Università degli studi di Milano-Bicocca

Architetture Dati

PaRAGraph: un sistema RAG basato sul dataset Tech-QA

Autori:
Alessandro Isceri - 879309 - a.isceri@campus.unimib.it
Mattia Ingrassia - 879204 - m.ingrassia3@campus.unimib.it

Luglio 2025

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Obiettivi del progetto

L'obiettivo del progetto è quello di sviluppare un sistema Retrieval-Augmented Generation (RAG) che permette di effettuare query in linguaggio naturale su una base di conoscenza composta da documenti e simili.

Il lavoro ha inizialmente previsto una preparazione dei dati, dove il testo di ogni documento è stato pulito rimuovendo caratteri superflui come spazi di troppo, caratteri speciali usati in HTML ed errori di formattazione. Ogni documento è stato successivamente diviso in sezioni sulla base della struttura della pagina HTML.

Dopo la prima elaborazione è stato creato e popolato un Elasticsearch Vector Store Index tramite l'utilizzo della libreria LlamaIndex.

Una volta popolato il Vector Store Index, è possibile interrogarlo tramite una domanda (query) in linguaggio naturale per ricavare i documenti con maggiore rilevanza. Infine, i documenti ricavati vengono utilizzati da un Large Language Model (LLM) per generare una risposta alla domanda iniziale.

Inoltre, è stato utilizzato un LLM-as-a-judge per valutare la correttezza delle risposte generate a delle domande ricavate da un dataset di benchmark. Per farlo sono stati presi in considerazione i seguenti parametri:

- Domanda
- Risposta generata
- Risposta corretta
- Contesto utilizzato per la generazione della risposta

Infine, è stata implementata una semplice interfaccia grafica con una versione più leggera del sistema RAG per poterlo utilizzare in locale.

Il sistema lavorerà sul dataset Tech-QA, che è formato da una raccolta di pagine web di alcuni forum di IBM.

Il contesto applicativo di natura tecnico-informatica, che verrà analizzato meglio in seguito, è abbastanza complesso a causa del contenuto dei documenti; essi possono infatti contenere parole in linguaggio naturale, codice, messaggi di errore non facilmente interpretabili dal punto di vista puramente semantico.

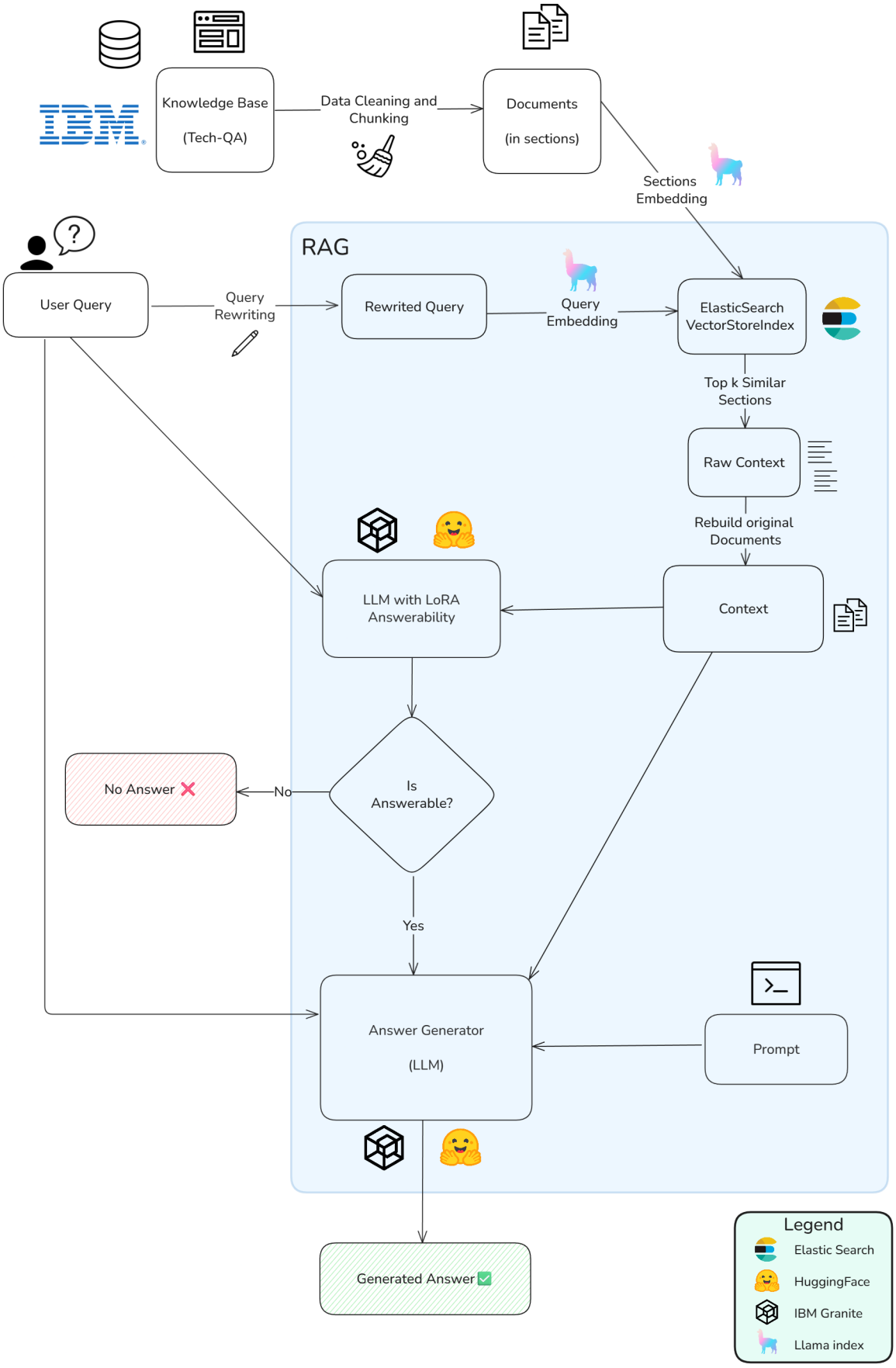

Figura 1: Workflow di PaRAGraph

1.2 Tecnologie utilizzate

1.2.1 Librerie

Di seguito vengono riportate le librerie rilevanti utilizzate per lo sviluppo del progetto:

- **BeautifulSoup**: utilizzata per l'estrazione del testo dalle pagine HTML.
- **DeepEval**: utilizzata per la validazione del modello.
- **LlamaIndex**: utilizzata per tutto il procedimento riguardante il recupero dei documenti.
- **PEFT**: utilizzata per applicare un adattatore LoRA durante la fase di generazione della risposta.
- **Transformers**: utilizzata principalmente per impiegare i modelli presenti su HuggingFace.
- **vLLM**: utilizzata per rendere più efficiente la fase di inferenza dei modelli utilizzati.

1.2.2 Database

Database impiegati nel progetto:

- **Elasticsearch**: utilizzato come Vector Store Index per memorizzare gli embedding generati.
- **MongoDB**: utilizzato come DB di documenti per memorizzare le conversazioni avviate tramite interfaccia grafica.

1.2.3 Modelli

Per quanto riguarda i vari modelli utilizzati, essi sono stati presi da HuggingFace, una piattaforma che permette di scaricare gratuitamente diversi modelli forniti dalla community o dalle aziende. In particolare, sono stati utilizzati i seguenti modelli:

- **ibm-granite/granite-embedding-30m-english**: utilizzato per la creazione e gestione degli embedding.
- **catyung/t5l-turbo-hotpot-0331**: utilizzato per riscrivere le query destinate al recupero dei documenti dal Vector Store Index.
- **naver/provence-reranker-debertav3-v1**: utilizzato per effettuare il pruning dei contesti.
- **ibm-granite/granite-3.2-8b-instruct**: utilizzato per la generazione delle risposte.
- **ibm-granite/granite-3.2-8b-lora-rag-answerability-prediction**: adattatore LoRA del modello precedente, utilizzato per determinare l'answerability della domanda.
- **ibm-granite/granite-3.2-2b-instruct**: un modello più piccolo utilizzato per la generazione delle risposte in locale.
- **Qwen/Qwen3-1.7B**: utilizzato per la validazione del sistema RAG.

