Controlla che l'indice dell'ultima frase del gruppo corrente
            non superi quello dell'ultima frase del testo
            if i + n < len(sentences):
                  # Crea il gruppo con le 3 frasi separate da uno spazio
                  group = ' '.join(sentences[i:i + n])
                  # Aggiunge il gruppo alla lista
                  groups.append(group)
      # Restituisce i gruppi
      return groups
```

Di seguito si mostra il codice per la suddivisione dei documenti tramite la funzione split_text_into_groups:

```
#Importa la classe Document
from langchain.docstore.document import Document
# Crea una lista vuota per memorizzare i documenti divisi
documents_splitted = list()
# Inizializza un contatore per generare ID univoci da associare ai
documenti
document_id = 0
# Itera sui documenti
for original_doc in documents:
      # Itera sui gruppi risultanti dalla funzione di split applicata al
      contenuto dell'oggetto Document corrente
      for group_content in
      split_text_into_groups(original_doc.page_content):
            # Copia i metadati originali del documento
            metadata = original_doc.metadata
            # Assegna un ID univoco al documento
            metadata['id'] = document_id
```

```
            # Incrementa l'ID per il prossimo documento
            document_id += 1
            # Crea un nuovo Document con il contenuto del chunk e i
            metadati aggiornati
            doc_splitted = Document(page_content=group_content,
            metadata=metadata)
            # Aggiunge il chunk alla lista
            documents_splitted.append(doc_splitted)
```

Nel passaggio successivo, i chunk vengono tradotti in inglese. Poiché gli embedding model e i LLM più utilizzati sono stati allenati prevalentemente sulla base di documenti in inglese, è più facile che l'elaborazione del testo dia risultati migliori se si utilizza questa lingua rispetto ad altre.

Le istruzioni seguenti effettuano la traduzione:

```
from googletrans import Translator
# Definisce di una funzione che utilizza googletrans per tradurre il testo
def google_translator(text, from_code="it", to_code="en"):
      translator = Translator()
      # Traduce il testo dalla lingua di origine alla lingua di
      destinazione
      translation = translator.translate(
            text,dest=to_code, src=from_code).text
      return translation
# Crea una lista vuota per memorizzare i documenti tradotti in inglese
documents_splitted_en = list()
# Itera sulla lista di chunk
for doc in documents_splitted:
      # Traduce il contenuto della pagina del documento corrente
      dall'italiano all'inglese
      doc_splitted_en = Document(
            page_content=google_translator(doc.page_content),
            metadata=doc.metadata
      )
      # Aggiunge il documento tradotto alla lista
      documents_splitted_en.append(doc_splitted_en)
```

Il risultato è la lista documents_splitted_en, che contiene i chunk in lingua inglese ed è quella che verrà passata all'embedding model.

3.2.4 Definizione dell'embedding model

Questa attività è finalizzata alla trasformazione dei chunk in vettori, necessaria affinché i loro contenuti semantici siano confrontabili in un secondo momento. Tuttavia, prima di creare il modello di embedding è necessario verificare che nessun gruppo di frasi superi la lunghezza massima consentita. Infatti,

qualsiasi porzione di testo oltre il limite viene troncata, col conseguente rischio di perdere informazioni rilevanti.

Nell'esempio seguente viene utilizzato il modello multilingua "all-mpnet-base-v2", ma gli studenti possono sceglierne uno differente dalla libreria di Hugging Face[10]. È anche possibile selezionare modelli che operano per una lingua specifica modificando i parametri di ricerca.

Si definisce la funzione get_model_max_len, che restituisce il numero massimo di token per il modello specificato:

```
# Imposta il nome dell'embedding model
embeddings_model_name = "all-mpnet-base-v2"
# Importa la classe SentenceTransformer e il modulo util per utilizzare i modelli di embedding
from sentence_transformers import SentenceTransformer, util
# Definisce la funzione per ottenere la lunghezza massima consentita dal modello
def get_model_max_len(model_name=embeddings_model_name):
    # Carica il modello specificato
    model = SentenceTransformer('sentence-transformers/' + model_name)
    # Accede al tokenizer dal modello SentenceTransformer
    tokenizer = model.tokenizer
    # Ottiene la lunghezza massima consentita dal modello
    max_len = tokenizer.model_max_length
    return max_len
```

Viene ora definita un'altra funzione, che determina la lunghezza di un testo in termini di token. Il numero di token viene determinato dal tokenizer associato al modello che si intende utilizzare:

```
# Importa la classe AutoTokenizer per i tokenizer automatici
from transformers import AutoTokenizer
# Definisce la funzione per misurare la lunghezza del testo
def count_text_token(text, model_name=embeddings_model_name):
    # Carica il tokenizer adatto al modello specificato
    tokenizer = AutoTokenizer.from_pretrained('sentence-transformers/' + model_name)
    # Tokenizza il testo
    tokens = tokenizer(text)
    # Conta il numero di token
    num_tokens = len(tokens['input_ids'])
    return num_tokens
```

[10] https://huggingface.co/models?library=sentence-transformers

L'informazione ricavata dalla variabile num_tokens è cruciale per effettuare il confronto con il vincolo caratteristico del modello. Qualora un chunk non dovesse rientrare nella lunghezza massima, è necessario segmentarlo ulteriormente, oppure scegliere un embedding model diverso.

Adesso che si hanno queste due funzioni è possibile controllare che ciascun chunk della lista documents_splitted rispetti il vincolo sul numero di token:

```
# Ottiene la lunghezza massima consentita dal modello di embedding
max_len_model = get_model_max_len()
# Itera sulla lista di chunk
for doc in documents_splitted:
      # Estrae il contenuto dall'oggetto Document
      text = doc.page_content
      # Verifica se il numero di token nel testo supera la lunghezza massima
      if count_text_token(text) > max_len_model:
            # Stampa un messaggio di errore
            print("Il testo supera la lunghezza massima del modello di embedding")
```

A questo punto si può procedere con la creazione del modello di embedding attraverso i comandi seguenti:

```
from langchain.embeddings import HuggingFaceEmbeddings
# Crea l'embedding model
embedding_model = HuggingFaceEmbeddings(
      model_name=embeddings_model_name,    # Nome del modello
      model_kwargs={"device": "cpu"},      # Argomenti chiave da passare al modello
      encode_kwargs={'normalize_embeddings': True}    # Argomenti chiave da passare al metodo encode del modello
)
```

3.2.5 Creazione del vector store

I vettori risultanti dal processo di embedding vengono archiviati all'interno di un vector store. Nell'ambito di questo progetto viene utilizzato il sistema di archiviazione vettoriale Chroma[11], che in fase di interrogazione restituisce i 4 embedding più simili a quello della query secondo il criterio della similarità del coseno:

```
from langchain.vectorstores import Chroma
# Genera il vector store Chroma
db = Chroma.from_documents(
      documents_splitted_en,  # Lista dei chunk in inglese
```

[11] https://docs.trychroma.com/

```
        embedding_model,  # Modello definito precedentemente
        collection_metadata={"hnsw:space": "cosine"}    # Metadati per la
        raccolta dei vettori mediante la similarità del coseno
)
```

3.2.6 Definizione del LLM

Per elaborare una risposta a partire dai vettori recuperati da Chroma è necessario un LLM. Nell'esempio fornito dai docenti viene utilizzato il modello "declare-lab/flan-alpaca-large", ma gli studenti possono sceglierne altri in alternativa dalla libreria di Hugging Face[12].

Di seguito il codice di riferimento:

```
# Crea una directory su Colab per memorizzare i file del modello
(sostituire con il nome del modello scelto)
!mkdir flan-alpaca-large
from huggingface_hub import snapshot_download
# Scarica il modello da Hugging Face Model Hub e lo memorizza nella
directory
snapshot_download(
        # Specifica il repository da cui si vogliono scaricare i file del
        modello (sostituire con quello del modello scelto)
        repo_id="declare-lab/flan-alpaca-large",
        # Specifica la cartella in cui inserire i file scaricati (sostituire
        con il nome del modello scelto)
        local_dir='./flan-alpaca-large/',
        # Specifica che non si vogliono usare i symlink per la gestione dei
        file locali
        local_dir_use_symlinks=False,
        # Specifica il token di autenticazione da utilizzare per il download
        token="hf_..."
)
# Importa la classe HuggingFacePipeline per eseguire in locale i modelli di
Hugging Face
from langchain.llms import HuggingFacePipeline
# Specifica il percorso alla directory contenente i file del modello
(sostituire con il nome del modello scelto)
model_dir = "./flan-alpaca-large/"
# Inizializza una pipeline per l'uso del modello scaricato
llm = HuggingFacePipeline.from_model_id(
```

[12] https://huggingface.co/models?pipeline_tag=text2text-generation

```
        model_id=model_dir,
        task='text2text-generation',
        model_kwargs={"temperature": 0.60, "min_length": 35, "max_length":
        500, "repetition_penalty": 5.0}
)
```

Nella definizione del modello, model_kwargs è un dizionario di argomenti chiave aggiuntivi. Vengono spiegati di seguito quelli menzionati:

- **temperature:** indicatore della creatività del testo generato. Valori più alti di temperatura rendono il testo più casuale e creativo, mentre valori più bassi producono testo più deterministico. Una temperatura di 1.0 (valore predefinito) produce testo moderatamente casuale.
- **min_length:** lunghezza minima del testo generato. Assicura che il testo generato abbia almeno una certa lunghezza per soddisfare i requisiti del contesto o dell'applicazione.
- **max_length:** lunghezza massima del testo generato. Limita la lunghezza del testo generato per evitare output eccessivamente lunghi e poco pratici.
- **repetition_penalty:** penalità per la ripetizione di parole o frasi nel testo generato. Valori più alti riducono la probabilità di ripetizione nel testo generato, incoraggiando una maggiore diversità e coerenza nel testo.

In sintesi, questi parametri consentono di regolare in modo più preciso il comportamento del modello durante la generazione del testo, influenzando aspetti come la creatività, la lunghezza e la coerenza del testo generato. Modificando questi parametri, è possibile adattare la generazione del testo alle specifiche esigenze dell'applicazione.

3.2.7 Definizione del retriever

Il codice seguente crea un oggetto di tipo RetrievalQA a partire dal LLM e dal vector store precedentemente configurati. La classe RetrievalQA rappresenta oggetti in grado di rispondere alle domande dell'utente. Tali oggetti si basano su un retriever che recupera i documenti con le informazioni utili e un LLM che rielabora le fonti e genera il testo.

Di seguito sono mostrate le istruzioni necessarie:

```
from langchain.chains import RetrievalQA
from langchain.llms import OpenAI
# Crea un oggetto RetrievalQA
qa = RetrievalQA.from_chain_type(
        llm=llm,    # Modello dfinito precedentemente
        chain_type="stuff",     # Tipo di catena da utilizzare (può variare
        in base all'applicazione)
        retriever=db.as_retriever(),  # Vector store definito precedentemente
        return_source_documents=True, # Opzione per restituire i documenti
        sorgente
        verbose=False     # Disattiva la modalità verbosa (output
        dettagliato)
)
```

3.2.8 Test del chatbot

A questo punto l'applicazione è completa e può essere provata. A tal proposito viene definita la seguente funzione, che a partire dal testo di una domanda produce come risultato un dizionario con le seguenti chiavi:

- **query:** domanda dell'utente.
- **result:** risposta del LLM.
- **source_documents:** lista dei 4 oggetti Document corrispondenti ai chunk recuperati dal retriever.

Sono mostrate di seguito le linee di codice per ottenere il risultato del chatbot:

```
import warnings

# Definisce una funzione per ottenere la risposta della LangChain a partire dalla domanda dell'utente

def get_chat_response(text, qa=qa, lang='it'):
      # Disabilita i warning all'interno della funzione
      with warnings.catch_warnings():
            warnings.filterwarnings("ignore")
            if lang == "it":
                  # Traduce il testo in inglese utilizzando Google_Translator
                  text_en = google_translator(text)
                  # Esegue una query usando la pipeline di RetrievalQA (qa) sul testo in inglese
                  res = qa(text_en)
                  res["query"] = text
                  # Traduce la risposta in italiano
                  res["result"] = google_translator(res["result"], from_code="en", to_code="it")
            else:
                  # Esegue una query usando la pipeline di RetrievalQA (qa) sul testo in lingua originale
                  res = qa(text)
      return res
```

Nel codice seguente viene definita una domanda ed evocata la funzione precedente:

```
# Definisce una query
query = "Cos'è il principio di Pareto?"
print("Query:", query)
# Ottiene la risposta utilizzando la funzione get_chat_response
res = get_chat_response(query)
# Estrae il testo di risposta
response_text = res['result']
```

```
print("Risposta:", response_text)
```

Oltre alla risposta, si possono anche ricavare i singoli chunk da cui essa proviene. Adesso l'informazione non è rilevante, ma questa funzionalità verrà ampiamente utilizzata nel prossimo capitolo per fare considerazioni sull'affidabilità del retriever (cfr. § 4.2).

Nell'esempio precedente si utilizza una singola query, ma gli studenti devono modificare il codice in modo tale che il chatbot risponda a tutte le 10 domande. Inoltre, le risposte vanno salvate in un file CSV da creare con la libreria pandas[13]:

```
# Crea un DataFrame vuoto

df1 = pd.DataFrame()

# Popola la prima colonna con i testi delle 10 domande (precedentemente
salvate in opportune variabili)

df1['domande'] = [query1, query2, query3, query4, query5, query6, query7,
query8, query9, query10]

# Popola la seconda colonna con i testi delle 10 risposte (precedentemente
salvate in opportune variabili)

df1['risposte'] = [response_text1, response_text2, response_text3,
response_text4, response_text5, response_text6, response_text7,
response_text8, response_text9, response_text10]

# Imposta il nome del file CSV

nome_file = 'domande_risposte_chatbot.csv'

# Trasforma il DataFrame in un file CSV con quel nome

df1.to_csv(nome_file, index=False)