1.3 Ambiente di sviluppo

Per lo sviluppo del sistema sono stati utilizzati diversi ambienti, ognuno dei quali presentava vantaggi e svantaggi in base al lavoro da svolgere.

Ambiente	**RAM**	**GPU**	**Disco**
Colab	13 GB	Nvidia T4 - 16 GB	100 GB
Kaggle	30 GB	Nvidia T4 (x2) - 16+16 GB Nvidia Tesla P100 16 GB	200 GB in input 20 GB in output
Locale	30 GB	AMD Radeon RX 6800 - 16 GB	500 GB

Tabella 1: Confronto delle specifiche tecniche dei vari ambienti utilizzati

Inizialmente, è stato usato Colab per la pulizia dei dati, in quanto era necessario lavorare con diversi file di grandi dimensioni ed è quindi risultato lo strumento più comodo, dato che offre un'integrazione nativa con Google Drive e, di conseguenza, si è rivelato utile anche per la condivisione dei file per facilitare il lavoro di gruppo.

Successivamente, è stato necessario utilizzare l'ambiente in locale per la generazione e memorizzazione degli embedding. Questa scelta è stata dettata dalla necessità di ospitare il database, siccome le opzioni offerte da Colab e Kaggle non offrivano abbastanza spazio di archiviazione né supportavano l'esecuzione di un motore di database. Inoltre, questo ambiente è stato usato per sviluppare la web-app.

Infine, l'ambiente più utilizzato in fase di sviluppo è stato Kaggle, che offre la potenza di calcolo maggiore, gratuita per 30 ore a settimana, e ha permesso l'esecuzione di tutti i task che richiedevano l'impiego di modelli di grandi dimensioni, come ad esempio la generazione delle risposte e la valutazione delle stesse.

2 Descrizione dei dataset

2.1 Tech-QA

2.1.1 Contenuto del dataset

Il dominio applicativo scelto riguarda il supporto tecnico agli utenti.
Nello specifico, il dataset, descritto in [1], è composto da una serie di pagine web, ottenute tramite crawling dai forum IBM Developer e IBM DeveloperWorks, le quali contengono problemi riscontrati da utenti reali con risposte verificate che sono contenute nelle IBM Technotes, ovvero documenti che trattano uno specifico problema tecnico.
Tra i contenuti della repository, è stato utilizzato il file `full_technote_collection.txt`, un file di 33 righe, dove ogni riga è composta da un array di oggetti JSON, ciascuno contenente una pagina del forum.
In totale sono presenti 801 997 oggetti che rappresentano altrettante pagine del forum. Ogni documento presenta una struttura analoga alla seguente:

- **id**: id unico della pagina del forum.
- **content**: codice HTML della pagina.
- **title**: titolo grezzo della pagina.
- **text**: testo grezzo contenuto nel `<body>` della pagina HTML.
- **metadata**: oggetto JSON contenente dei metadati della pagina.
  - **sourceDocumentId**: la maggior parte delle volte coincide con l'id.
  - **date**: data dell'ultima modifica.
  - **productName**: nome del prodotto descritto dalla domanda.
  - **productId**: id del prodotto, utile per disambiguare o referenziare i prodotti.
  - **canonicalUrl**: URL originale della pagina, può non corrispondere a quello attuale.

Di seguito, viene riportato un esempio di documento presente nel dataset:

```
{
    "id": "swg1PK13560",
    "content": "<div class=\"ibm-card\"> ... </div>",
    "title": "IBM PK13560: ILLEGAL MONITOR STATE EXCEPTION RAISED IN...",
    "text": "AIX SUBSCRIBE\n You can track all active APARs for this... ",
    "metadata": {
      "sourceDocumentId": "swg1PK13560",
      "date": "2006-06-06",
      "productName": "Runtimes for Java Technology",
      "productId": "SSNVBF",
      "canonicalUrl": "http://www.ibm.com/support/docview.wss?uid=swg1PK13560"
    }
}
```

Listing 1: Struttura di un oggetto JSON contenuto nel file `full_technote_collection.txt`

2.1.2 Analisi esplorativa

Per ottenere maggiori informazioni sul contenuto dei dati utilizzati, è stato deciso di effettuare delle analisi.
In primo luogo, è stata analizzata la distribuzione temporale dell'ultima modifica di ogni pagina web.

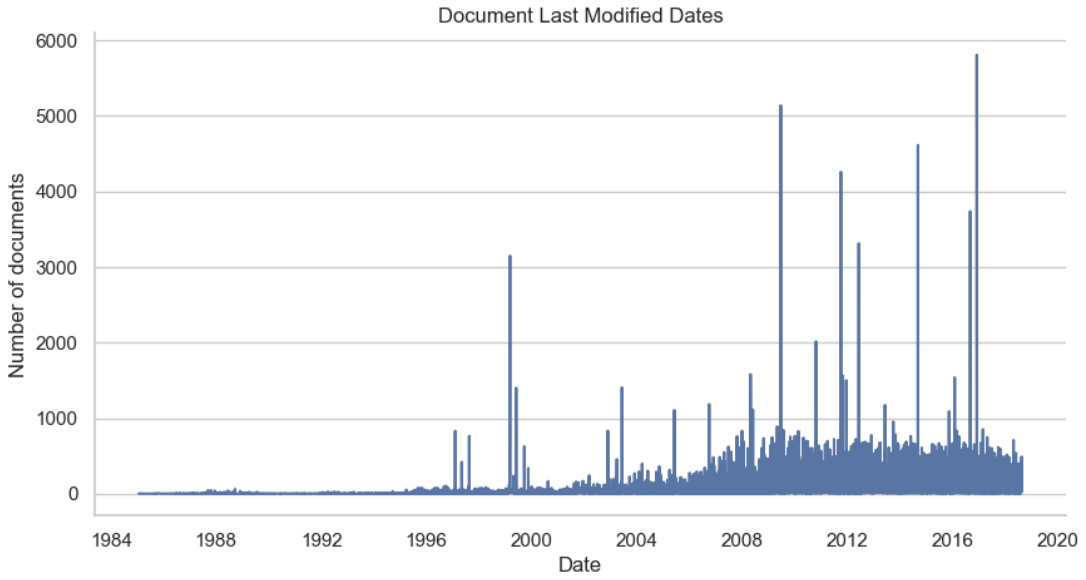

Figura 2: Distribuzione temporale dell'ultima modifica di ogni documento

Dall'immagine 2 si può notare che i documenti raccolti, per quanto riguarda l'ultima modifica, spaziano tra gli anni 1985 e 2019, l'anno in cui è stato creato il dataset.
Inoltre, è possibile affermare che a partire dall'anno 2000 in poi, l'attività sui forum è aumentata notevolmente; questo indica che un sistema RAG che lavora con questi documenti dovrebbe utilizzare i dati più recenti siccome vengono modificati molto spesso.

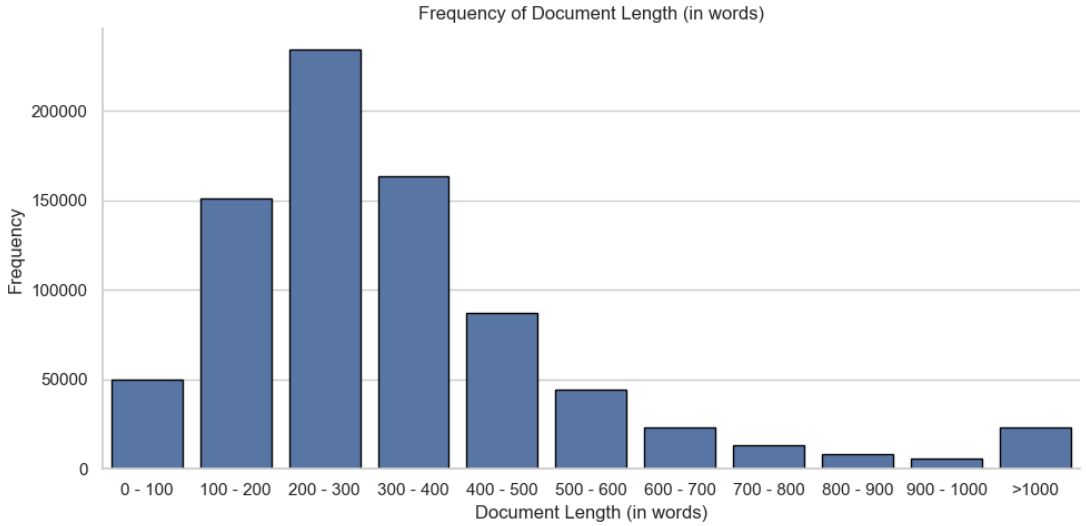

Figura 3: Distribuzione della lunghezza dei documenti

Dall'immagine 3 si osserva che la maggior parte dei documenti ha un numero di parole compreso tra 100 e 400, quindi non si tratta di documenti eccessivamente lunghi; tuttavia, nel dataset,

sono presenti alcuni documenti molto lunghi (più di 1000 parole) e il documento più lungo raggiunge le 87 013 parole. La mediana è pari a 285, come prevedibile osservando il grafico.

Infine, per studiare la copertura dei vari argomenti, i documenti sono stati raggruppati in 18 categorie. Le categorie sono state individuate grazie alle parole chiave più frequenti presenti nei metadati.

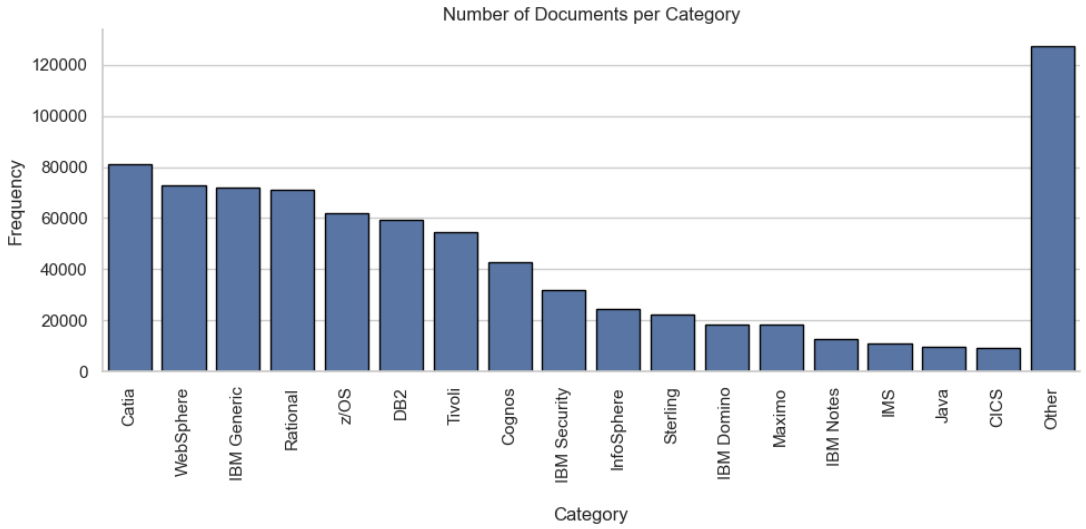

Figura 4: Distribuzione dei documenti per categoria

Dal grafico 4 si può osservare che 'Catia', 'WebSphere' e le varie sottosezioni comprese in 'IBM Generic' sono argomenti molto diffusi nei documenti. Esistono anche molti temi citati poche volte nei forum, questi ricadono nella categoria 'Other'.

Questa analisi è utile perché permette di capire quali argomenti sono maggiormente trattati nelle pagine e, di conseguenza, quali hanno più informazioni a disposizione. Ad esempio, una query riguardante 'WebSphere' dovrebbe essere più facilmente rispondibile rispetto a una richiesta di aiuto su 'Java', siccome la categoria 'WebSphere' contiene un elevato numero di documenti.

2.2 Dataset di benchmark

2.2.1 Contenuto del dataset

Per la validazione del sistema è stato utilizzato un insieme di 910 domande e risposte fornite da Nvidia, studiate apposta per la valutazione di sistemi RAG costruiti a partire dal dataset Tech-QA.
Ogni oggetto è descritto dai seguenti attributi:

- **id**: id della domanda.
- **question**: testo della domanda.
- **answer**: testo della risposta (ground truth).
- **is_impossible**: booleano che indica se è possibile rispondere alla domanda in base alle informazioni disponibili.

- **contexts**: un array contenente oggetti JSON che descrivono il contesto da cui è stata ricavata la risposta, ogni oggetto è descritto dagli attributi:
  - **filename**: nome del file.
  - **text**: testo contenuto nel file.

Di seguito viene riportato un oggetto di esempio.

```
{
    "id": "TRAIN_Q000",
    "question": "User environment variables no longer getting picked up... ?",
    "answer": "To work around the issue, set environment variables that...",
    "is_impossible": false,
    "contexts": [
        {
            "filename": "swg21996508.txt",
            "text": "Title: IBM STREAMS 4.1.1.1 and 4.1.1.2  JOBS DO NOT..."
        }
    ]
}
```

Listing 2: Struttura di una domanda utilizzata per il benchmark

2.2.2 Analisi esplorativa

Per quanto riguarda il dataset di benchmark, sono state effettuate delle analisi sui contenuti delle domande e sulla loro lunghezza, per avere una panoramica su di esse.
Dalla figura 5 si possono vedere le parole che compaiono con maggiore frequenza nelle domande. Da questo grafico sono state escluse le stop words, in modo tale da poter analizzare gli argomenti principali trattati nelle domande. Si noti, inoltre, che molte parole sono contenute nelle categorie ricavate in 2.1.2, come ad esempio 'IBM', 'WebSphere' e 'Security'.

Figura 5: Parole più diffuse nelle domande

Secondariamente, dalla figura 6 è possibile capire quali siano i pattern ricorrenti, riguardanti le prime 3 parole, nella struttura delle domande. Spesso le parole chiave sono precedute dalla classica struttura sintattica delle domande; ciò indica una buona qualità di formulazione delle domande.