# Stampa il contenuto del file CSV in un dataframe

pd.read_csv('/content/domande_risposte_chatbot.csv')
```

Si prevede che quest'ultimo frammento di codice non sia messo a disposizione degli studenti. I gruppi devono trovare una soluzione in autonomia a partire dalle istruzioni per creare un dataframe con pandas.

3.2.9 Indicazioni per la consegna

Alla fine di questa fase gli studenti devono inserire il notebook funzionante in formato .ipynb e il CSV con le domande e le risposte in una cartella con il nome del chatbot, che va compressa e caricata sul portale della didattica.

3.3 Confronto tra configurazioni diverse

La terza fase del progetto prevede che gli studenti cambino alcune variabili del chatbot e confrontino i risultati delle diverse configurazioni. Nello specifico, si prevede che essi provino 3 embedding model diversi e 3 LLM diversi, per un totale di 9 configurazioni. Ognuna di queste dev'essere valutata sulla base del contenuto prodotto in risposta alle 10 domande definite durante la preparazione del chatbot (cfr. § 3.1.5), e alla fine ogni gruppo deve identificare la combinazione ottima di embedding model e LLM secondo un proprio criterio di selezione. Quest'ultimo può tenere conto sia dell'accuratezza del

[13] https://pandas.pydata.org/

retriever nel recuperare informazioni rilevanti che della qualità del risultato del LLM in termini di coerenza semantica e correttezza linguistica.

Le attività previste in questa fase sono le seguenti:

- Valutazione delle configurazioni;
- Verifica del file CSV;
- Questionario su sfide tecniche e soddisfazione.

3.3.1 Valutazione delle configurazioni

Le squadre devono partire dal codice assemblato nella seconda fase del progetto per testare due embedding model e due LLM alternativi a quelli originali. Successivamente si prevede che ogni risposta ottenuta venga valutata sotto i seguenti aspetti:

- **Tempo di risposta:** indica il tempo macchina richiesto per generare il risultato a partire dall'istante in cui viene evocata la funzione get_chat_response (cfr. § 3.2.8).
- **Efficacia del retriever:** indica se l'applicazione recupera con successo quelle porzioni di testo che contengono le informazioni utili per rispondere alla domanda.
- **Efficacia semantica del LLM:** indica se la risposta prodotta dal LLM ha una sua coerenza interna, senza essere troppo dispersiva o ridondante.
- **Efficacia linguistica del LLM:** indica se la risposta prodotta dal LLM è corretta grammaticalmente, cioè priva di errori di sintassi o grammatica.

I gruppi devono scrivere uno script che restituisca un dataframe pandas con le risposte delle 9 configurazioni alle 10 domande, e partire dai risultati per effettuare le valutazioni. Una volta valutate tutte le 9 configurazioni, i gruppi devono comunicare ai docenti qual è la migliore secondo loro, specificando embedding model, LLM e criterio di selezione adoperato.

Si raccomanda di specificare agli studenti che in questo stadio non è tanto importante se il chatbot soddisfi o meno le aspettative in termini di prestazioni. Ciò che conta è giudicare in modo rigoroso e obiettivo la qualità dei risultati ottenuti in quanto, come detto fin dall'inizio, è proprio questo spirito critico l'oggetto principale della valutazione dei docenti.

3.3.2 Verifica del file CSV

Ad ognuna delle 9 configurazioni vengono sottoposte le 10 domande di prova, e per ognuno dei 90 risultati vanno assegnati dei valori a delle metriche di valutazione. Tutti i dati risultanti da questa operazione devono essere raccolti in un file CSV nominato "performance_evaluation.csv" con le colonne seguenti:

- **Domanda:** testo della domanda di prova.
- **Risposta:** testo della risposta generata.
- **embedding_model:** nome del modello utilizzato per la creazione degli embedding.
- **llm_model:** nome del modello utilizzato per l'elaborazione delle fonti e la formulazione della risposta.
- **response_time:** tempo di risposta in millisecondi.
- **retrieval_effectivness:** valutazione binaria sull'efficacia del retriever, pari ad "1" se si ritiene che la risposta contenga informazioni affidabili e a "0" altrimenti.
- **llm_semantic_effectivness:** valutazione binaria sull'efficacia del LLM dal punto di vista semantico, pari ad "1" se si ritiene che la risposta abbia un senso logico e a "0" altrimenti.
- **llm_syntax_grammar_effectivness:** valutazione binaria sull'efficacia del LLM dal punto di vista linguistico, pari ad "1" se si ritiene che la risposta sia grammaticalmente e sintatticamente corretta e a "0" altrimenti.

I gruppi possono verificare la correttezza del formato di questo file utilizzando il seguente codice presente all'interno del notebook, che definisce la funzione test_your_csv:

```
# Importa il pacchetto numpy per il calcolo scientifico

import numpy as np
```

```
# Importa il pacchetto pandas per la costruzione di strutture dati relazionali
import pandas as pd
# Importa il modulo integrato os per l'interazione con il sistema operativo
import os
# Definisce la funzione per verificare la struttura del file CSV presente al percorso 'csv_path'
def test_your_csv(csv_path: str):
	# Controlla che il percorso esista
	if not os.path.exists(csv_path):
		raise FileNotFoundError(f"file path {csv_path} does not exists")
	# Controlla che il file sia un CSV
	if not os.path.isfile(csv_path) or not csv_path.endswith('.csv'):
		raise FileNotFoundError(f"file path {csv_path} is not a csv file")
	# Crea un DataFrame con lo stesso contenuto del CSV
	df = pd.read_csv(csv_path)
	# Crea una lista con i nomi delle colonne previste
	exp_col_names = ["domanda", "risposta", "embedding_model", "llm_model", "response_time", "retrieval_effectivness", "llm_semantic_effectivness", "llm_syntax_grammar_effectivness"]
	# Recupera i nomi delle colonne del dataframe
	df_columns = df.columns.tolist()
	# Controlla che il numero delle colonne sia corretto
	if len(df_columns) != len(exp_col_names):
		print("Warning: Number of columns is incorrect")
		return False
	else:
		print("Correct number of columns")
	# Controlla che i nomi delle colonne siano quelli corretti
	if any(df_columns[i] != exp_col_names[i] for i in range(len(df_columns))):
		print("Warning: Columns name does not match the correct names")
		print(df_columns)
		print(exp_col_names)
		return False
	else:
```

```
        print("Correct columns names")
    # Crea una lista con i tipi di dato previsti per le colonne
    exp_dtypes = [object, object, object, object, "int64", "int64",
    "int64", "int64"]
    # Ottiene i tipi di dato di ciascuna colonna del DataFrame
    df_dtypes = df.dtypes
    # Controlla che i tipi di dato siano quelli corretti
    if any(df[exp_col_names[i]].dtypes != exp_dtypes[i] for i in
    range(len(exp_dtypes[:6]))):
        print("Warning: Columns dtype does not match the correct dtypes
        \n")
        print(df_dtypes[-1])
        print(exp_dtypes)
        return False
    else:
        print("Correct columns dtypes")
    # Controlla che le ultime 3 colonne siano binarie
    for name in exp_col_names[-3:]:
        if any(x not in {0,1} for x in df[name]):
            print(f"Warning: Column {name} contains non binary
            values")
            return False
    print("Binary columns are correct")
    return True
```

3.3.3 Questionario su sfide tecniche e soddisfazione

Dopo aver valutato e confrontato le varie configurazioni, si prevede che gli studenti rispondano ad un questionario sull'esperienza, le cui domande e i relativi obiettivi sono riportati in Tabella 3.1tabella 3.1. Le risposte possono essere utili ai docenti per migliorare in futuro l'esperienza degli studenti.

Indice	Domanda	Obiettivo
Q1	Quali sono le difficoltà maggiori che hai riscontrato nella creazione del chatbot?	Comprendere le attività più critiche nel processo di sviluppo del chatbot ed adattare eventuali interventi didattici per affrontare tali ostacoli.
Q2	Quali sono i principali problemi riscontrati nella fase di caricamento dei documenti, e come li hai risolti?	Esplorare le problematiche più comuni emerse durante l'importazione dei documenti all'interno del framework e fornire indicazioni mirate sulle strategie di risoluzione.
Q3	Quali sono i principali problemi riscontrati nella fase di splitting dei documenti, e come li hai risolti?	Valutare la capacità degli studenti di gestire la segmentazione dei testi e identificare eventuali lacune concettuali da affrontate attraverso approfondimenti durante le lezioni.

Q4	I documenti in che lingua erano? Hai dovuto tradurli in inglese? Se sì, come hai eseguito la traduzione? E perché li hai tradotti?	Raccogliere informazioni sulla lingua utilizzata dagli studenti e sulle decisioni prese in merito alla traduzione del testo, per valutare la loro consapevolezza delle implicazioni linguistiche nel contesto del progetto.
Q5	Quali embedding model hai testato? Spiega la logica con cui hai scelto il migliore.	Valutare la comprensione riguardo alle caratteristiche dei modelli di embedding presenti su Hugging Face e promuovere una riflessione critica sulle strategie di selezione di quelli più adatti per il progetto.
Q6	Quali LLM hai testato? Spiega la logica con cui hai scelto il migliore.	Valutare la familiarità degli studenti con i LLM e la loro capacità di selezionare da Hugging Face i più appropriati per il proprio chatbot.
Q7	Che risultati ti aspettavi di ottenere all'inizio dei laboratori?	Farsi un'idea delle aspettative degli studenti per valutare se nel complesso sono riusciti ad apprezzare questa esperienza innovativa.
Q8	Che risultati avete effettivamente ottenuto?	Valutare l'efficacia del processo di apprendimento e identificare eventuali discrepanze rispetto alle aspettative, nell'ottica di comprendere il livello di soddisfazione raggiunto.
Q9	Pensi che il tuo chatbot potrebbe essere realmente utilizzato? Le risposte sono affidabili?	Valutare la percezione degli studenti riguardo all'utilità pratica del chatbot e stimolare una riflessione critica sulla sua affidabilità.
Q10	Aggiungete ulteriori commenti e curiosità sul lavoro svolto, oltre che a dubbi e critiche sul design dei laboratori sul chatbot.	Fornire agli studenti l'opportunità di condividere ulteriori feedback sul lavoro svolto e raccogliere informazioni aggiuntive per migliorare le future edizioni del progetto.

Tabella 3.1 - Questionario su sfide tecniche e soddisfazione

Integrando queste domande nel notebook, i docenti possono ricavare informazioni sull'impressione dei vari gruppi riguardo all'intera esperienza. Tali informazioni possono essere utilizzate per adattare il livello di difficoltà dell'esercitazione alla preparazione degli studenti e apportare migliorie.

3.3.4 Indicazioni per la consegna

Una volta completate le attività comprese in questa fase, ogni gruppo deve scaricare la propria versione del notebook (.ipynb), che deve contenere anche le risposte al questionario, e il file performance_evaluation.csv. Entrambi vanno inseriti in una cartella con il nome del chatbot, da comprimere e caricare sul portale della didattica.

3.4 Integrazione con interfaccia Telegram

A valle dello sviluppo e del confronto delle varie configurazioni, si prevede che ogni gruppo integri quella migliore nel canale Telegram impostato all'inizio del progetto. L'obiettivo di questa fase è quello di rendere il chatbot fruibile non solo da un ambiente di programmazione come un Jupyter notebook, ma da un'interfaccia user-friendly che permette un'interazione in tempo reale.

3.4.1 Collegamento al canale Telegram

In questo caso agli studenti viene solo fornito uno spunto per l'implementazione. Di seguito si riporta un esempio di script che può essere utilizzato come base da cui partire:

```
# Importa librerie e classi

pip install Python-telegram-bot==13.13

from telegram.ext import Updater, MessageHandler, CommandHandler, Filters

# Definisce il token (sostituire con quello ricevuto al momento della
configurazione del canale Telegram)

TOKEN = "token"

# Definisce la funzione che ritorna la risposta alla domanda (sostituire
questa con la propria versione)

def get_chat_response(text):

      return "Risposta a: " + text  # Esempio di risposta statica

# Definisce la funzione per mostrare la risposta all'interno della chat

def reply_to_question(update, context):

      # Ottiene il testo del messaggio dell'utente

      user_text = update.message.text

      # Chiede la risposta al modello

      bot_response = get_chat_response(user_text)

      # Invia la risposta all'utente

      update.message.reply_text(bot_response)    # bot_response["result"]

# Definisce la funzione per gestire i comandi (personalizzare secondo le
proprie esigenze)

def start(update, context):

      update.message.reply_text("Ciao! Sono il tuo bot. Fai domande e
      risponderò!")

def main():

      # Crea un oggetto updater e passa il token del tuo bot

      updater = Updater(TOKEN, use_context=True)

      # Ottiene il dispatcher per registrare i gestori di eventi

      dp = updater.dispatcher

      # Aggiunge un gestore per rispondere alle domande degli utenti

      dp.add_handler(MessageHandler(Filters.text & ~Filters.command,
      reply_to_question))

      # Aggiunge un gestore per il comando /start

      dp.add_handler(CommandHandler("start", start))