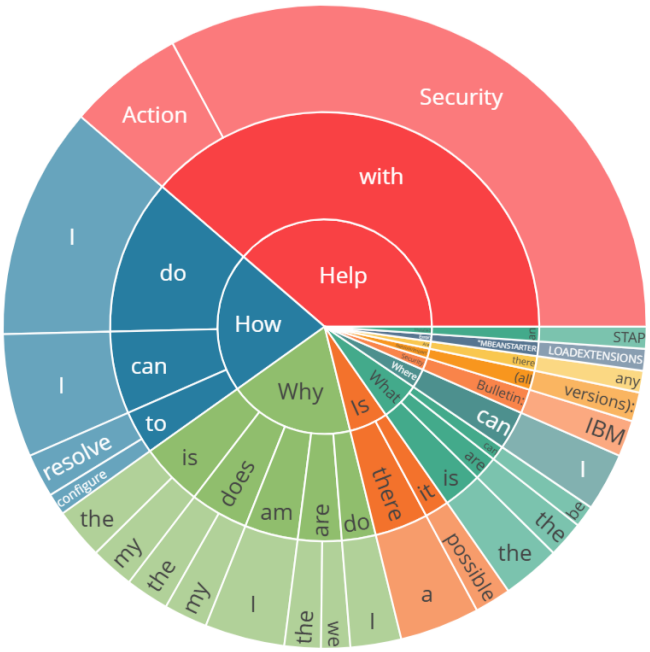

Figura 6: Pattern di parole iniziali più frequenti nelle domande

Infine, come si può vedere dalla figura 7 la maggior parte delle domande contenute nel dataset ha una lunghezza inferiore a 75 parole; ciò significa che le domande sono mediamente lunghe. Esistono addirittura alcune domande più articolate che contano più di 150 parole. Le domande lunghe potrebbero mettere in difficoltà il modello usato per la generazione delle risposte a causa della loro complessità intrinseca.

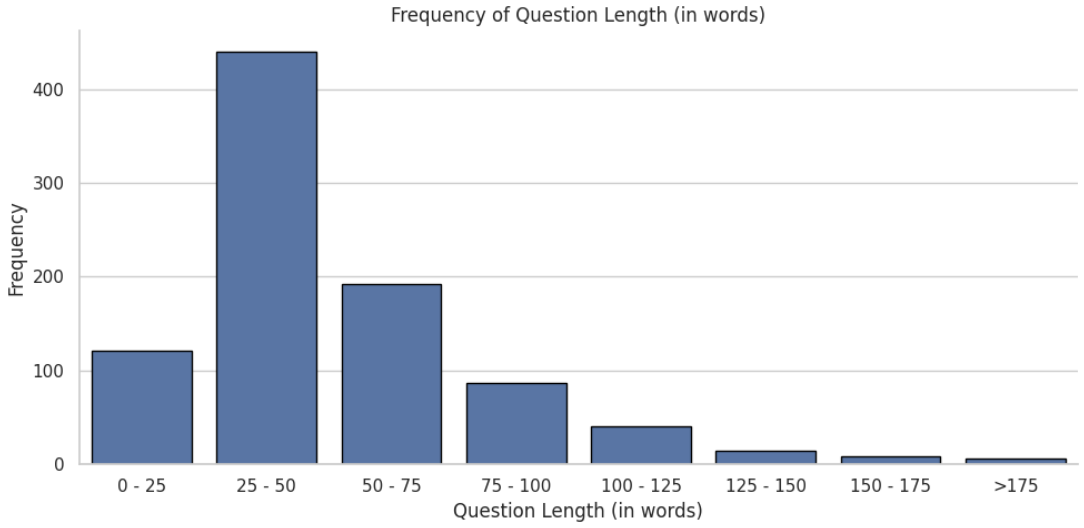

Figura 7: Distribuzione delle lunghezze delle domande

3 Preparazione dei dati

L'obiettivo della preparazione dei dati è quello di ottenere embedding di buona qualità. Per farlo, la prima parte del lavoro consisteva nel pulire i dati a disposizione in modo tale da migliorarne la qualità e, successivamente, dividere ogni documento in sezioni, con l'obiettivo di creare una struttura adatta alla creazione degli embedding.

3.1 Pulizia dei dati

Alcuni documenti presentavano errori di formattazione, come ad esempio tag `<h2>` non chiusi oppure piccole inconsistenze tra `text` e `content`, che sono stati risolti in prima battuta.
Inizialmente, per pulire l'attributo `text` sono stati eliminati tutti gli spazi in eccesso come spazi multipli, `\n, \t`.
Per gestire al meglio le informazioni contenute nel campo `content`, ovvero nella pagina HTML, è stata utilizzata la libreria BeautifulSoup, che offre diversi metodi per manipolare ed effettuare operazioni sui contenuti dei vari tag HTML. Nello specifico, la libreria è risultata utile per facilitare la creazione delle sezioni, in particolare sono state eseguite le seguenti operazioni:

- sostituzione dei tag `<br>` con un semplice spazio.
- sostituzione dei tag `<a href="link">text</a>` con `text [link]`.
- estrapolazione dei contenuti dei tag `<h2>` per ottenere i titoli delle sezioni.

Il testo ottenuto a partire dai tag `<h2>` viene pulito eliminando gli spazi in eccesso; tramite espressioni regolari, sono stati rimossi caratteri speciali come `&XA0;`, un carattere che compariva spesso nelle pagine HTML, che rappresenta uno spazio e causava alcuni problemi nelle fasi successive. In seguito, siccome i testi contenevano con una certa frequenza caratteri particolari quali: `["(", ")", "+", "*", "?", "[", "]", "\", "\\"]`, è stato effettuato l'escaping degli stessi dato che, in un successivo momento, verranno utilizzate le espressioni regolari per dividere il testo in sezioni.
Infine, sono stati eliminati gli spazi all'inizio e alla fine della stringa, e il testo è stato riscritto in maiuscolo. Il testo risultante dalla pulizia del contenuto dei tag `<h2>` è stato utilizzato come titolo delle varie sezioni.

3.2 Divisione in sezioni

Una volta individuati tutti i titoli delle sezioni di un documento, per trovare il testo di una sezione viene effettuata una ricerca con match esatto tra il titolo della sezione successiva e l'attributo `text`. Se il match non va a buon fine (talvolta il testo è mal formattato), la ricerca viene effettuata senza case sensitiveness. In questo modo, trovando la posizione di inizio della sezione successiva, è possibile determinare la posizione finale della sezione corrente. La prima sezione partirà dalla posizione 0 nel testo e l'ultima terminerà con la posizione dell'ultimo carattere del testo.
Dato che per generare gli embedding viene utilizzato un modello che supporta un contesto formato al massimo da 512 parole, è stato deciso di dividere ulteriormente le sezioni troppo lunghe, così da evitare una perdita di informazioni dovuta al possibile troncamento.

Informazioni sulle sezioni	
Numero iniziale di sezioni	7 276 540
Numero di sezioni da troncare	37 920
Sezione più lunga (in parole)	86 923
Numero finale di sezioni	7 362 780

Tabella 2: Statistiche sulle sezioni dei documenti

Per ogni documento viene quindi creato un array `sections`, il quale contiene la suddivisione in sezioni, ognuna delle quali è costituita dai seguenti attributi:

- **title**: titolo della sezione.
- **text**: testo della sezione (comprende il titolo).
- **start**: posizione iniziale della sezione.
- **end**: posizione finale della sezione.

Una volta diviso il documento in sezioni, viene creato un oggetto JSON strutturato come segue:

```
{
    "id": "swg1PK13560",
    "title": "IBM PK13560: ILLEGAL MONITOR STATE EXCEPTION RAISED IN JIT...",
    "text": "AIX SUBSCRIBE You can track all active APARs for this...",
    "metadata": {
        "sourceDocumentId": "swg1PK13560",
        "date": "2006-06-06",
        "productName": "Runtimes for Java Technology",
        "productId": "SSNVBF",
        "canonicalUrl": "http://www.ibm.com/support/docview.wss?uid=swg1PK13560"
    },
    "sections": [
        {
            "title": "APAR STATUS",
            "text": "AIX SUBSCRIBE You can track all active APARs for ... ",
            "start": 0,
            "end": 104
        },
        //   ...
        {
            "title": "APPLICABLE COMPONENT LEVELS",
            "text": "APPLICABLE COMPONENT LEVELS * R420 PSN UP...",
            "start": 1125,
            "end": 1208
        }
    ]
}
```

Listing 3: Struttura di un singolo documento diviso in sezioni

3.3 Creazione dei nodi

Per la creazione dei nodi è stata utilizzata la libreria LlamaIndex.
Un nodo rappresenta una fetta del documento originale, dove un documento può essere un PDF, l'output di una chiamata API, oppure dei dati ottenuti da un'interrogazione ad un database ecc...
LlamaIndex permette la creazione dei nodi a partire dai documenti impostando una dimensione dei chunk fissa; tuttavia, è stato deciso di non utilizzare questa funzionalità per poter ottenere dei nodi (e relativi embedding) più significativi a livello semantico, siccome ogni documento è stato precedentemente diviso in sezioni.
La soluzione studiata prevede di avere un nodo per ogni sezione creata, in modo tale che il testo del nodo sia uguale al testo della sezione; inoltre, il nodo contiene dei metadati:

- **document_title**: il titolo del documento da cui proviene la sezione.
- **document_id**: l'id del documento da cui proviene la sezione.
- **section_title**: il titolo della sezione.