      # Avvia il bot

      updater.start_polling()
```

```
    # Attende che il bot sia arrestato manualmente (es. premendo Ctrl+C)
    updater.idle()
if __name__ == '__main__':
    main()
```

Una volta sostituito il token e la definizione della funzione get_chat_response, questo codice crea un bot Telegram che risponde alle domande dell'utente. La sessione viene avviata quando l'utente seleziona il comando "/start".

3.4.2 Indicazioni per la consegna

A valle di questa fase, i link presenti nella colonna "Link al chatbot" del file "GRUPPI DI LAVORO" dovrebbero rimandare a canali Telegram funzionanti, in quando completamente collegati con l'applicazione sviluppata. Inoltre, i gruppi devono compilare la colonna "Link al notebook con il chatbot completo" con il link al notebook di questa fase.

3.5 Valutazione finale del chatbot

In quest'ultima fase del progetto, si prevede che gli studenti confrontino le risposte del proprio chatbot con quelle di ChatGPT.

3.5.1 Confronto con ChatGPT

I gruppi devono sottoporre a ChatGPT le 10 domande di prova e confrontare i risultati con le risposte della propria configurazione migliore. A tal fine, è richiesto che sia popolata una tabella con queste colonne:

- **Indice domanda:** indice numerico progressivo (1-10);
- **Domanda:** testo della domanda di prova;
- **Risposta iniziale:** risposta definita in fase di progettazione;
- **Risposta del chatbot Telegram:** output del chatbot;
- **Risposta di ChatGPT:** output di ChatGPT;
- **Valutazione:** giudizio numerico pari a "0" se non si notano differenze significative tra le due risposte, "1" se la risposta del proprio chatbot è migliore di quella di ChatGPT, "2" se quella di ChatGPT è migliore di quella del chatbot.

Infine, si richiede che i gruppi rispondano alla seguente domanda presente al fondo del notebook: "Sulla base della valutazione svolta, il tuo chatbot restituisce risposte migliori o peggiori di ChatGPT?".

I risultati del confronto possono essere utilizzati per fare considerazioni sulle prestazioni delle applicazioni realizzabili con la LangChain e valutarne l'efficacia rispetto ad un modello di intelligenza artificiale generativa ampiamente utilizzato.

3.5.2 Indicazioni per la consegna

Il notebook di questa fase va caricato in formato .ipynb sul portale della didattica all'interno di una cartella compressa con il nome del chatbot.

4 Analisi dei risultati

In questo capitolo verranno presentate alcune analisi dei dati raccolti durante la prima edizione del progetto "Q&A Information Retrieval", incentrato sulla creazione e sulla valutazione di un'applicazione

di Q&A supportata da RAG, cioè in grado di rispondere a domande su un argomento specifico a partire da una serie di documenti.

Le analisi includono diversi aspetti, dalle prestazioni dei chatbot sviluppati al feedback degli studenti riguardo all'esperienza didattica. In una prima sezione del capitolo verranno analizzati i risultati della progettazione del chatbot. Dopodiché saranno presentati i risultati riguardo alla capacità di recuperare informazioni rilevanti dai documenti a disposizione. A seguire ci sarà un'analisi delle risposte degli studenti al questionario su sfide tecniche e livello di soddisfazione. Infine, verrà descritto l'andamento complessivo dell'esercitazione dal punto di vista dei docenti.

4.1 Progettazione dei chatbot

In questa prima sezione vengono presentati e discussi i risultati delle analisi sui dati relativi alla progettazione del chatbot. In particolare, è fornita una panoramica sugli argomenti selezionati e vengono fatte diverse considerazioni sui modelli testati e quelli scelti per la configurazione migliore.

4.1.1 Classificazione degli argomenti

Il progetto prevede che i gruppi siano liberi di scegliere l'argomento sul quale il proprio chatbot deve saper rispondere, nonché le 10 domande di prova da sottoporre per misurarne le prestazioni. In questa attività la decisione è demandata alla creatività degli studenti e l'unico vincolo è che l'argomento sia sufficientemente specifico da garantire, potenzialmente, risultati migliori rispetto ad un modello generalista come ChatGPT.

La tabella 4.1 riporta i 30 gruppi che hanno partecipato al progetto, identificati da un codice univoco, e i relativi argomenti scelti.

ID	Argomento
G01	Andamento della qualità dell'aria a Torino
G02	Scelta del cane da guardia
G03	Storia, ricette e valori nutrizionali dei dolci natalizi
G04	Vita, stile e opere della scrittrice inglese Jane Austen
G05	Prodotti IKEA per l'arredamento dedicato al gaming
G06	Ricette del noto chef Cracco
G07	Premio Oscar 2022 per il migliore attore protagonista
G08	L'Inferno della Divina Commedia
G09	Storia del cinema d'animazione
G10	Viaggi a Parigi
G11	Discografia del cantante Caparezza.
G12	Musei di Torino
G13	Corso di Basi di dati del Politecnico di Torino
G14	Mete tra le montagne piemontesi per svolgere attività
G15	Ristoranti con una stella Michelin a Torino.
G16	Ricette vegane salate
G17	Vita degli squali
G18	Centrocampisti del Real Madrid 2023/2024
G19	Protagoniste del programma tv RuPaul's Drag Race
G20	Centro Sperimentale di Cinematografia
G21	Sintomi, cause e rimedi alle allergie ai gatti
G22	Itinerari per il cammino in Italia
G23	Film American Psycho
G24	Botteghino americano del 2019
G25	Opere e stile dell'artista Banksy
G26	Il mondo dei ninja

G27	Film proiettati e servizi offerti in uno specifico cinema
G28	Il gioco League of Legends e il torneo mondiale 2023
G29	Piante da appartamento e come prendersene cura
G30	Storia della mozzarella di bufala campana DOP

Tabella 4.1 - Campione di 30 gruppi

Tali argomenti sono stati raggruppati nelle seguenti categorie:

- **Intrattenimento:** in questa categoria sono inclusi argomenti legati al cinema, alla televisione e ai videogiochi. I chatbot forniscono informazioni su film, programmi televisivi, videogiochi e altri aspetti dell'intrattenimento moderno.
- **Sport:** questa categoria comprende argomenti legati a varie attività sportive, come calcio e arrampicata. I chatbot forniscono informazioni su squadre e atleti oppure consigli sull'attività fisica.
- **Salute:** in questa categoria sono presenti chatbot che forniscono informazioni su patologie specifiche e relativi sintomi, cause e rimedi, nonché suggerimenti per il miglioramento della salute.
- **Arredamento:** Questa categoria si concentra sull'arredamento degli spazi abitativi e lavorativi. Include chatbot che forniscono consigli su come arredare al meglio la casa o l'ufficio.
- **Animali:** comprende chatbot che rispondono a domande sugli animali e forniscono informazioni e consigli relativi agli animali domestici e alla loro cura.
- **Cultura:** questa categoria comprende argomenti legati alla storia, alla letteratura, all'arte e alla musica. I chatbot rispondono a domande su scrittori, artisti, opere d'arte e molto altro.
- **Cucina:** comprende i chatbot che forniscono ricette, consigli culinari e informazioni sui valori nutrizionali degli alimenti.
- **Turismo:** questa categoria contiene chatbot per informazioni su destinazioni turistiche, attrazioni, luoghi da visitare e consigli per viaggiare in modo sicuro e divertente.

I risultati sono riportati nel diagramma a torta della figura 4.1.

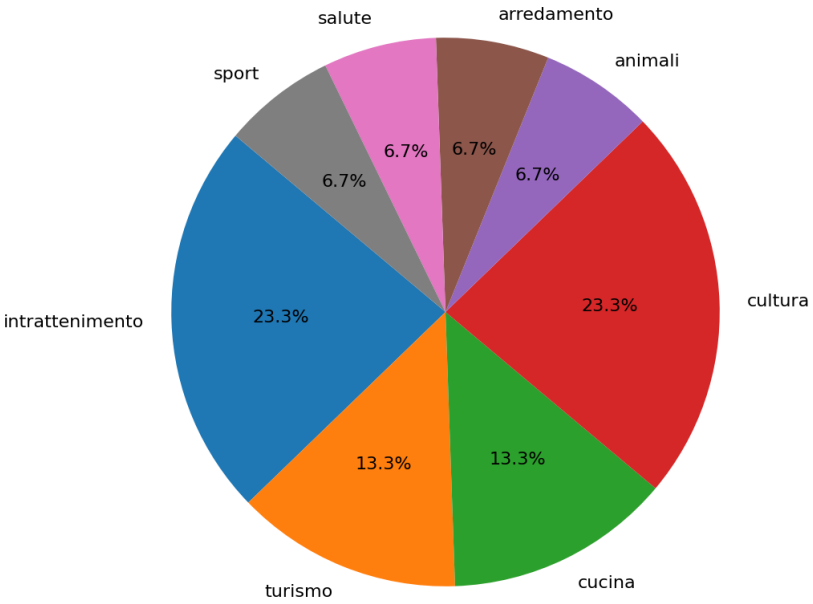


Figura 4.1 - Distribuzione degli argomenti scelti per il chatbot

Si può osservare che non c'è stata una categoria con la maggioranza assoluta. La maggioranza relativa (23.3%) è stata raggiunta dal dominio dell'intrattenimento (es. candidati all'Oscar per il migliore attore protagonista del 2022) e da quello della cultura (es. opere e stile dell'artista Banksy).

Al secondo posto si trovano turismo (es. musei di Torino) e cucina (es. storia, ricette e valori nutrizionali dei dolci natalizi) con il 13.3%. Infine, a pari merito con il 6.7% ci sono lo sport (es. Centrocampisti titolari del Real Madrid 2023/2024), la salute (es. Sintomi, cause e rimedi alle allergie ai gatti), l'arredamento (es. Piante da appartamento e come prendersene cura) e gli animali (es. Vita degli squali).

Lasciare questa scelta nelle mani degli studenti ha permesso l'attivazione di un approccio creativo, favorendo una motivazione maggiore all'interno della classe. Sembra che essi abbiano cercato di creare qualcosa di molto originale, rispettando il vincolo sulla specificità dell'argomento. Inoltre, hanno dimostrando di aver apprezzato l'attività per il suo margine di creatività.

4.1.2 Modelli più popolari

Tra la seconda e la terza fase del progetto, si prevede che gli studenti provino 3 modelli di embedding e 3 LLM presenti nel repository di Hugging Face, da cui potevano consultare la scheda del singolo modello con caratteristiche tecniche e statistiche di utilizzo. In questa sezione si illustrano i modelli più testati. Per quest'analisi i gruppi G06, G07, G18, G26 e G29 sono stati esclusi poiché non sono riusciti a concludere l'attività. Quindi il campione si riduce a 25 gruppi.

Sono stati provati 21 embedding model diversi, ma solo 15 da almeno 2 gruppi. La figura 4.2 mostra un grafico a barre che ha sulle ordinate questi ultimi e sulle ascisse il numero di gruppi che hanno provato il singolo modello.

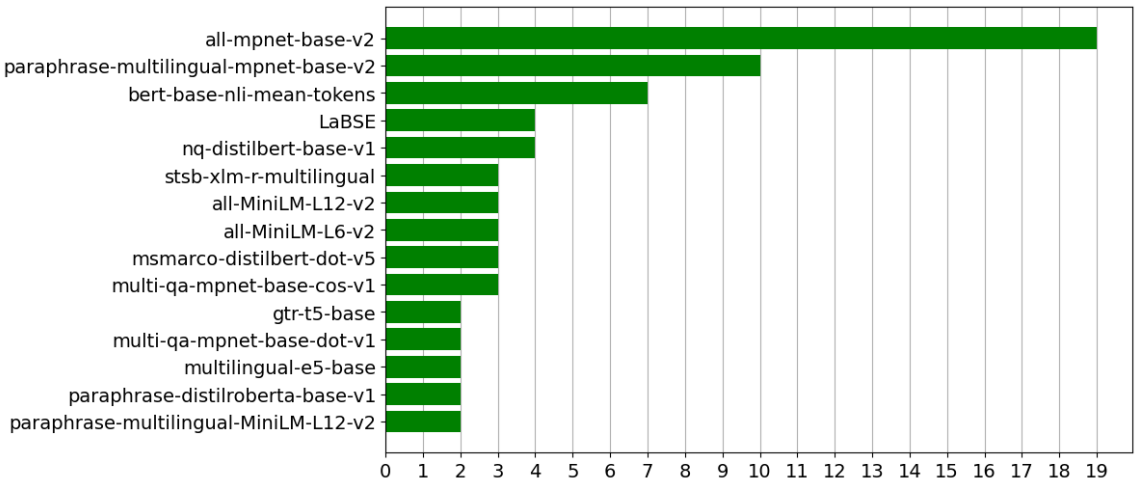


Figura 4.2 - Embedding model provati da almeno 2 gruppi

Si osserva che i modelli più popolari sono "all-mpnet-base-v2", testato da 19 gruppi (76%), "paraphrase-multilingual-mpnet-base-v2", testato da 10 gruppi (40%), e "bert-base-nli-mean-tokens", testato da 7 gruppi (28%), ad indicare una tendenza verso l'adozione di embedding avanzati e multilingua. D'altra parte, la presenza di modelli più leggeri come "all-MiniLM-L12-v2" e "all-MiniLM-L6-v2" suggerisce un interesse nell'efficienza computazionale senza che vengano compromesse le prestazioni. Allo stesso tempo, modelli specializzati come "msmarco-distilbert-dot-v5" e "nq-distilbert-base-v1" evidenziano la ricerca di soluzioni ottimizzate per task specifici. Altri modelli sono stati scelti da un solo gruppo e quindi si pensa che non meritino particolare attenzione. In sintesi, la diversità dei risultati dimostra che non è stato facile per gli studenti convergere su scelte comuni.

Sulla base delle risposte ad una domanda del questionario della fase 3, emerge che la scelta è stata guidata da diversi parametri. In primis, i modelli testati hanno tutti un valore di dimensional dense vector space inferiore o pari a 768 perché i vincoli di RAM non permettevano di lavorare con più dimensioni. Inoltre, molti gruppi hanno selezionato i 3 modelli ordinando quelli presenti nella libreria di Hugging Face per numero di download o like decrescente e scegliendo i primi. Altri si sono concentrati più su caratteristiche tecniche quali la leggerezza, la versatilità, la complessità e la velocità.

La figura 4.3 mostra un grafico analogo al precedente, relativo ai LLM. Anche in questo caso, sull'asse delle y sono riportati i modelli e sulle x il numero di gruppi che li hanno provati. Sono stati provati 23 modelli, ma questa volta solamente 10 hanno un conteggio superiore a 1.

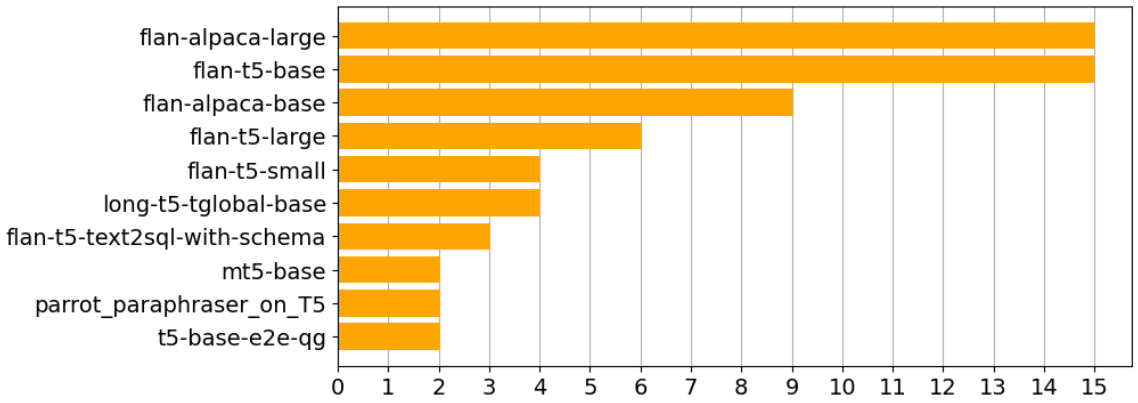


Figura 4.3 - LLM provati da almeno 2 gruppi

Si nota che i modelli "flan-alpaca-large" e "flan-t5-base" emergono come i più comunemente utilizzati, con un conteggio di 15 gruppi (60%) ciascuno, seguiti da "flan-alpaca-base" con 9 (36%). Osservando i 5 modelli più popolari, risulta evidente la propensione alla scelta di modelli basati sulla Few-shot Language Adaptation Network (FLAN), un tipo di rete neurale progettata per adattare modelli di linguaggio naturale a specifici compiti o domini utilizzando solo pochi esempi di addestramento. Alcuni modelli sono stati utilizzati solo da uno o due gruppi, come "mt5-base" e "t5-base-finetuned-question-answering", indicando una varietà di approcci. Complessivamente, anche questo grafico riflette la non ovvietà della scelta dei modelli che meglio si adattano alle attività del progetto.

Gli studenti hanno selezionato i 3 modelli da testare in modo simile a come hanno fatto per gli embedding model, cioè concentrandosi sui più popolari di Hugging Face, in termini di numero di download e di rating degli utenti. Tuttavia, è stato possibile individuare alcuni metodi di selezione particolari ed interessanti. Un gruppo ha fatto dei test aggiuntivi in un notebook a parte per escludere i modelli peggiori e ridurre il numero di alternative possibili. Un altro, invece, ha chiesto a ChatGPT quali fossero i modelli migliori che avessero dimensioni ridotte e fossero adatti per lo sviluppo di un chatbot. Non sono mancati casi in cui la scelta è stata limitata da vincoli tecnici, tra cui lentezza, complessità, costo computazionale, problemi di autenticazione e sovraccarico RAM.

4.1.3 Configurazioni migliori

Dopo aver effettuato i confronti tra le 9 configurazioni, i gruppi hanno scelto la migliore sulla base di un criterio a loro discrezione. Per mostrare i risultati di questa attività viene utilizzata una heat map.

Una heat map è una tecnica di visualizzazione grafica che rappresenta i dati in una griglia utilizzando colori diversi per indicare l'intensità della caratteristica che si vuole misurare. Si tratta di uno strumento ampiamente utilizzato in vari campi per visualizzare la distribuzione spaziale o la concentrazione di determinati fenomeni su una superficie, permettendo di identificare in maniera efficiente le aree di maggiore interesse.

In figura 4.4, la griglia rappresenta i LLM sulle righe e gli embedding model sulle colonne. Ogni cella contiene il numero di gruppi che hanno scelto la combinazione corrispondente come migliore, e assumono diverse gradazioni di rosso sulla base di questo valore.

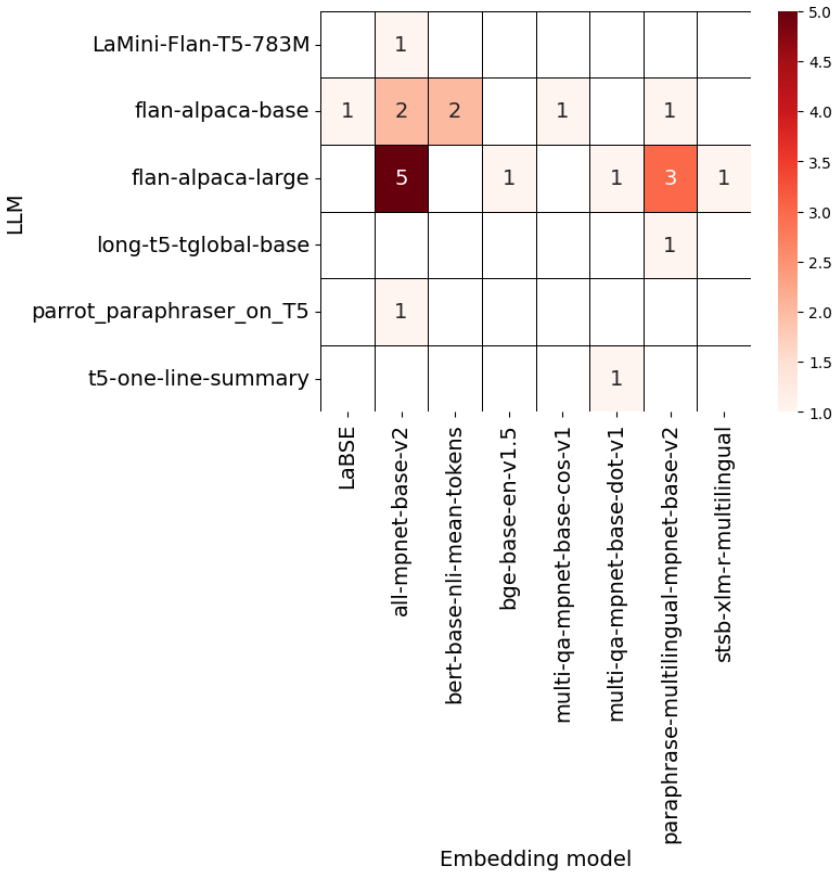


Figura 4.4 - Heat map delle migliori combinazioni scelte dagli studenti

Si osserva che la combinazione scelta più frequentemente è composta dal LLM “flan-alpaca-large” e dall’embedding model “all-mpnet-base-v2”, per un totale di 5 gruppi. D’altra parte, 3 gruppi hanno scelto come migliore la configurazione con “flan-alpaca-large” e “paraphrase-multilingual-mpnet-base-v2”. Le combinazioni scelte da 2 gruppi coinvolgono entrambe “flan-alpaca-base”, combinato a “all-mpnet-base-v2” in un caso e a “bert-base-nli-mean-tokens” nell’altro. Quelle restanti sono state scelte come migliori solo da un gruppo o addirittura da nessuno.

Intervistando alcuni studenti durante questa fase del progetto è stato possibile ottenere informazioni sui criteri utilizzati per selezionare la configurazione migliore. Si è potuto osservare che sono stati sperimentati diversi metodi, in quanto ogni gruppo aveva la libertà di definire le proprie regole, ma si sono identificati dei ragionamenti comuni. Su un totale di 22 gruppi, si è ricavato che:

- Quasi tutti i gruppi (95.5%) hanno considerato aspetti come la correttezza della risposta, la sua semantica o la sua grammatica. Solo il gruppo G23 ha ragionato esclusivamente su vincoli tecnici come il tempo macchina o la disponibilità di RAM. Questo significa che, nel complesso, è stato possibile avere uno spazio di manovra nella scelta della configurazione migliore dal punto di vista della qualità del risultato.
- Altri gruppi oltre a G23 hanno incluso nel metodo di decisione i criteri di velocità e spazio in memoria. In particolare, 4 gruppi (18.2%) hanno considerato il tempo macchina richiesto per ottenere i risultati, e altrettanti hanno considerato i vincoli di RAM.
- Per quanto riguarda la qualità percepita delle risposte generate, sembra che tra la precisione del contenuto, la coerenza interna e la correttezza linguistica, quest’ultima sia stata messa in secondo piano rispetto alle altre il più delle volte. Ciò dimostra che per gli studenti è stato prioritario che la loro applicazione fornisse informazioni coerenti con la domanda invece che prive di errori sintattici o grammaticali.

- A livello di metodo, 6 gruppi (27.3%) hanno dichiarato di avere utilizzato un indicatore aggregato a partire dai valori binari assegnati alle metriche definite dai docenti, con l'obiettivo di definire un punteggio complessivo che potesse aiutare nella scelta della configurazione. D'altra parte, 3 gruppi (13.6%) si sono soffermati sul testo della risposta generata, mettendo da parte i numeri e dedicandosi ad un approccio orientato alla critica del risultato finale. Il resto del campione (59.1%) non ha comunicato esplicitamente di avere seguito un metodo rigoroso.

In sintesi, la scelta della configurazione migliore tra le 9 confrontate ha fornito risultati che confermano la creatività e l'inventiva degli studenti. Comunque, aspetti di base come l'affidabilità delle informazioni restituire, la sensatezza della risposta e la sua correttezza linguistica sono stati comuni alla maggior parte.

4.2 Affidabilità dei chatbot

In questa sezione verranno messe da parte le valutazioni degli studenti per far spazio ad un'analisi delle prestazioni dei chatbot sviluppati dal punto di vista di un esterno. L'obiettivo è quello fornire un'indicazione sull'affidabilità delle risposte generate nei diversi casi di test. Per misurare questa caratteristica si è pensato di fare riferimento alla rilevanza delle informazioni utilizzate dal LLM per generare le risposte.

Come spiegato nel capitolo 2, il chatbot suddivide in chunk i documenti a disposizione e li conserva in un database sotto forma di vettori. Quando riceve in input una domanda, ne calcola l'embedding e lo confronta con quelli del vector store, selezionando le porzioni di testo più simili a livello semantico. Queste ultime vengono infine passate al LLM, che si occupa di rielaborarle e fornire l'output all'utente. Il vantaggio di utilizzare LangChain è che questi frammenti testuali vengono restituiti assieme alla risposta, e sono quindi consultabili.

Il metodo consiste proprio nel valutare la rilevanza di ognuno dei 4 chunk recuperati rispetto alla risposta originale, che si è assunta essere corretta e derivata esclusivamente dal contenuto dei documenti. Per farlo, si è deciso di utilizzare una variabile binaria che assume valore "1" se il chunk contiene almeno un'informazione utile a rispondere alla domanda, e "0" altrimenti.

In questo modo vengono penalizzati solo i casi in cui il chunk recuperato dal vector store è inutile, ad esempio quanto è del tutto decontestualizzato oppure quando il suo contenuto richiama quello della domanda ma non aggiunge nulla di significativo. D'altra parte, se un singolo chunk non è sufficiente a rispondere completamente, verrà comunque valutato in modo positivo. Si prevede infatti che il chatbot utilizzi più chunk diversi per fornire una risposta completa, non uno solo che la contiene per intero.

In alcuni casi le domande erano poste male o si riferivano ad informazioni non contenute in alcuno dei documenti, sebbene le istruzioni specificassero di formularle in modo tale che il chatbot avesse a disposizione la conoscenza sufficiente per rispondere. Comunque, si è deciso di valutare questi esempi come fallimenti perché, invece che dichiarare di non aver trovato la risposta tra i documenti, i chatbot hanno recuperato chunk sconnessi e inutili.

Per generare e scaricare i risultati, ogni gruppo è stato testato tramite il codice disponibile in Appendice A. Di seguito si riportano le sezioni in cui si suddivide il notebook:

- **Installazione pacchetti:** vengono installati con pip pacchetti come langchain, chromadb e huggingface_hub.
- **Caratterizzazione gruppi:** definizione della classe Gruppo e creazione di un oggetto per ogni gruppo da testare.
- **Caricamento documenti:** i documenti vengono salvati sotto forma di oggetti Document, uno per ogni pagina PDF e uno per ogni pagina web.
- **Suddivisione documenti:** i vari Document vengono dati in input alla funzione di split, che ritorna una lista di oggetti Document relativi ai singoli chunk.
- **Traduzione chunk:** eventuali chunk in italiano vengono tradotti in inglese con Google Translator. Da qui si ottiene la lista dei chunk in inglese e un dizionario con le corrispondenze tra le due lingue.

- **Definizione modelli:** viene effettuato un controllo sui token dei chunk, per verificare che l'embedding model sia adatto, e avviene la creazione di retriever e LLM.
- **Generazione risultato:** vengono lette le domande e calcolata per ciascuna di esse una risposta. Si ottengono così quest'ultima e i 4 chunk recuperati.
- **Formattazione risultato:** i dati vengono esportati in formato CSV per una facile lettura.

La classe Gruppo è stata definita per poter utilizzare sempre lo stesso codice, che è stato eseguito a rotazione su tutti gli oggetti tramite una variabile gruppo_corrente che veniva assegnata di volta in volta. Gli attributi più significativi della classe Gruppo sono riportati di seguito:

- **ID_gruppo:** codice identificativo composto da una "G" e da un numero sequenziale che va da "01" a "30" (es. G01);
- **url_git:** URL del repository GitHub in cui sono contenuti i PDF, le domande di prova e le risposte originali;
- **urls:** lista delle URL riferite alle pagine web che fanno parte della documentazione;
- **embeddings_model_name:** nome dell'embedding model migliore;
- **repo_id:** percorso alla directory in cui sono contenuti i file da scaricare per l'utilizzo del LLM migliore;
- **n:** lunghezza dei chunk in termini di numero di frasi, valorizzato se la funzione di split utilizzata è quella proposta dai docenti;
- **overlap:** numero di frasi sovrapposte tra 2 chunk consecutivi, valorizzato se la funzione di split utilizzata è quella proposta dai docenti.

Per ogni gruppo, l'embedding model e il LLM sono stati selezionati a partire dai 3 provati, sulla base del confronto effettuato nella terza fase del progetto e secondo i criteri descritti in sezione 4.1.3.

Inoltre, non tutti i gruppi hanno un valore per n e overlap. Questo perché alcuni hanno deciso di utilizzare una funzione di split personalizzata, strutturata in modo diverso e di conseguenza caratterizzata da altri parametri.

In quest'analisi sono stati esclusi dal campione originario i seguenti gruppi:

- **G06, G07, G18, G26 e G29:** non essendo riusciti a concludere la fase 3, non è stato possibile definire i modelli necessari per effettuare il test.
- **G16:** aveva scelto il PDF di un libro con più di 1000 pagine, che assieme agli altri contribuiva ad un totale di circa 50k chunk, un numero ingestibile. Anche dopo aver effettuato vari tentativi per aggirare il problema, l'esecuzione si interrompeva.
- **G24:** aveva un PDF e una pagina web che non venivano caricati correttamente all'interno dell'ambiente di sviluppo. Si è concluso che ciò fosse causato nel primo caso dal formato del file e da problemi di autorizzazione all'accesso nel secondo.
- **G23:** il link al repository GitHub non esisteva più al tempo del test, quindi i PDF sono andati persi.

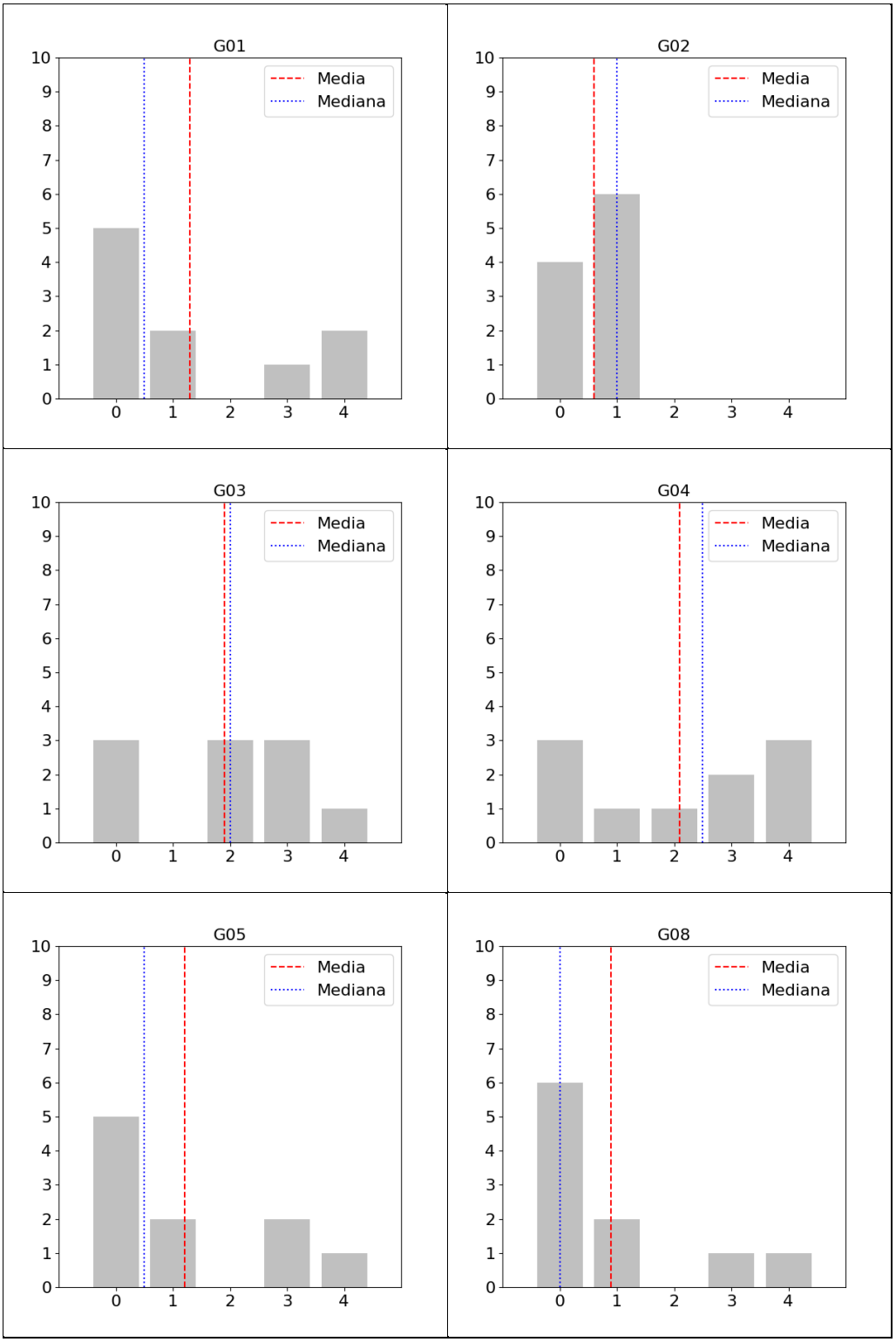
Le analisi presentate nelle prossime sezioni sono quindi effettuate su un totale di 22 gruppi. Per quanto riguarda i gruppi G16 e G24, che sono stati esclusi per via di una documentazione inadeguata, l'opzione di eseguire il test ignorando i documenti problematici è stata rifiutata per evitare che alcune domande non trovassero risposta tra le fonti rimanenti e ottenessero un punteggio non rappresentativo dell'effettiva affidabilità del chatbot.

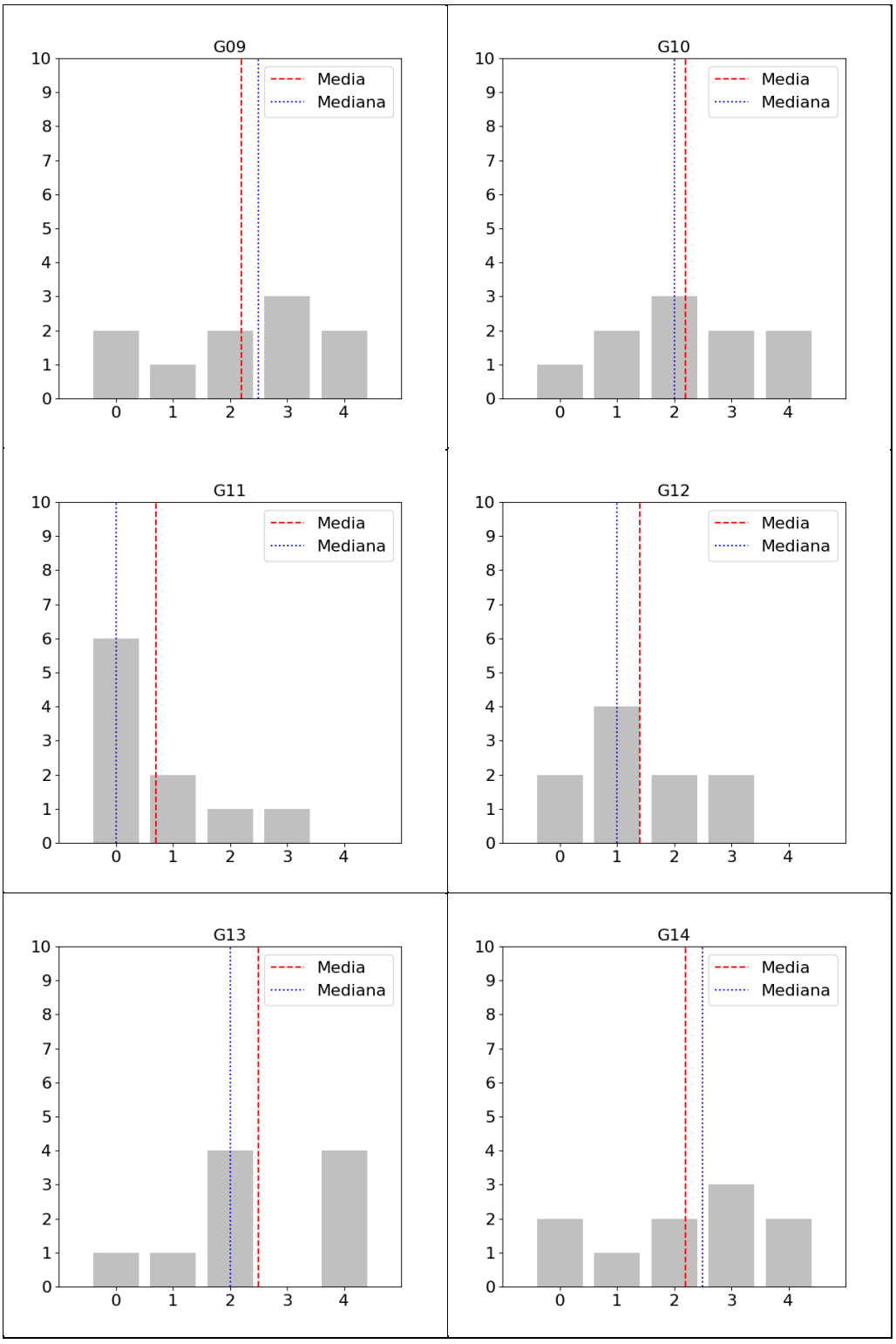
4.2.1 Distribuzioni dei punteggi di affidabilità