Infine, sono state definite delle relazioni per ogni nodo; in particolare, ciascun nodo contiene un riferimento al nodo che rappresenta la sezione precedente e a quello che contiene la sezione successiva.
Ovviamente, il nodo iniziale non avrà un predecessore e il nodo finale non avrà un successore.

3.4 Popolamento dell'Elasticsearch Vector Store Index

I Vector Store Index sono delle strutture che permettono di memorizzare in maniera efficiente l'embedding di ogni nodo.
Essi vengono spesso utilizzati nei sistemi RAG, e LlamaIndex permette la creazione di un Vector Store Index a partire da un elenco di nodi, memorizzandoli e indicizzandoli.
Per la creazione degli embedding è stato utilizzato il modello `granite-embedding-30m-english` fornito da HuggingFace.
Purtroppo, non avendo a disposizione una grande potenza di calcolo, è stato scelto un modello con un numero ridotto di parametri (30 milioni), così da poterlo eseguire in locale.
Nonostante la scelta di un modello ridotto e l'utilizzo di una scheda video dedicata, i tempi di creazione degli embedding sono stati decisamente elevati (circa 15 ore); l'utilizzo di altri modelli con più parametri avrebbe aumentato ulteriormente i tempi di esecuzione.
Infine, per garantire la persistenza dei dati è stato utilizzato Elasticsearch, un database che permette di memorizzare facilmente un Vector Store Index.
Date le elevate dimensioni del Vector Store Index generato (circa 64 GB) anch'esso è stato memorizzato in locale.

4 Recupero dei documenti

Per rispondere alle domande dell'utente, il sistema avrà bisogno di accedere ai documenti che contengono informazioni relative alla domanda.
Per ottenere i documenti più rilevanti alla domanda, vengono eseguite le seguenti operazioni:

- Query Rewriting
- Document Retrieval
- Context Pruning

4.1 Query Rewriting

Per prima cosa, una volta che il sistema RAG viene interrogato, viene effettuata un'operazione di riscrittura della domanda in modo tale da costruire una query ottimizzata che recuperi i documenti di riferimento dal Vector Store Index.
Per effettuare questa operazione viene utilizzato il modello `t5l-turbo-hotpot-0331` che riscrive la query mettendo in evidenza le parole chiave e togliendo le informazioni superflue al recupero dei documenti di riferimento.
Ad esempio, la seguente domanda viene riscritta come segue:

"Can I apply a TIP 2.2 fix pack directly to a TIP 2.1 installation?"

$$\downarrow$$

"TIP 2.2 fix pack; TIP 2.1 installation;"

In questo modo, per ogni nuova domanda si esegue lo split del testo, e si ottengono diverse sub-queries, una per ogni ';'.
Rispetto all'esempio precedente, le sub-queries ottenute sono: "TIP 2.2 fix pack" e "TIP 2.1 installation".

4.2 Document Retrieval

Viene in seguito effettuata un'interrogazione al Vector Store Index per ogni sub-query, che restituisce i migliori documenti a seconda dei parametri scelti:

- **top_k**: il numero massimo di documenti che ogni interrogazione deve restituire.
- **min_p**: il valore minimo che deve avere lo score di una sezione per far si che essa venga presa in considerazione.

Lo score di una sezione è un valore compreso tra 0 e 1, calcolato attraverso l'uso della `cosine similarity` tra l'embedding della query e l'embedding del documento. Più lo score si avvicina a 1, più i contenuti del documento sono simili a quelli citati nella query e viceversa.
Per ogni nodo trovato, che corrisponde a una sezione, viene ricostruito interamente il documento originale per fare in modo che si abbiano tutte le informazioni necessarie durante le fasi successive; ad esempio, una domanda posta da un utente al sistema RAG potrebbe avere una similarità elevata con una domanda posta sui forum, ma non con la sua risposta.
Per farlo, per ogni nodo recuperato dal retriever si scorrono i predecessori finché non si arriva al nodo iniziale, dopodiché si scorrono tutte le sezioni in avanti, concatenando il contenuto di ognuna di esse, così da ottenere il testo originale del documento.

In questo modo, alla domanda effettuata al RAG viene associata una lista di contesti formati nel seguente modo:

```
[
    {
        "document_id": "swg21960785",
        "document_title": "IBM 5.3.0.6-ISS-XGS-All-Models-IF0003 - United States",
        "sections": [
            {
                "section_title": "TECHNOTE (FAQ)",
                "section_text": "5.3.0.6-ISS-XGS-All-Models-IF0003 ..."
            },
            {
                "section_title": "QUESTION",
                "section_text": "QUESTION What fixes are included in ... "
            },
            {
                "section_title": "ANSWER",
                "section_text": "ANSWER Abstract IBM Security Network ..."
            }
        ]
    },
    {
        "document_id": "swg21903749",
        "document_title": "IBM 5.3.0.6-ISS-XGS-All-Models-IF0001 - United States",
        "sections": [
            {
                "section_title": "FIX README",
                "section_text": "IBM Security Network Protection; XGS; ..."
            },
            {
                "section_title": "ABSTRACT",
                "section_text": "ABSTRACT What fixes does 5.3.0.6-ISS-XGS..."
            },
            {
                "section_title": "CONTENT",
                "section_text": "CONTENT This document discusses the fixes..."
            }
        ]
    },

    // ...

]
```

Listing 4: Struttura dei contesti recuperati

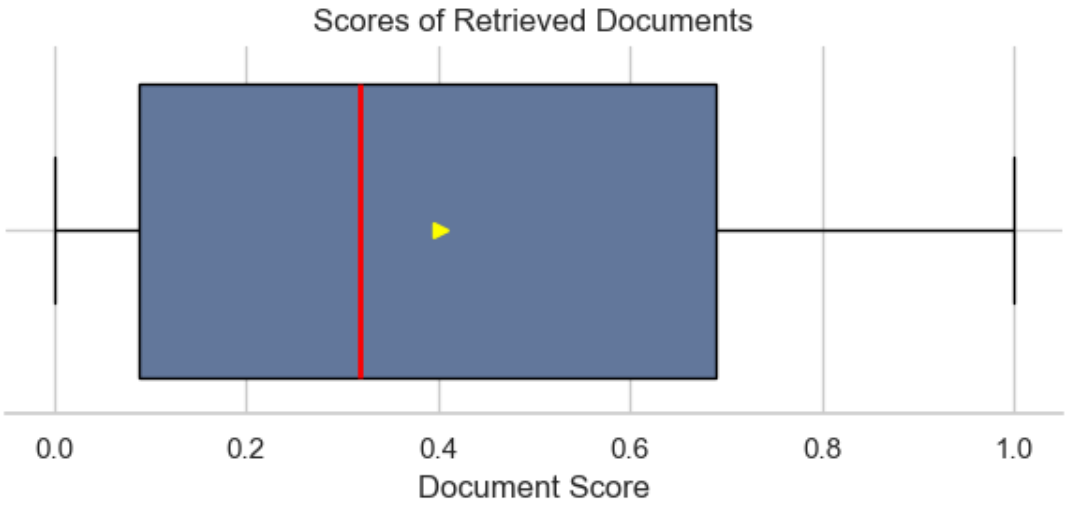

Figura 8: Distribuzione degli score dei documenti recuperati

Come si può notare dal grafico, la distribuzione degli scores spazia principalmente dal valore di 0.1 a 0.7, con una media di circa 0.4.
Questo significa che, in media, la similarità tra la query e i documenti recuperati non è molto alta; ciò non è una sorpresa, data la complessità dei temi trattati e la semantica prettamente tecnico-informatica dei documenti. Infatti, il linguaggio specifico di questo ambito può creare problemi, sia per la creazione degli embedding, sia per la generazione delle risposte, soprattutto se non si usano modelli su cui viene eseguito un Fine Tuning per questo contesto applicativo.