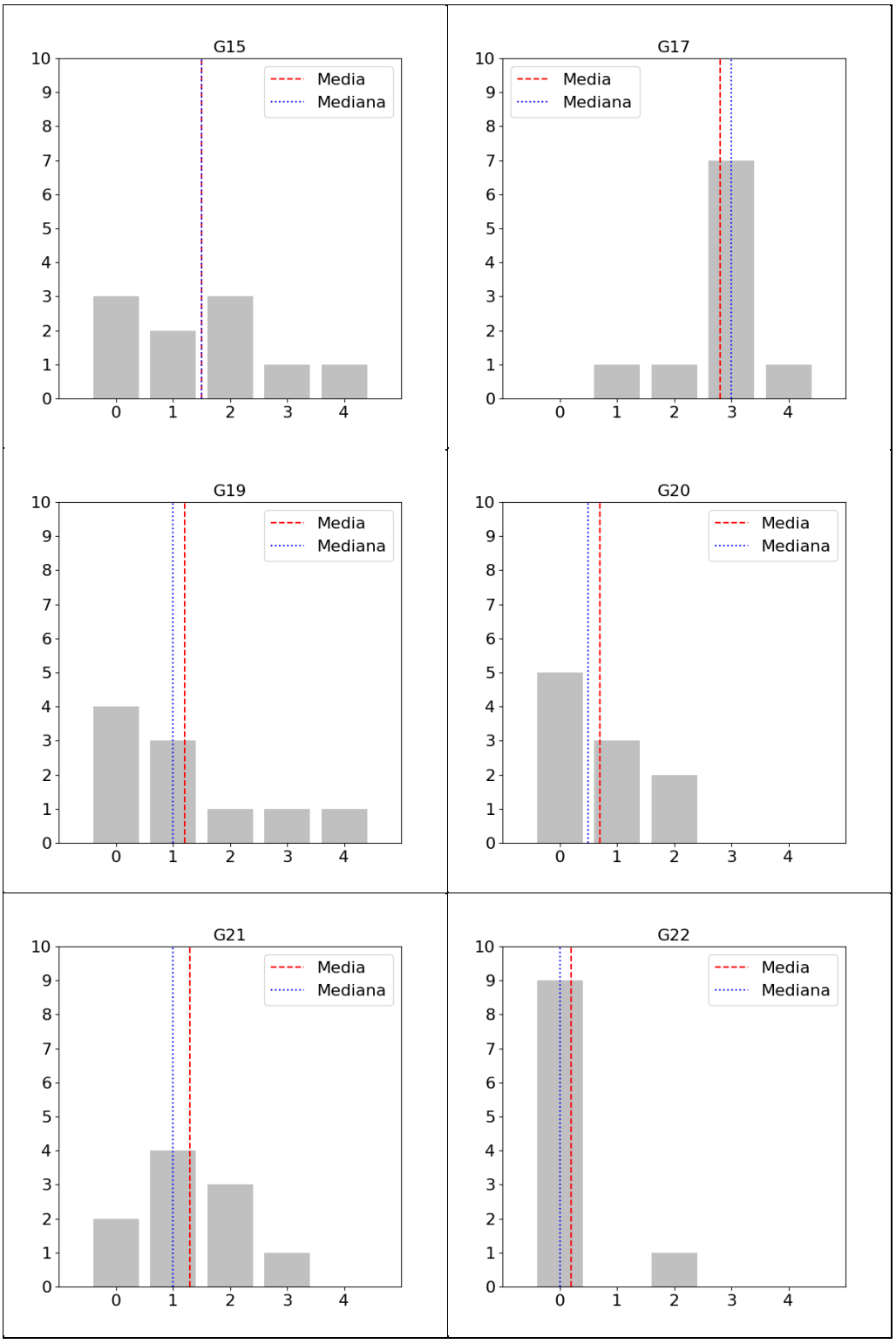
Come anticipato, il vector store fa una classifica dei chunk a disposizione in base alla similarità del coseno calcolata tra i loro embedding e quello della domanda, e restituisce i primi 4. Tali chunk vengono da me valutati con "1" o "0" secondo le regole suddette. In questo modo, per ognuna delle 10 domande di prova si ottiene un punteggio compreso tra 0 e 4, dato dalla somma dei valori binari.

Ad ogni gruppo può quindi essere associata una distribuzione del punteggio. La tabella 4.2 contiene 22 grafici a barre rappresentativi di tali distribuzioni. Ogni grafico è identificato dal codice del gruppo ed ha sull'asse delle ascisse i valori di punteggio e su quello delle ordinate la frequenza ovvero il numero di domande. Osservando le singole barre si può visualizzare quante domande hanno raggiunto un certo

punteggio. Inoltre, come indicato nella legenda, sono riportate anche la media e la mediana della distribuzione.




G09
G10
G11
G12
G13
G14
Media
Mediana


G15
Media
Mediana
G17
Media
Mediana
G19
Media
Mediana
G20
Media
Mediana
G21
Media
Mediana
G22
Media
Mediana

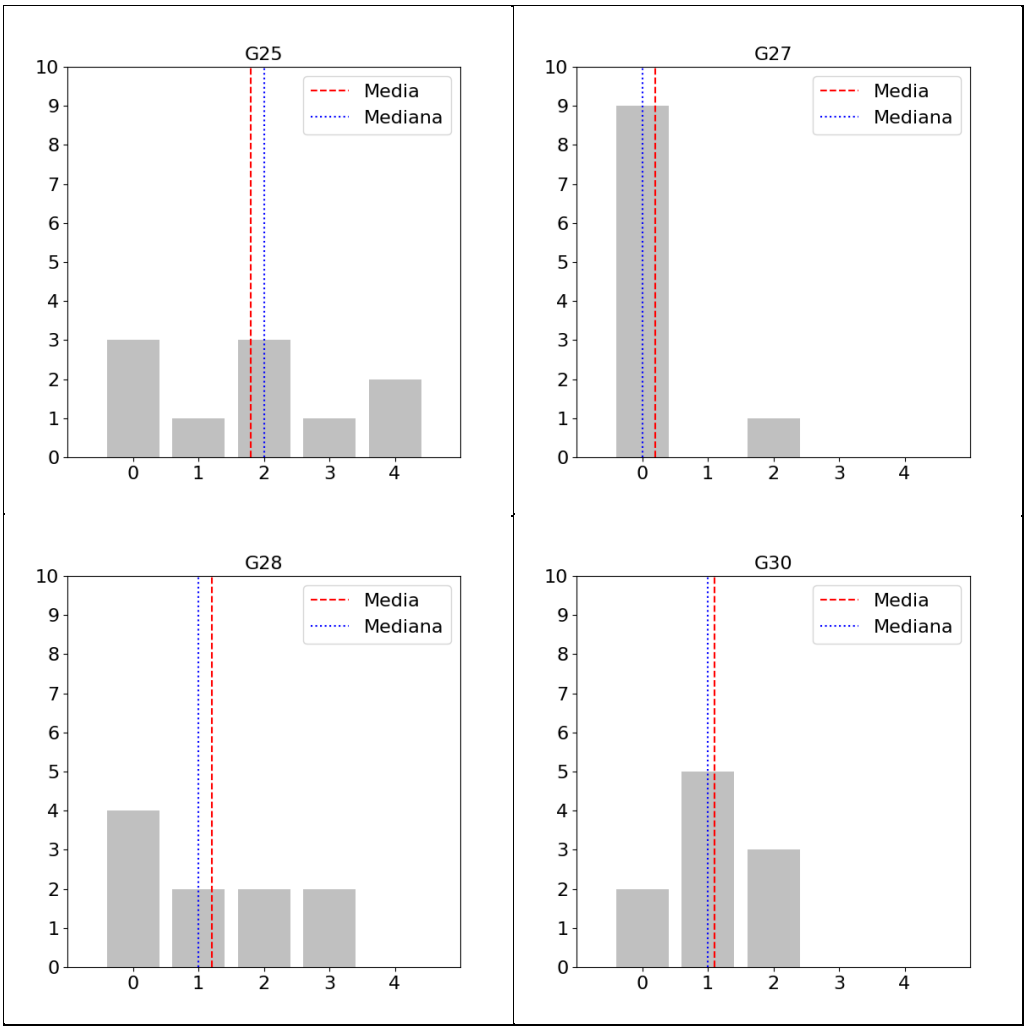


Tabella 4.2 - Distribuzioni dei punteggi

Si può notare che poco più della metà del campione (12 gruppi su 22) ha ottenuto punteggi compresi tra 0 e 4. Un sottoinsieme più ristretto (4 gruppi) non ha mai raggiunto un punteggio superiore a 3 in alcuna domanda, e altrettanti hanno ottenuto punteggi da 0 a 2. Il gruppo G02 ha riportato solo punteggi pari a 0 o 1, suggerendo un'anomalia nei risultati. Questo gruppo era uno dei pochi ad aver utilizzato una funzione di split personalizzata, che suddivideva i documenti in gruppi di parole invece che di frasi. Questo potrebbe essere uno dei motivi per cui ha ottenuto risultati scadenti.

Tra i gruppi con prestazioni peggiori si trovano anche G22 e G27, che tra l'altro presentano una distribuzione con la stessa forma. Entrambi avevano incluso tra i loro documenti pagine web con un layout complesso, che ipoteticamente hanno generato rappresentazioni vettoriali poco affidabili. Inoltre, il gruppo G27 è stato l'unico a preferire l'embedding model "multi-qa-mpnet-base-cos-v1" (cfr. § 4.1.3), testato solamente da altri due gruppi (cfr. § 4.1.2). Anche questo potrebbe aver influenzato negativamente i suoi risultati.

D'altra parte, il gruppo G17 ha ottenuto punteggi pari o superiori 1 in tutte le domande, raggiungendo un punteggio di 3 per 7 domande su 10. È evidente che si tratta di uno dei gruppi migliori. Questo gruppo aveva scelto come configurazione migliore quella composta da "all-mpnet-base-v2" e "flan-alpaca-large", l'embedding model e il LLM più popolari (cfr. § 4.1.2). Non sorprende quindi che questa sia stata la configurazione più comunemente selezionata come migliore (cfr. § 4.1.3), confermando che questi due modelli sono effettivamente i più affidabili.

In conclusione, è chiaro che la qualità dei documenti e la scelta della configurazione abbiano influenzato in modo significativo l'affidabilità dei chatbot, sottolineando l'importanza di tali fattori nella progettazione di questo tipo di applicazioni.

4.2.2 Misure di tendenza centrale dei punteggi

In questa sezione si analizzano le medie e le mediane delle distribuzioni appena commentate. In quanto misure di tendenza centrale, esse possono fornire informazioni interessanti su quali gruppi abbiano ottenuto prestazioni migliori o peggiori.

Si è deciso di riportarle entrambe perché a seconda della forma della distribuzione può essere preferibile guardare l'una piuttosto che l'altra. La media fornisce una rappresentazione accurata quando i dati sono distribuiti in modo approssimativamente normale e non ci sono outlier significativi (es. G09, G10, G25). D'altra parte, la mediana si adatta ai casi in cui ci sono valori estremi rilevanti o quando i dati sono distribuiti in modo asimmetrico (es. G08, G11, G30). In alcuni casi, comunque, può essere utile considerare entrambe le misure per ottenere una visione più completa della distribuzione dei dati.

La figura 4.5 riporta un grafico a barre in cui i 22 gruppi sono disposti in ordine decrescente di media a partire da sinistra. Per ogni gruppo sono riportate entrambe le misure di centratura, i cui valori sono leggibili dalle lunghezze delle barre. Come indicato in legenda, le barre rosse rappresentano le medie, mentre quelle blu le mediane.

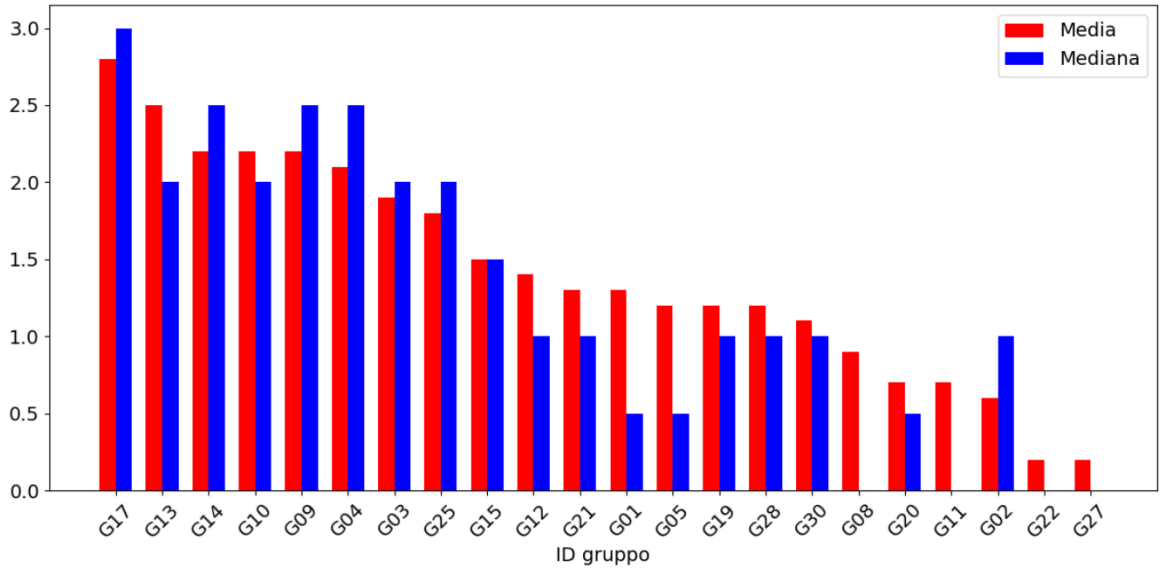


Figura 4.5 - Media e mediana dei punteggi di affidabilità

Da una prima occhiata si può notare che tutti i gruppi hanno ottenuto una media compresa tra 0.2 e 2.8, mentre le mediane variano su un range leggermente più esteso, che va da 0 a 3. Il gruppo migliore risulta G17 secondo entrambe le misure, come anticipato nella sezione precedente. D'altra parte, i gruppi peggiori sono G22 e G27, che hanno media pari a 0.2 e mediana nulla. Anche in questo caso viene confermato quanto già osservato dai grafici delle distribuzioni.

Nell'ottica di voler considerare un punteggio soglia per determinare se un chatbot è affidabile, si illustrano ora le distribuzioni di entrambe le misure di tendenza centrale. Osservando questi grafici è possibile concludere se, complessivamente, i chatbot sviluppati hanno avuto successo.

Si è pensato di considerare 1 come soglia, poiché quasi sempre un solo chunk contenente informazioni utili era sufficiente a generare una risposta corretta. Leggendo le risposte generate dai chatbot, si è potuto notare che la differenza tra le domande con punteggi pari ad 1 e 0 era nettamente superiore a quella tra domande che avevano ottenuto 2, 3 e 4. Sopra l'1 le differenze non erano molto significative,

perché le domande erano spesso puntuali e non richiedevano di mettere assieme porzioni di testo diverse.

In figura 4.6 è mostrato l'istogramma delle medie.

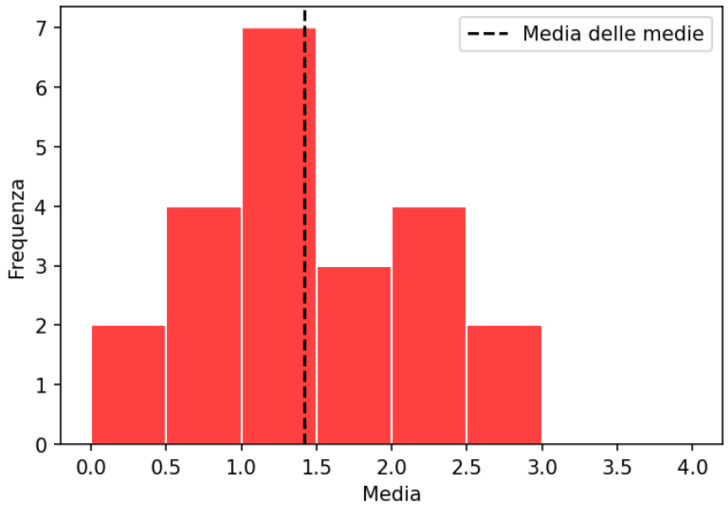


Figura 4.6 - Distribuzione delle medie dei punteggi di affidabilità

Questa rappresentazione comunica in modo immediato che i 22 gruppi nel complesso hanno sviluppato chatbot in grado di recuperare almeno un chunk utile a generare una risposta simile a quella originale. Infatti la media delle medie è superiore a 1.

D'altronde, 6 gruppi (G02, G08, G11, G20, G22 e G27) non hanno raggiunto la soglia di successo con la media dei punteggi. Tuttavia, considerando che il 72.7% del campione ha superato il test, possiamo concludere che la maggior parte dei gruppi ha sviluppato chatbot efficaci.

La figura 4.7 riporta l'istogramma delle mediane.

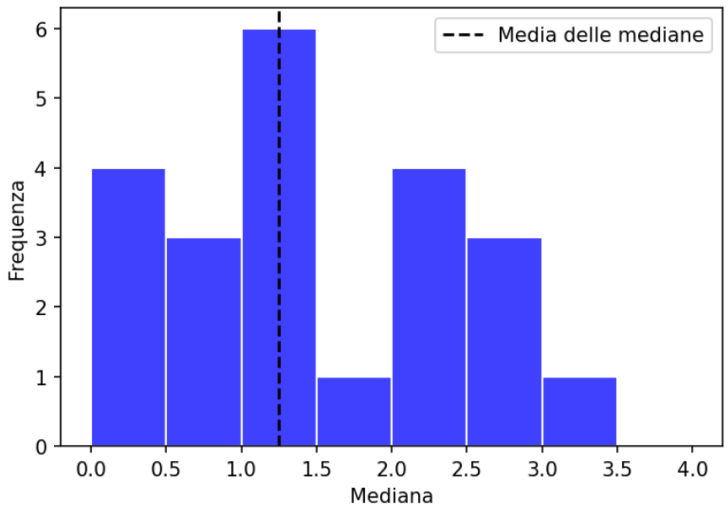


Figura 4.7 - Distribuzione delle mediane dei punteggi di affidabilità

Anche sulla base della mediana risulta che i chatbot abbiano complessivamente superato il test, essendo la media delle mediane superiore a 1.

In questo caso, sono 7 i gruppi che non hanno raggiunto la soglia di successo (G01, G05, G08, G11, G20, G22, G27). Questo significa che il 68.2% del campione ha superato il test, un risultato leggermente inferiore al precedente ma comunque positivo.

La differenza tra i risultati che si origina dalla scelta della misura di riferimento è dovuta ad alcuni gruppi in particolare, che rappresentano fallimenti in un caso e successi nell'altro.

I gruppi con le differenze più significative tra le due misure sono G08, G01, G05 e G11. Questo fenomeno potrebbe essere attribuibile al fatto che i dati seguono una distribuzione che si discosta particolarmente da una normale, probabilmente a causa delle dimensioni ridotte del campione di domande.

Tuttavia, tale discrepanza ha avuto un impatto determinante solamente su G01, G02 e G05, che presentano valori superiori a 1 per una misura e inferiori per l'altra. Questi gruppi meritano una particolare attenzione per decidere quale delle due misure considerare, al fine di definire con maggiore precisione la percentuale dei chatbot affidabili all'interno del campione. Poiché tutte e tre le distribuzioni mostrano asimmetrie, affidarsi alla mediana sarebbe più appropriato; tuttavia, per valutare questo aspetto sarebbe necessaria un'analisi più approfondita che è stata esclusa dal presente documento.

4.2.3 Confronto tra punteggi e valutazioni degli studenti

In questa sezione, i punteggi di affidabilità appena discussi vengono confrontati con le valutazioni effettuate dagli studenti durante la terza fase del progetto.

Come spiegato nel capitolo 3, prima di selezionare la configurazione ottimale, i gruppi devono valutare tutte le possibili combinazioni degli embedding model e dei LLM che hanno scelto di provare. A tal fine, si prevede che assegnino dei valori binari alle metriche di retrieval_effectiveness, llm_semantic_effectiveness e llm_syntax_grammar_effectiveness.

Per questa analisi ci si concentrerà sui valori di retrieval_effectiveness ottenuti dalle configurazioni migliori dei 22 gruppi. La tabella 4.3 contiene i dati che sono stati utilizzati per disegnare i prossimi grafici. Di seguito viene spiegato il significato delle diverse colonne:

- **ID gruppo:** codice identificativo del gruppo.
- **Punt:** somma dei punteggi di affidabilità (da 0 a 4) delle 10 domande di prova.
- **Bin 1:** somma di valori binari corrispondenti ai punteggi di affidabilità, considerando che ad ogni domanda con un punteggio maggiore di 0 è stato assegnato il valore "1", altrimenti uno "0".
- **Bin 2:** somma di valori binari corrispondenti ai punteggi di affidabilità, considerando che ad ogni domanda con un punteggio maggiore di 1 è stato assegnato il valore "1", altrimenti uno "0".
- **Bin 3:** somma di valori binari corrispondenti ai punteggi di affidabilità, considerando che ad ogni domanda con un punteggio maggiore di 2 è stato assegnato il valore "1", altrimenti uno "0".
- **Bin 4:** somma di valori binari corrispondenti ai punteggi di affidabilità, considerando che ad ogni domanda con un punteggio maggiore di 3 è stato assegnato il valore "1", altrimenti uno "0".
- **Stud:** somma dei valori di retrieval_effectiveness delle 10 domande di prova.
- **Ratio_punt:** rapporto tra il punteggio totale (Punt) e il massimo punteggio ottenibile, che è di 40 poiché ogni domanda di prova ha ricevuto fino a 4 punti.
- **Ratio_stud:** rapporto tra la retrieval_effectiveness totale (Stud) e il massimo valore ottenibile, che è di 10 poiché ogni domanda di prova è stata valutata con "0" o "1".

Le colonne Bin 1-4 sono motivate dal fatto che la metrica di retrieval_effectiveness definita dai docenti era binaria, mentre il mio punteggio di affidabilità poteva variare da 0 a 4. Ai fini dell'analisi era necessario rendere comparabili i due valori, quindi sono state fatte quattro ipotesi su come effettuare questo confronto.

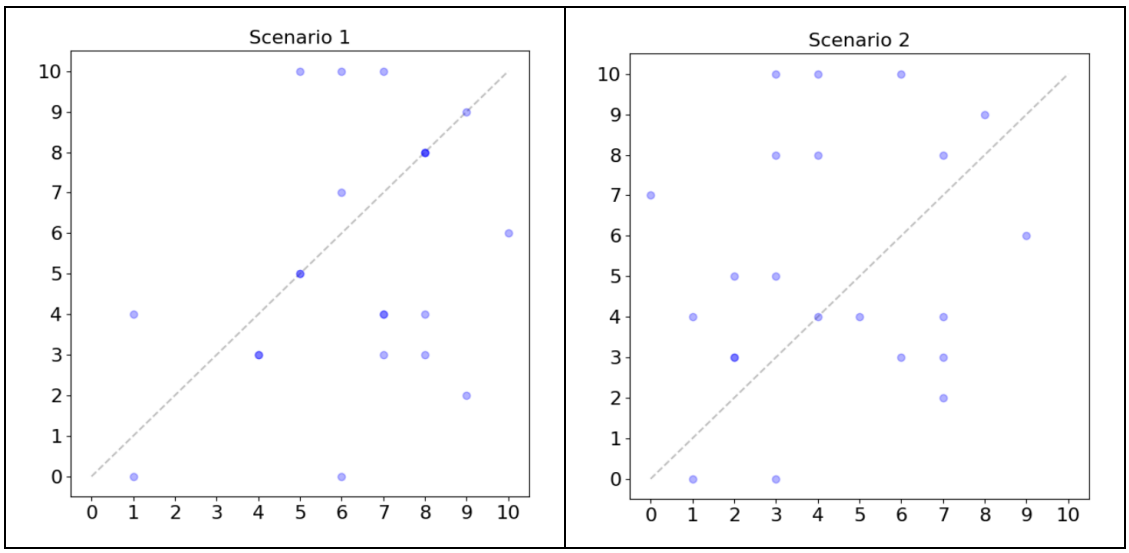
ID gruppo	Punt	Bin 1	Bin 2	Bin 3	Bin 4	Stud	Ratio_punt	Ratio_stud
G01	13	5	3	3	2	10	0.33	1.00

G02	6	6	0	0	0	7	0.15	0.70
G03	19	7	7	4	1	4	0.48	0.40
G04	21	7	6	5	3	3	0.53	0.30
G05	12	5	3	3	1	5	0.30	0.50
G08	9	4	2	2	1	3	0.23	0.30
G09	22	8	7	5	2	3	0.55	0.30
G10	22	9	7	4	2	2	0.55	0.20
G11	7	4	2	1	0	3	0.18	0.30
G12	14	8	4	2	0	4	0.35	0.40
G13	25	9	8	4	4	9	0.63	0.90
G14	22	8	7	5	2	8	0.55	0.80
G15	15	7	5	2	1	4	0.38	0.40
G17	28	10	9	8	1	6	0.70	0.60
G19	12	6	3	2	1	0	0.30	0.00
G20	7	5	2	0	0	5	0.18	0.50
G21	13	8	4	1	0	8	0.33	0.80
G22	2	1	1	0	0	0	0.05	0.00
G25	18	7	6	3	2	10	0.45	1.00
G27	2	1	1	0	0	4	0.05	0.40
G28	12	6	4	2	0	10	0.30	1.00
G30	11	8	3	0	0	8	0.28	0.80

Tabella 4.3 - Dati sul confronto tra punteggi di affidabilità e valutazioni degli studenti

Gli scenari di confronto derivati dalle quattro regole di conversione sono rappresentati negli scatter plot della tabella 4.4. I quattro scenari differiscono per il significato attribuito all'asse delle ascisse. Nello scenario 1 ci sono i valori della colonna Bin 1, nello scenario 2 quelli di Bin 2, nello scenario 3 quelli di Bin 3 e nello scenario 4 quelli di Bin 4. Sull'asse delle ordinate, invece, sono riportati i valori della colonna Stud. In questo modo ogni gruppo è identificato da un punto, che in caso di sovrapposizioni risulterà di un colore più intenso.

La linea tratteggiata raffigura la retta y=x, utile per fare considerazioni su quale valore sia maggiore tra quello calcolato a partire dai punteggi di affidabilità mediante le diverse regole di conversione e quello degli studenti. Se un determinato punto si trova sopra la linea significa che il gruppo ha considerato il proprio chatbot più affidabile di quanto abbia fatto io, e viceversa altrimenti.



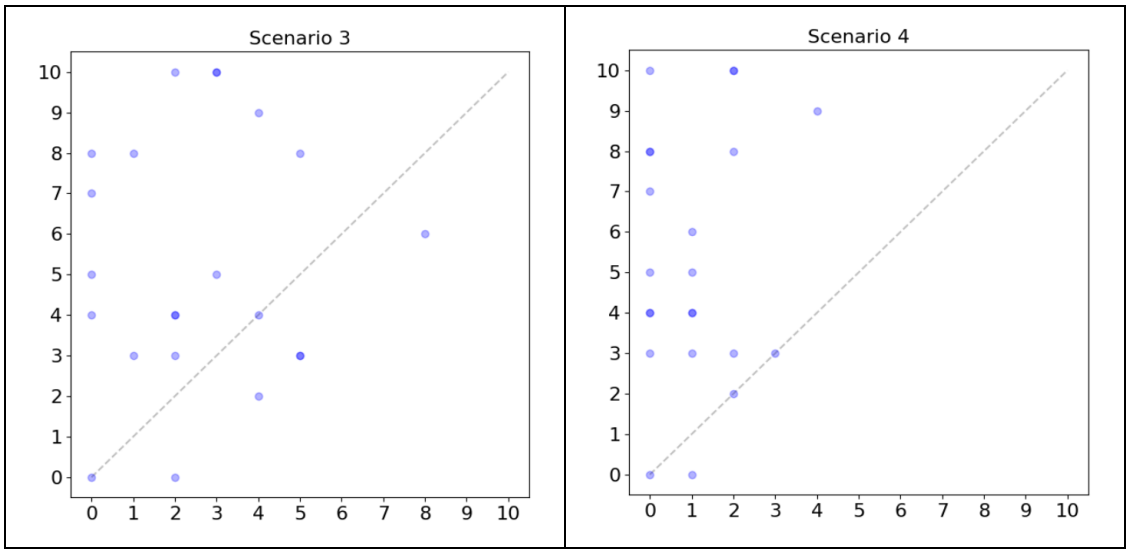


Tabella 4.4 - Scenari di confronto tra punteggi binari di affidabilità e valutazioni degli studenti

Osservando i dati, per prima cosa si nota che dal primo scenario al quarto i punti si spostano dalla zona in basso a destra a quella in alto a sinistra, a dimostrare che la regola di conversione adoperata è sempre più stringente.

Lo scenario 1 coincide con l'assunzione che una risposta sia affidabile se almeno uno dei chunk recuperati contiene informazioni pertinenti alla domanda. Dalla disposizione dei punti si ricava che 6 gruppi hanno valutato il proprio chatbot allo stesso modo, mentre 5 si sono rivelati più ottimisti e 11 più pessimisti.

Nello scenario 2 si presuppone che una risposta sia corretta solo se almeno due chunk contengono informazioni rilevanti. A seguito di questa regola più severa, solamente un gruppo ha dato una valutazione uguale alla mia. I restanti sono stati più generosi nel 59.1% dei casi e meno nel 36.4% dei casi.

Lo scenario 3 dimostra che considerare dei successi solo le risposte che derivano da almeno 3 chunk corretti è un metodo troppo restrittivo. Infatti, solamente 7 gruppi non hanno dato una valutazione più generosa. Di questi, 2 si sono dati lo stesso punteggio da me assegnato.

Nello scenario 4 la situazione è ancora più evidente. Solamente il gruppo G19 ha valutato il proprio chatbot in modo più severo rispetto a me; ma anche in questo caso la differenza tra il mio e il suo valore è solo di 1.

Per concludere, sembra che gli studenti siano d'accordo nel ritenere che per considerare affidabile una risposta sia sufficiente che il vector store recuperi anche un solo chunk contenente informazioni di rilievo. D'altra parte, fare differenze tra i casi in cui due, tre o quattro chunk sono corretti è un approccio che in pochi hanno adottato. Questo dimostra che, a meno che la domanda non sia talmente complessa da richiedere la combinazione di più chunk, un chunk su quattro basti per ottenere un risultato positivo.

In alternativa ai 4 scenari appena discussi, si è pensato di presentare un quinto approccio al confronto tra le mie valutazioni e quelle degli studenti. Come anticipato, la scelta di utilizzare delle regole di conversione era motivata dal fatto che la metrica di retrieval_effectiveness definita dai docenti era binaria. Tuttavia, il risultato della somma di questi valori può essere visto anche come livello di performance sul totale, pari a 10 perché ogni gruppo ha dieci domande di prova. Similmente, la somma dei punteggi di affidabilità può dare una misura comparabile se rapportata al massimo raggiungibile di 40.

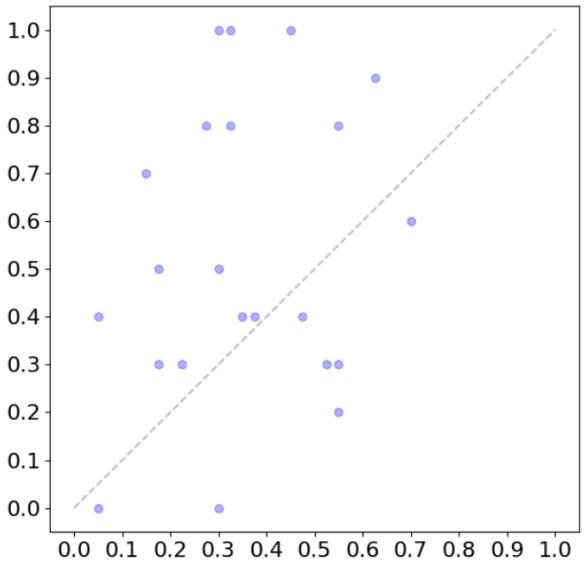


Figura 4.8 - Confronto tra valori normalizzati di punteggi di affidabilità e valutazioni degli studenti

L'obiettivo del grafico di figura 4.8 è proprio quello di mettere in relazione queste due grandezze, i cui valori sono contenuti nelle colonne Ratio_stud e Ratio_punt della tabella 4.3. Ogni gruppo è nuovamente rappresentato da un punto, ma questa volta le coordinate sono date da valori che vanno da 0.0 a 1.0 e che rappresentano la mia valutazione sull'asse delle ascisse e quella degli studenti sull'asse delle ordinate.

Dallo scatter plot sembra che la maggior parte dei gruppi ritenga il proprio chatbot più affidabile di quanto risulti da quest'ultimo approccio, infatti si osservano più punti sopra la linea tratteggiata che sotto. Probabilmente, nel complesso loro sono stati più generosi di me nel valutare il proprio prodotto.

4.2.4 Confronto con ChatGPT

Nell'ultima fase del progetto, ai gruppi viene richiesto di testare ChatGPT con le 10 domande di prova e confrontare le risposte ottenute con quelle della propria applicazione. In questa sezione si discute ad alto livello l'impressione generale degli studenti riguardo alle differenze tra i due modelli.

Osservando le risposte dei diversi gruppi alla domanda aperta posta alla fine del notebook, una delle prime cose evidenti è che ChatGPT fornisce molto spesso risposte più complete, strutturate e linguisticamente corrette. Pertanto, per quanto riguarda la semantica e la correttezza sintattica e grammaticale batte senza dubbio i LLM utilizzati dagli studenti. Ciò è dovuto al fatto che ChatGPT è stato allenato su una vasta quantità di dati e ha a disposizione una marea di fonti a cui attingere.

Tuttavia, diversi gruppi hanno segnalato che, nonostante la forma del contenuto generato da ChatGPT sia di qualità nettamente superiore, non sempre le sue risposte contengono le informazioni cercate. Questo dimostra che è di estrema importanza, quando si ha a che fare con sistemi pre-addestrati, analizzare in modo critico l'output ottenuto, anche se risulta apparentemente sensato. Chiedersi sempre se la risposta prodotta è affidabile è un buon metodo per evitare il rischio di disinformazione.

Comunque, non mancano le evidenze che testimoniano il vantaggio di ChatGPT anche sul fronte dell'affidabilità. Diversi gruppi hanno dichiarato che, nonostante il proprio chatbot fosse stato alimentato con la documentazione necessaria per rispondere alle domande di prova, ChatGPT ha generato risposte più accurate e corrette.

Ci sono poi dei casi in cui l'applicazione sviluppata dagli studenti ha dimostrato una tale efficacia da essere considerata alla pari con il modello di OpenAI. Infatti, diversi gruppi hanno dichiarato di aver ottenuto risposte molto simili, seppur espresse in altre parole. Allo stesso modo, è emerso anche il

contrario: alcuni hanno lamentato che né l'uno né l'altro modello sono stati in grado di replicare fedelmente le risposte originali.

Inoltre, una critica ricorrente nei confronti di ChatGPT è che i risultati tendono ad essere troppo generici, a differenza di quelli ottenuti con i chatbot amatoriali, i quali sono stati progettati appositamente per rispondere a domande su un argomento specifico. Sembra che GPT adotti questo comportamento quando gli mancano le informazioni sufficienti a rispondere in modo preciso, specialmente nei casi in cui la domanda fa riferimento a fonti che potrebbe non aver indicizzato.

Ne è un esempio il gruppo G13, che avendo sviluppato un'applicazione incentrata su un insegnamento del Politecnico di Torino, ha affermato che i propri risultati fossero più accurati di quelli di ChatGPT. Qualcosa di simile è stato osservato sul gruppo G05, che ha focalizzato il proprio lavoro su alcuni prodotti da arredamento di IKEA. Dal loro confronto risulta che il proprio chatbot avesse un vantaggio di informazione sul contesto che gli ha permesso di generare risposte più soddisfacenti.

Leggermente diversa è stata l'esperienza dei gruppi che hanno scelto un argomento che fa riferimento ad eventi successivi alla data limite di ChatGPT, attualmente impostata a gennaio 2022 (es. G15, G27). In questo caso sembra che il modello di OpenAI abbia ammesso di non conoscere affatto le risposte alle domande, dimostrando di avere una sorta di consapevolezza riguardo al proprio bacino di conoscenza ed evitando di fornire una risposta vaga.

Infine, alcuni gruppi condividono un punto a favore delle proprie applicazioni: a parità di correttezza della risposta, ChatGPT tende a fornire informazioni superflue e talvolta anche errate, mentre le risposte ottenute dalla LangChain sono più puntuali e concise.

In sintesi, l'analisi dei confronti effettuati dai diversi gruppi evidenzia che, sebbene ChatGPT offra spesso risposte linguisticamente corrette e ben articolate, la sua affidabilità può variare notevolmente. D'altra parte, gli esempi di RAG sviluppati dagli studenti, che sono alimentati con dati specifici, possono fornire risposte più mirate e soddisfacenti, specialmente su argomenti circoscritti. Tuttavia, l'ampia conoscenza di ChatGPT può risultare vantaggiosa in contesti dove la diversità delle fonti è cruciale. Resta il fatto che la valutazione critica dell'output, indipendentemente dal sistema utilizzato, è essenziale per verificare la precisione e l'affidabilità delle risposte.

4.3 Feedback degli studenti

Questa sezione presenta e discute i risultati del progetto dal punto di vista didattico, ottenuti attraverso l'analisi delle risposte degli studenti ad alcune delle domande presenti nel questionario della fase 3 (cfr. sezione 3.3.3). Tali domande sono state progettate in modo tale che i docenti potessero raccogliere informazioni riguardo alle sfide che essi hanno dovuto affrontare e al livello di soddisfazione raggiunto.

Nelle prossime analisi il campione è composto da 25 gruppi, essendo stati esclusi da quello originale G06, G07, G18, G26 e G29.

4.3.1 Sfide tecniche

In questa sezione vengono riassunti i risultati relativi alle domande Q1, Q2, e Q3 del questionario. La prima domanda chiedeva quali fossero in generale le difficoltà riscontrate durante lo sviluppo dell'applicazione. Analizzando le risposte degli studenti è stato possibile individuare dei fattori comuni.

Quasi la metà dei gruppi (12 su 25) ha dichiarato di aver avuto problemi nella comprensione e nella scrittura del codice. La causa principale è che molti studenti ritenevano di non avere accumulato l'esperienza Python sufficiente per utilizzare le librerie necessarie e gestire la quantità di dati richiesta. Alcuni hanno provato a risolvere chiedendo chiarimenti ai docenti, ai loro collaboratori o agli stessi compagni, che però molte volte avevano le stesse difficoltà. Altri si sono affidati a ChatGPT, ma non sempre hanno ottenuto consigli soddisfacenti. Di conseguenza, molti non avevano la possibilità di avanzare il proprio lavoro in autonomia e dipendevano dall'orario di laboratorio, che per l'elevato numero di richieste era spesso troppo ridotto.

La maggior parte (13 gruppi) ha segnalato di aver avuto difficoltà nella ricerca di embedding model e LLM adeguati. Infatti, molto spesso l'esecuzione del codice veniva interrotta a causa del superamento

del limite RAM, e gli studenti hanno dovuto fare svariati tentativi prima di trovare una coppia di modelli sufficientemente leggeri da permettere di completare il run. D'altra parte, alcuni hanno dovuto cambiare modelli a causa dei risultati ottenuti, che non erano all'altezza di valutazioni successive. Un gruppo ha addirittura commentato che in alcuni casi il LLM che aveva scelto si limitava a ripetere la domanda senza aggiungere nulla. Nel complesso, l'esplorazione di Hugging Face e le esecuzioni multiple hanno richiesto una quantità di tempo significativa.

Un sottoinsieme piuttosto ristretto (6 gruppi) ha considerato tra le sfide più importanti la ricerca dei documenti. Quasi tutti hanno attribuito questa criticità alla presenza di errori durante il caricamento, ma un gruppo ha segnalato che la documentazione disponibile sull'argomento scelto era talmente ridotta da non riuscire a garantire risultati accettabili.