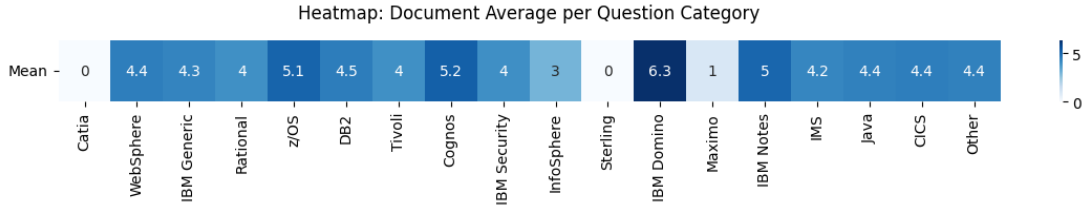

Figura 9: Numero medio di documenti per categoria di domanda

Per costruire questo grafico, è stato inizialmente utilizzato il modello `cross-encoder/nli-deberta-v3-lar` per classificare ogni domanda con una delle categorie individuate in 2.1.2. Successivamente, è stata calcolata la media del numero di documenti ottenuti per ogni categoria. Le domande di benchmark non coprivano tutte le categorie; infatti, alcune entry della heatmap sono pari a 0, questo indica l'assenza di domande appartenenti a tali classi. In media si vede che per la maggior parte delle categorie si tende ad avere almeno 4 documenti per domanda, costruendo una buona base di contesti per la generazione delle risposte.

4.3 Context Pruning

Durante lo sviluppo, sono stati effettuati dei tentativi volti all'implementazione di una fase di context pruning, con lo scopo di migliorare la qualità dei documenti e ridurre il costo computazionale in fase di generazione delle risposte.
Questa operazione consiste nel rimuovere le parti superflue, mal formattate o di bassa rilevanza per la domanda, diminuendo la dimensione del contesto senza perdere informazioni importanti.

Per farlo, è stato utilizzato il modello `naver/provence-reranker-debertav3-v1`, un modello di piccole dimensioni ottimizzato per questo task.
Di seguito viene riportato un esempio di context pruning, dove una sezione viene potata:

"ANSWER SSP is not vulnerable to this attack. Since SSP only uses Elliptic Curve Diffie-Hellman Encryption (ECDHE) for TLS processing, it is fine. And since the Maverick toolkit requires at least 1024 bit keys, the SFTP processing is okay as well"

↓

"ANSWER SSP is not vulnerable to this attack. Since SSP only uses Elliptic Curve Diffie-Hellman Encryption (ECDHE) for TLS processing, it is fine."

Purtroppo, il context pruning in questo caso non si è rivelato uno strumento utile: i risultati del sistema tendevano a peggiorare lievemente rispetto a quando venivano utilizzati i documenti originali.
Probabilmente, i cattivi risultati sono dovuti al fatto di tagliare troppe sezioni o eliminare troppe informazioni utili; presumibilmente ciò accadeva perché il modello usato non era specializzato per lavorare su documenti di tipo tecnico-informatico e aveva un numero ridotto di parametri.
È stato quindi deciso di non utilizzare questa tecnica nel sistema finale.

5 Generazione delle risposte

5.1 Answerability prediction

Come ultimo passaggio, prima della effettiva generazione della risposta, abbiamo usato un modello di predizione della answerability della domanda. Per fare ciò è stato utilizzato il modello `ibm-granite/granite-3.2-8b-instruct` con la sua adattazione LoRA per l'answerability `ibm-granite/granite-3.2-8b-lora-rag-answerability-prediction`.
Il modello LoRA (Low-Rank Adaption) consiste in un adattamento del modello base ottimizzato per il task di answerability, ovvero una classificazione binaria zero-shot della domanda.
Date le dimensioni elevate del modello e la poca potenza computazionale a disposizione, è stato necessario quantizzare il modello per poterlo utilizzare.
Questo modello, dati come parametri in ingresso la domanda e i contesti recuperati, semplicemente contrassegna tale domanda come `"answerable"` o `"unanswerable"`.
Può ritornare utile per generare le risposte solo in caso di effettiva rispondibilità della domanda presa in esame, evitando di generare risposte potenzialmente inutili o errate.

5.1.1 Risultati ottenuti

Sono state misurate le performance del modello attraverso il confronto con i dati del dataset di validation (2.2.1).
Sono stati effettuati diversi tentativi per cercare di ottenere le migliori prestazioni possibili; attraverso l'impiego di diversi modelli di text reranking e text generation si è cercato di assegnare uno score alla rispondibilità di una domanda, senza successo. Sono state inoltre sperimentate numerose configurazioni per ogni modello, soffermandosi in particolare sulle combinazioni degli iperparametri `top_k` e `min_p`.

Per quanto riguarda la classificazione, le istanze contrassegnate come positive (`"unanswerable"`) rappresentano le domande a cui non è possibile rispondere, mentre le istanze appartenenti alla classe negativa (`"answerable"`) coincidono con le domande a cui è possibile rispondere.
I migliori risultati, riportati di seguito, sono stati ottenuti con un `top_k` pari a 20 e un `min_p` di 0.45.

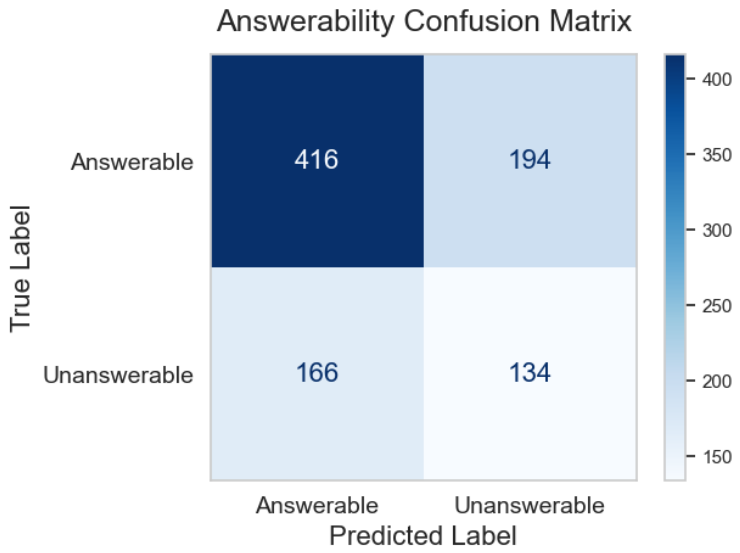

Figura 10: Matrice di confusione della answerability

Metrica	Valore
Accuracy	60.44 %
Precision unanswerable	40.85 %
Recall unanswerable	44.67 %
F1-Score unanswerable	42.68 %
Precision answerable	71.48 %
Recall answerable	68.20 %
F1-Score answerable	69.80 %
Macro F1-Score	56.24 %

Tabella 3: Risultati della predizione di answerability di PaRAGraph

Come si può notare, il modello riesce a prevedere correttamente se ad una domanda si può rispondere circa il 60% delle volte. Inoltre, si può vedere dalle metriche di precision, recall, F1-score e anche dall'elevato numero di falsi positivi, che il modello fatica a classificare correttamente la classe `"unanswerable"`.
Il modello si comporta meglio con la classe negativa: ottenendo un F1-score del 70% sulla classe `"answerable"`.

Sistema	Macro F1-Score
SQuAD 2.0 – FT	48.39 %
SQuAD 2.0 + FT	54.05 %
NQ – FT	48.39 %
NQ + FT	55.31 %
TAP v0.1	52.67 %
PaRAGraph	56.24 %

Tabella 4: Comparazione di PaRAGraph con altri sistemi di riferimento

Nella tabella 4 vengono confrontati i risultati ottenuti da PaRAGraph con quelli riportati nel paper dove viene descritto il dataset Tech-QA [1].
I modelli utilizzati nell'articolo sono 5; sia `SQuAD 2.0` che `NQ` sono varianti di `BERTLARGE`, un language model che presenta layer aggiuntivi a seconda del dataset utilizzato per il training, rispettivamente `SQuAD 2.0` e `NQ`.
La sigla FT indica se il modello è stato fine-tunato sul dataset Tech-QA (+ FT indica che è stato fine-tunato, - FT indica che non è stato fine-tunato). Invece, il modello `TAP v0.1` è costituito da un modulo di document ranking e da un selettore delle risposte, entrambi basati su un modello pre-trained di `BERT` (small). Se lo score del primo componente supera una certa soglia, la domanda è classificata come `"answerable"` e viceversa. Questo modello è stato allenato sul dataset `HotPotQA`.
Come si può notare dalla tabella 4, i modelli su cui viene eseguito il Fine Tuning ottengono un F1-Score medio più alto. Su PaRAGraph non è stato eseguito il Fine Tuning; tuttavia, il sistema è in grado di ottenere un Macro F1-Score più alto dei modelli presentati nell'articolo.
L'ambiente di sviluppo utilizzato da IBM per lo studio precedentemente citato è costituito da un sistema che comprende 128 GB di RAM, 64 GB di archiviazione e due GPU V100, dotate di 16 GB di VRAM ciascuna.

Nel complesso, la loro dotazione era migliore di quella utilizzata in questo progetto, che viene riportata a 1.3.

5.2 Answer generation

Per generare le risposte è stato usato il modello `ibm-granite/granite-3.2-8b-instruct` in combinazione con la libreria `vLLM` che permette di sfruttare al massimo la parallelizzazione su GPU multiple durante la fase di inferenza.
Per eseguire il codice di questa parte è stato principalmente utilizzato Kaggle 1.3, che permette di usare diverse GPU.
In particolare, grazie a queste tecnologie e ulteriori ottimizzazioni, è stato possibile generare le risposte delle 610 domande classificate come `"answerable"` in circa un'ora; precedentemente, utilizzando le classiche librerie di HuggingFace e una sola GPU, il tempo di generazione delle risposte poteva superare diverse ore di esecuzione.
Per la generazione delle risposte è stata utilizzata la seguente configurazione:

Parametro	**Valore**	**Descrizione**
`max_len`	16384	Numero massimo di token in ingresso, limitato per evitare crash dovuti all'esaurimento della memoria disponibile.
`temperature`	0.05	Creatività del modello, si è scelto un valore basso di proposito, per evitare che il modello generi risposte troppo fantasiose, limitandolo a rispondere utilizzando solamente i documenti ricavati dal contesto.
`max_new_tokens`	8192	Numero di token in uscita, scelto con l'obiettivo di generare risposte piuttosto complete senza porre limiti troppo stringenti.

Tabella 5: Configurazione del modello `Granite-3.2-8b` per la generazione delle risposte

Per dare indicazioni ben definite al modello è stato creato un prompt ad hoc, strutturato come segue:

```
You are Granite, an AI developed by IBM. You are a helpful RAG (Retrieval-Augmented
Generation) system designed to answer user queries based only on the content of the
retrieved documents.

### Key Instructions:
1. **Only Use Retrieved Documents for Answers**: Provide an answer using specific,
direct information in the retrieved documents.
2. **No Speculation**: Do not try to make inferences or use general knowledge.
3. **Do Not Mention Documents**: Never refer to, mention, or include any details
about the documents in your response. Do not say things like "According to the
document," or "The document indicates...". Simply provide the answer.
4. **No Extra Information**: Do not elaborate or provide additional context.
```

Il modello, prendendo in ingresso il prompt, la query e i documenti inerenti alla domanda (ottenuti dal sistema di document retrieval 4.2), è in grado di generare una risposta, che viene salvata in un documento JSON che contiene le seguenti informazioni:

- **user_input**: domanda originale.
- **retrieved_contexts**: documenti forniti al modello per generare la risposta.
- **response**: risposta generata dal modello.
- **reference**: risposta presa dal dataset di benchmark da considerare come ground truth.

Di seguito, viene riportato un oggetto a titolo esemplificativo:

```
[
    {
        "user_input": "User environment variables no longer getting ...",
        "retrieved_contexts": [
            " FLASH (ALERT) ABSTRACT In Streams 4.1.1.1 and 4.1.1.2 ...",
            " TECHNOTE (TROUBLESHOOTING) PROBLEM(ABSTRACT) When ...",
            " TECHNOTE (TROUBLESHOOTING) PROBLEM(ABSTRACT) Error ...",
            " TECHNOTE (TROUBLESHOOTING) PROBLEM(ABSTRACT) Unable ...",
            " APAR STATUS * CLOSED AS PROGRAM ERROR. ERROR ..."
        ],
        "response": "After upgrading to Streams 4.1.1.1 or 4.1.1.2 ...",
        "reference": "To work around the issue, set environment ..."
    },

    // ...

]
```

Listing 5: Esempio di oggetto contneuto nel file `generated_answers.json`

6 Validazione dei risultati

6.1 Metriche di valutazione

Per valutare la qualità delle risposte generate dal sistema RAG prodotto è stata utilizzata la libreria DeepEval che fornisce alcune metriche utili per la valutazione di LLM o di sistemi che si basano su di essi; in particolare, questa libreria offre 4 metriche basate sull'utilizzo di un LLM-as-a-judge per la valutazione di sistemi RAG:

- **Answer Relevancy**: valuta quanto la risposta generata è rilevante rispetto alla domanda effettuata.
- **Contextual Precision**: valuta se i documenti recuperati più rilevanti per la domanda dell'utente hanno un ranking più alto di quelli irrilevanti.
- **Contextual Recall**: valuta a che livello il contesto generato è allineato con la risposta attesa (ground truth).
- **Contextual Relevancy**: valuta la rilevanza generale dell'informazione contenuta nei documenti recuperati rispetto all'input dell'utente.