Dall'analisi delle risposte alla domanda Q2, che chiedeva quali fossero state le difficoltà durante il caricamento dei documenti, è stato possibile ricavare ulteriori dettagli. Alcune squadre hanno avuto problemi con i file PDF, che non venivano caricati perché il formato non era supportato, perché contenevano troppe immagini oppure perché erano caratterizzati da un'impaginazione che ne rendeva difficile l'interpretazione. Tuttavia, hanno risolto cambiando il formato del file nello standard ISO 32000 o sostituendo il PDF con un altro che contenesse prevalentemente testo. Un gruppo ha optato per una soluzione alquanto originale: scrivere da zero dei documenti che contenessero le risposte alle domande di prova.

D'altra parte, le pagine web non venivano caricate perché protette da copyright, perché richiedevano delle credenziali d'accesso o perché erano dotate di un layout troppo complesso. Per far fronte a questi problemi, alcuni hanno deciso di convertire le pagine in PDF, eliminando header e pubblicità. Altri, invece, hanno direttamente cambiato siti, scegliendone alcuni che fossero privi di restrizioni e di facile lettura da parte dell'applicazione.

La domanda Q3 chiedeva di segnalare eventuali difficoltà nella fase di suddivisione dei documenti. Dalle analisi precedenti sembra che questo passaggio non sia stato considerato da nessuno come particolarmente critico, ma osservando le risposte alla terza domanda sono emersi alcuni aspetti interessanti. In particolare, molto spesso la lunghezza dei segmenti superava il numero massimo di token consentito dal modello di embedding. Per far fronte a questo problema, i gruppi hanno suddiviso ulteriormente i segmenti che non rispettavano il vincolo, oppure si sono inventati una propria funzione di split. Inoltre, diversi gruppi hanno fatto vari tentativi per trovare i valori ottimali della lunghezza dei chunk e dell'overlap, specificando che questi parametri influivano significativamente sull'affidabilità delle risposte generate.

Altre sfide meno comuni incontrate durante l'intero processo di sviluppo sono state la definizione delle domande di prova e la comprensione delle consegne e degli obiettivi delle varie attività di laboratorio, entrambe segnalate da un solo gruppo.

In sintesi, si può affermare che gli studenti siano stati messi alla prova in vari momenti, ma dai risultati sembra che le sfide elencate non abbiano ostacolato l'andamento complessivo delle attività. Infatti, la maggior parte dei gruppi ha adottato strategie efficaci per superarle.

4.3.2 Livello di soddisfazione

In questa sezione vengono riassunti i risultati relativi alle Q7, Q8 e Q9 del questionario. Queste domande sono state progettate per capire quali fossero le aspettative degli studenti e quali risultati, invece, avessero effettivamente ottenuto. Inoltre, le risposte danno un'idea su quali chatbot, sempre secondo gli studenti, possano essere considerati sufficientemente affidabili da essere utilizzati al di fuori dell'esercitazione.

Dalle risposte fornite alle tre domande emerge una varietà di opinioni riguardo al livello di soddisfazione nei confronti dell'applicazione realizzata. Inizialmente, molti gruppi esprimevano aspettative positive, sperando di creare un chatbot funzionante basato sull'intelligenza artificiale, ma avevano sottovalutato la complessità del processo di sviluppo. Alcuni di questi si aspettavano che i documenti forniti fossero sufficienti a generare risposte concise, complete e logicamente corrette; altri, che il chatbot fosse in grado di effettuare ragionamenti complessi, come confronti o valutazioni. Tuttavia, non sono mancati gruppi che si sono dimostrati un po' più cauti. Una squadra ha ammesso che non pensava di ottenere

da subito risposte gratificanti, un'altra sembrava consapevole che le risposte sarebbero state abbozzate e poco comprensibili.

D'altra parte, i risultati effettivamente ottenuti non sempre hanno soddisfatto le aspettative. Pochi gruppi sono rimasti positivamente sorpresi dalle prestazioni della propria applicazione, e anche in questi casi è stato specificato che la situazione era migliorabile. Molti di più sono invece quelli che hanno lamentato la mancanza di senso compiuto e affidabilità nelle risposte. Tra questi, un gruppo ha evidenziato che a volte le risposte sembravano addirittura inventate a partire da conoscenze pregresse del LLM. In generale, la maggior parte ha dubitato dell'utilità pratica della propria applicazione.

La questione dell'affidabilità dei chatbot è stata approfondita nelle risposte alla nona domanda. Alcuni gruppi hanno dichiarato che, se ulteriormente perfezionato, il proprio chatbot potrebbe essere effettivamente utilizzato, ad esempio per divertimento o per scopi didattici. Al contrario, una parte ritiene che impiegare il proprio chatbot in un contesto reale sarebbe troppo rischioso, in quanto porterebbe il più delle volte a conclusioni errate. In linea di massima, tutte le squadre hanno riconosciuto un potenziale nello strumento realizzato, ma molto spesso la scelta dei documenti e dei modelli ha compromesso i risultati finali. Pertanto, sarebbero necessarie notevoli migliorie nelle fasi di progettazione e sviluppo prima di poter considerare queste applicazioni realmente utilizzabili.

In sintesi, è difficile dire se nel complesso gli studenti siano rimasti soddisfatti o meno dal proprio lavoro, perché come si osserva dai loro riscontri non tutti hanno avuto la stessa esperienza. Le impressioni variano da quella di grande soddisfazione per i risultati ottenuti, seppur con alcune ottimizzazioni possibili, a quella di completa delusione per la mancanza di affidabilità e precisione nelle risposte.

4.3.3 Commenti e critiche sul progetto

L'ultima domanda del questionario chiedeva agli studenti di condividere eventuali commenti e critiche sul lavoro svolto, in modo tale che i docenti potessero approfittare di queste informazioni per migliorare l'esperienza didattica del progetto.

Diversi gruppi hanno dimostrato interesse e coinvolgimento positivo. Alcuni dei commenti che rientrano in questa categoria sono stati i seguenti:

- "Il laboratorio ci ha messo alla prova, ma nonostante la nostra comprensione del linguaggio marginale, siamo rimasti incuriositi dal mondo degli embedding model e del large language model".
- "Il laboratorio è stato molto interessante perché ci ha permesso di approcciarci al mondo dei bot, guidandoci per step attraverso le varie fasi di realizzazione".
- "Abbiamo trovato il lavoro svolto estremamente innovativo e stimolante, specialmente considerando l'attualità dell'argomento trattato, che fa parte integrante della nostra vita quotidiana. Un aspetto da approfondire potrebbe essere comprendere come si possa perfezionare l'addestramento del bot attraverso molte iterazioni, mirando a ottenere risposte sempre più precise nel corso del tempo".
- "Abbiamo ritenuto interessante esplorare questo aspetto della programmazione relativo alla creazione di un chatbot. È stato affascinante scoprire come le diverse combinazioni di embedding model e language model possano produrre risposte così diverse tra loro e cercare la combinazione ottimale è stato stimolante".
- "Thank you for these lab sessions; they have opened new doors and unveiled a set of possibilities for us. Personally, we got inspired to continue learning more about natural language processing (NLP) to further enhance our understanding and skills in this exciting field".

Altri hanno espresso delle critiche pur mantenendo un'impressione positiva. Di seguito alcuni esempi:

- "Il laboratorio è ben strutturato, avremmo forse preferito una guida più rigorosa sulle azioni da svolgere, vista comunque la complessità della programmazione utilizzata".
- "L'attività è stata molto interessante e sicuramente utile però alcuni aspetti tecnici relativi ai codici di implementazione dovrebbero essere approfonditi meglio al fine di avere una visione completa e chiara di ciò che si sta facendo".
- "È un tema interessante, ma anche complesso da comprendere. Tutto sommato è stato molto d'aiuto per capire come funziona la libreria pandas, e capire all'incirca la tipologia di dati preferiti

da un chatbot realizzato in questo modo. Forse il tempo che abbiamo avuto a disposizione è poco per programmarne uno efficiente".

Non mancano le squadre che sono rimaste insoddisfatte dall'organizzazione o dai risultati ottenuti. I loro commenti potrebbero essere utili per migliorare le prossime versioni del progetto. Di questi, si riportano i più significativi:

- "Purtroppo non siamo riusciti ad apprezzare al massimo questo laboratorio per mancanza di competenze iniziali e secondo noi c'era bisogno di almeno un altro collaboratore in aula per riuscire ad ottenere risposte più velocemente".
- "Abbiamo trovato che il laboratorio fosse un po' distante dal contenuto del corso e, in particolare, che non disponessimo di quasi nessuna conoscenza pregressa per poterlo affrontare. Pensavamo di dover svolgere un laboratorio che avesse a che fare con gli argomenti svolti a lezione, ma questo, purtroppo, ci ha un po' disorientati e ci ha portati a dover svolgere due volte parte del lavoro".
- "La creazione del chatbot ha comportato l'utilizzo di molti frammenti di codice Python di cui abbiamo faticato a comprendere il funzionamento e la logica. L'esperienza di laboratorio è risultata quindi confusionaria e spesso abbiamo perso di vista l'obiettivo".
- "I laboratori sono stati abbastanza complessi. Le migliorie da introdurre sono sicuramente l'aggiunta di ore di spiegazione teorica delle varie fasi del lavoro per comprendere meglio l'utilizzo delle diverse librerie per la creazione del chatbot".

Da quest'analisi emerge un quadro variegato ma istruttivo sulle percezioni riguardanti il progetto. Mentre alcuni hanno mostrato interesse e apprezzamento, altri hanno evidenziato delle critiche costruttive che potrebbero essere utili per migliorare la struttura e l'organizzazione delle future edizioni. È importante considerare attentamente tutti questi feedback, al fine di ottimizzare ulteriormente il percorso didattico e garantire un'esperienza più piacevole per tutti gli studenti coinvolti.

4.4 Valutazione dei docenti

Quest'ultima sezione del capitolo è dedicata all'andamento del progetto al punto di vista dei docenti. L'obiettivo dell'esperienza era quello di integrare la teoria dei database relazionali con uno strumento tecnologico all'avanguardia che coinvolge l'intelligenza artificiale generativa.

Dei 30 gruppi che hanno partecipato al progetto, 25 sono stati in grado di sviluppare e valutare un'applicazione funzionante di Q&A basata sulla RAG, raggiungendo l'obiettivo principale dei laboratori. La qualità del loro lavoro è stata misurata non tanto dalle prestazioni delle loro applicazioni, quanto dalla capacità di mostrarsi partecipi e curiosi.

Il punteggio finale è stato calcolato tenendo conto di vari aspetti, quali il numero di attività portate a termine, la puntualità delle consegne, la sottomissione delle risposte ai questionari, il lavoro extra e lo spirito collaborativo. I valori assegnati vanno da 1 a 4, e la maggior parte dei gruppi ha ottenuto il massimo. Il progetto aveva un peso del 15% sul voto finale dell'insegnamento.

Diverse squadre sono andate oltre alle richieste dei docenti. Alcuni hanno provato un numero maggiore di LLM e altri hanno definito più domande di prova da sottoporre al chatbot. Inoltre, una coppia si è mostrata particolarmente attiva nel fornire assistenza a chi si trovava in difficoltà, e per questo è stata premiata con un voto più alto.

In conclusione, si può dire che, nonostante le sfide affrontate, gli obiettivi relativi alle capacità tecniche sono stati raggiunti dalla maggior parte degli studenti. Inoltre, sembra che gli studenti abbiano acquisito la consapevolezza che i sistemi di intelligenza artificiale generativa possono generare risposte apparentemente sensate per gli esseri umani ma potenzialmente non affidabili.

5 Proposte di miglioramento

Nell'ottica di ripetere il progetto "Q&A Information Retrieval" almeno l'anno prossimo, questa sezione raccoglie una serie di proposte per le future edizioni, che possono essere uno spunto anche per chi volesse creare una propria versione del progetto.

5.1 Miglioramenti didattici

Dal punto di vista didattico, il progetto potrebbe essere migliorato dall'aggiunta di nuove attività. Di seguito si riporta qualche idea:

- **Test incrociati tra gruppi diversi:** far provare il chatbot ai compagni, che per ognuna delle domande devono dire se le risposte ottenute rispettano le aspettative. In questo modo ogni gruppo può ricevere da altri utenti un riscontro sull'effettiva utilità del proprio chatbot.
- **Questionari aggiuntivi:** per ognuna delle fasi, sottoporre agli studenti domande sull'esperienza che potrebbero essere utili per fare ulteriori analisi in vista delle future edizioni del progetto.
- **Valutazione dell'influenza di vari fattori sulla qualità delle risposte del chatbot:** ad esempio, la specificità dell'argomento, la lunghezza e la struttura dei documenti, e la complessità delle domande.
- **Considerare il metodo di suddivisione dei documenti tra le variabili per la creazione delle configurazioni:** oltre a embedding model e LLM, gli studenti potrebbero sperimentare diversi text splitter e valutarne le prestazioni relative. In particolare, potrebbero variare le dimensioni dei chunk e l'overlap tra di essi.

D'altra parte, per migliorare la qualità delle valutazioni svolte dagli studenti, è importante che le metriche siano definite in maniera più rigorosa, per evitare che i risultati delle valutazioni dei chatbot siano troppo soggettivi. A tal fine potrebbero rivelarsi utili delle sessioni preliminari per discutere in modo collettivo sul significato delle metriche, attraverso degli esempi. Le metriche che si pensa possano essere adatte riguardano i seguenti aspetti:

- **Verità della risposta:** rispetto al contenuto dei documenti, che livello di verità assumono le informazioni contenute nella risposta del chatbot? Ad esempio, si potrebbe decidere di dare un giudizio qualitativo del tipo "basso", "medio", "alto" e far coincidere ad ognuno un numero. In tal caso si potrebbe utilizzare una scala di potenza (es. 1, 3, 9), ed eventualmente normalizzare.
- **Coerenza interna:** i singoli elementi della risposta del chatbot rientrano in un contesto comune? I concetti esposti seguono una sequenza logica (es. causa-effetto, temporalità)? Si potrebbero stabilire dei pesi per questo e altri aspetti inerenti alla coerenza e valutare ciascuno in modo binario, in modo tale da ottenere un indicatore aggregato rappresentativo delle prestazioni ottenute.
- **Correttezza linguistica:** la risposta del chatbot presenta errori di grammatica, ortografia, sintassi o punteggiatura? Per ogni tipo di errore si potrebbero definire dei valori di penalità in modo tale che sia possibile dare un punteggio complessivo sulla qualità dal punto di vista della lingua.
- **Tempo di risposta:** qual è il tempo macchina necessario al chatbot per produrre la risposta? Questa metrica potrebbe dare un'indicazione sull'efficienza di un chatbot rispetto all'altro, e se messa in relazione alle altre permette di effettuare un confronto che tiene conto sia dei costi computazionali che dei benefici in termini di qualità del risultato.

Una discussione di questo tipo sarebbe efficace non solo nell'ottica di ottenere risultati più efficaci dalla valutazione degli studenti, ma anche per far loro esercitare il pensiero critico in aula.

5.2 Miglioramenti tecnici

A livello tecnico, una soluzione più interessante rispetto all'applicazione di Q&A basata su RAG potrebbe essere l'agente conversazionale, anche conosciuto come assistente virtuale o chatbot avanzato.

Un agente è un programma avanzato di intelligenza artificiale generativa, progettato per simulare una conversazione umana naturale con gli utenti. Nelle applicazioni Q&A e nei chatbot, una sequenza di azioni è predefinita all'interno del codice di sviluppo. Negli agenti, invece, un modello di linguaggio viene utilizzato come motore di ragionamento per determinare quali azioni compiere e in quale ordine.

Gli agenti conversazionali possono gestire conversazioni più complesse, mantenere il contesto delle interazioni precedenti e persino apprendere dalle chat per migliorare le prestazioni nel tempo. È questo che li contraddistingue maggiormente dai chatbot, che tendono ad essere più orientati ai compiti e basati su regole predefinite. Entrambi sono utilizzati per migliorare l'esperienza dell'utente e automatizzare processi, ma gli agenti conversazionali offrono una maggiore flessibilità e contestualizzazione nelle conversazioni.

Per concludere, il campo dell'intelligenza artificiale è in continua evoluzione e sta cambiando molto rapidamente. È molto probabile che in futuro gli strumenti discussi in questo elaborato verranno presto superati da altri ancora più avanzati, offrendo nuove opportunità e sfide da esplorare.

Conclusioni

Il presente lavoro di tesi ha esplorato una sperimentazione didattica incentrata sul tema dell'intelligenza artificiale generativa che si è svolta nell'ambito di un corso di Basi di dati. Gli studenti coinvolti nel progetto, iscritti al terzo anno del corso di laurea in Ingegneria del Cinema e dei Mezzi di Comunicazione del Politecnico di Torino, hanno dovuto sviluppare un'applicazione di Q&A basata sulla tecnica della RAG.

L'obiettivo dell'esercitazione era quello di far loro conoscere in modo pratico una tecnologia all'avanguardia finalizzata alla generazione di contenuto informativo a partire da un insieme di fonti, e farli riflettere sull'affidabilità dei propri modelli affinché assumessero consapevolezza riguardo al rischio di disinformazione caratteristico dell'intelligenza artificiale generativa.

Grazie all'analisi dei risultati ottenuti dalle applicazioni realizzate e delle risposte degli studenti a interviste e questionari è stato possibile discutere su aspetti di progettazione dei chatbot, prestazioni in termini di affidabilità e feedback riguardanti le sfide incontrate e il livello di soddisfazione raggiunto.

Nel complesso, sono emersi sia aspetti positivi che aree di miglioramento. Tra i punti positivi, gli studenti hanno apprezzato l'esperienza innovativa e stimolante, che li ha esposti a tecnologie avanzate come i LLM, e che ha suscitato interesse per approfondimenti futuri nel campo del NLP. Tuttavia, diverse sfide emergono per le future edizioni del progetto, in particolare la necessità di trovare un equilibrio migliore tra complessità del lavoro e supporto adeguato da parte dei collaboratori del corso. Le osservazioni degli studenti hanno evidenziato la necessità di adattare il livello di difficoltà del laboratorio alle loro conoscenze tecniche preliminari, possibilmente offrendo diversi livelli di difficoltà in modo tale da evitare frustrazione da un lato e mancanza di stimoli dall'altro.

Nonostante molti studenti siano riusciti a sviluppare un'applicazione in grado di produrre risposte linguisticamente corrette e coerenti al contenuto delle fonti a disposizione, una parte significativa di essi ha espresso riserve sulla affidabilità complessiva del chatbot, riconoscendo le capacità dei LLM di generare risposte che possono avere senso per gli esseri umani ma che potrebbero contenere informazioni false.

Negli ultimi mesi, l'intelligenza artificiale generativa ha dimostrato di avere un enorme potenziale in vari contesti accademici e professionali, dal recupero di informazioni alla generazione di contenuto creativo. Tuttavia, è di estrema importanza utilizzarla sempre con spirito critico e consapevolezza dei suoi limiti. Gli utenti devono aver presente che i modelli di linguaggio non comprendono il contesto nel modo in cui lo farebbe un essere umano, e possono generare risposte fuorvianti o errate. Pertanto, è

fondamentale valutare attentamente l'affidabilità delle risposte ottenute e integrare queste con altre fonti di conoscenza per prendere decisioni responsabili.

Appendice A Codice per l'estrazione dei chunk

Codice chunk retrieval - Colab.pdf

Bibliografia

[1] Laura Farinetti and Lorenzo Canale. 2024. Chatbot Development Using LangChain: A Case Study to Foster Creativity and Critical Thinking. In Proceedings of the 2024 Conference on Innovation and Technology in Computer Science Education V. 1 (Milan, Italy) (ITiCSE 2024). Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3649217.3653557

[2] Ait Baha, T., El Hajji, M., Es-Saady, Y., & Fadili, H. (2023). The impact of educational chatbot on student learning experience. *Education and Information Technologies*. https://doi.org/10.1007/s10639-023-12166-w

[3] Clarizia, F., Colace, F., Lombardi, M., Pascale, F., & Santaniello, D. (2018). Chatbot: An Education Support System for Student. In A. Castiglione, F. Pop, M. Ficco, & F. Palmieri (Eds.), *Cyberspace Safety and Security* (pp. 291–302). Springer International Publishing.

[4] Giannini, S. (2023). Reflections on generative AI and the future of education. White Paper Published on 18 September 2023. https://teachertaskforce.org/sites/default/files/2023-07/2023_Giannini-UNESCO_Generative-AI-and-the-future-of-education_EN.pdf

[5] Daun, M., & Brings, J. (2023). How ChatGPT Will Change Software Engineering Education. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023)* (pp. 110–116). Association for Computing Machinery. https://doi.org/10.1145/3587102.3588815

[6] Laato, S., Morschheuser, B., Hamari, J., & Björne, J. (2023). AI-Assisted Learning with ChatGPT and Large Language Models: Implications for Higher Education. In *2023 IEEE International Conference on Advanced Learning Technologies (ICALT)* (pp. 226–230). https://doi.org/10.1109/ICALT58122.2023.00072

[7] Mosaiyebzadeh, F., Pouriyeh, S., Parizi, R., Dehbozorgi, N., Dorodchi, M., & Batista, D. M. (2023). Exploring the Role of ChatGPT in Education: Applications and Challenges. In *Proceedings of the 24th Annual Conference on Information Technology Education (SIGITE '23)* (pp. 84–89). Association for Computing Machinery. https://doi.org/10.1145/3585059.3611445

[8] Pinto, G., Cardoso-Pereira, I., Monteiro, D., Lucena, D., Souza, A., & Gama, K. (2023). Large Language Models for Education: Grading Open-Ended Questions Using ChatGPT. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering (SBES '23)* (pp. 293–302). https://doi.org/10.1145/3613372.3614197

[9] Rajabi, P., Taghipour, P., Cukierman, D., & Doleck, T. (2023). Exploring ChatGPT's Impact on Post-Secondary Education: A Qualitative Study. In *Proceedings of the 25th Western Canadian Conference on Computing Education (WCCCE '23)* (Article 9, 6 pages). Association for Computing Machinery. https://doi.org/10.1145/3593342.3593360

[10] Rajala, J., Hukkanen, J., Hartikainen, M., & Niemel, P. (2023). Call Me Kiran – ChatGPT as a Tutoring Chatbot in a Computer Science Course. In *Proceedings of the 26th International Academic Mindtrek Conference (Mindtrek '23)* (pp. 83–94). Association for Computing Machinery. https://doi.org/10.1145/3616961.3616974

[11] Zhai, X. (2023). ChatGPT for Next Generation Science Learning. *XRDS, 29*(3), 42–46. https://doi.org/10.1145/3589649

[12] Zheng, Y. (2023). ChatGPT for Teaching and Learning: An Experience from Data Science Education. In *Proceedings of the 24th Annual Conference on Information Technology Education (SIGITE '23)* (pp. 66–72). Association for Computing Machinery. https://doi.org/10.1145/3585059.3611431