Per validare la bontà del modello implementato, è stato deciso di generare risposte a un campione di 354 domande prese casualmente dalle 610 domande classificate come `"answerable"`, in modo tale da avere un buon numero di esempi per valutare il modello.
I risultati ottenuti dal sistema e i grafici risultanti vengono riportati di seguito:

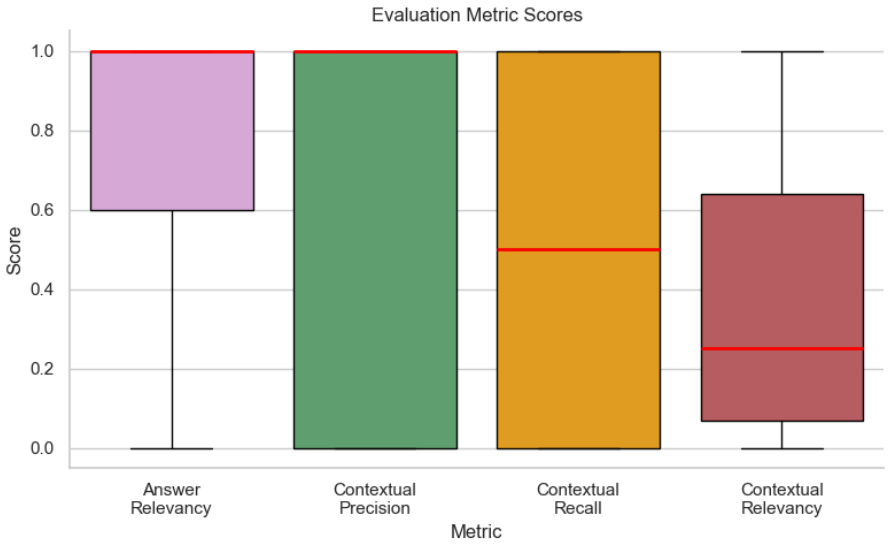

Figura 11: Distribuzione degli score ottenuti per ogni metrica

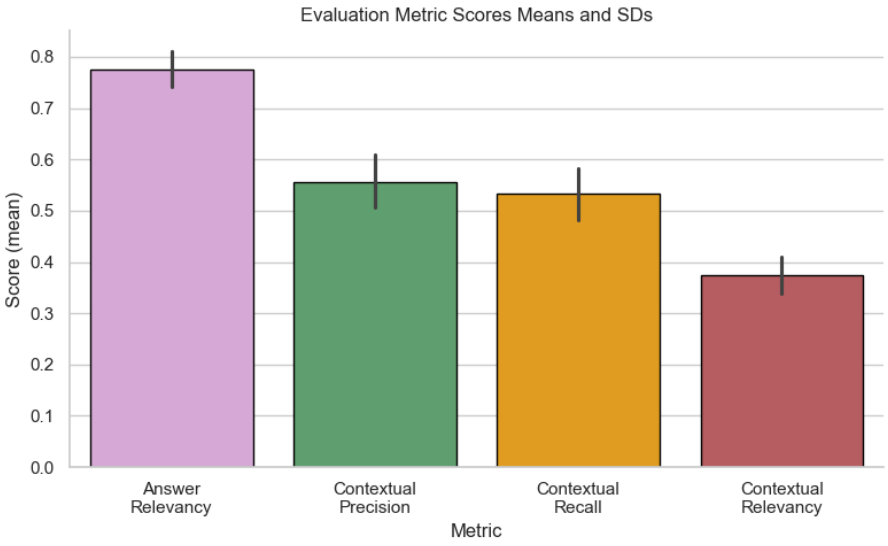

Figura 12: Media e deviazione standard di ogni metrica

Dalle figure 11 e 12 si può osservare che il sistema prodotto ottiene degli ottimi risultati per quanto riguarda la metrica di Answer Relevancy con una media di quasi 0.8 ed una deviazione standard abbastanza ridotta che indica una discreta stabilità degli scores. Sono stati raggiunti dei buoni risultati per le metriche Contextual Precision e Contextual Recall con una media intorno allo 0.6; la Contextual Precision ha però presentato una deviazione standard abbastanza elevata. Infine, per quanto riguarda la Contextual Relevancy sono stati ottenuti i risultati peggiori, con una mediana pari solamente a 0.25, una media che si assesta intorno allo 0.37 ed una deviazione standard ridotta ad indicare una bassa qualità dei risultati.

6.1.1 Answer Relevancy

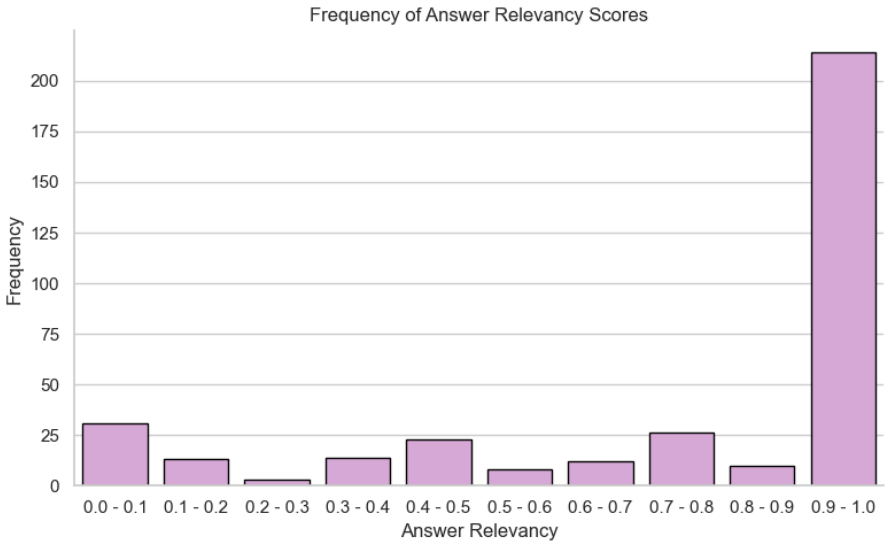

Figura 13: Distribuzione degli score di Answer Relevancy

A supporto di quanto detto in precedenza, dalla figura 13 si nota che la metrica di Answer Relevancy ottiene un elevato numero di score appartenenti all'intervallo finale. I risultati otte-

nuti per questa metrica indicano che PaRAGraph produce il più delle volte risposte rilevanti rispetto alle domande effettuate.

6.1.2 Contextual Precision

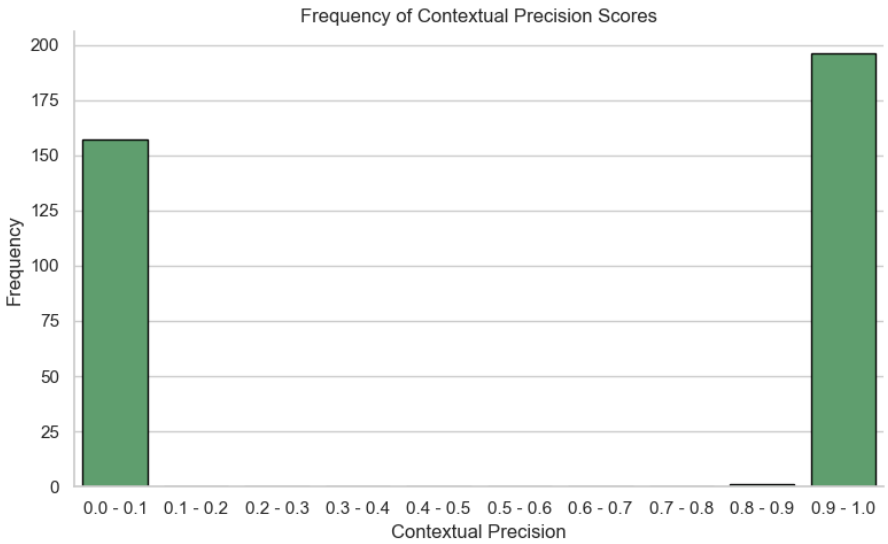

Figura 14: Distribuzione degli score di Contextual Precision

Come si può vedere dalla figura 14, questa metrica ha un comportamento estremamente altalenante; infatti assume principalmente valori compresi tra lo 0% e il 10% o tra il 90% e il 100%. Questo potrebbe indicare che i documenti recuperati non sono sempre nell'ordine ottimale per la generazione della risposta. Potenzialmente, avendo a disposizione una potenza di calcolo maggiore, sarebbe stato possibile generare degli embedding di più alta qualità e ottenere risultati migliori.

6.1.3 Contextual Recall

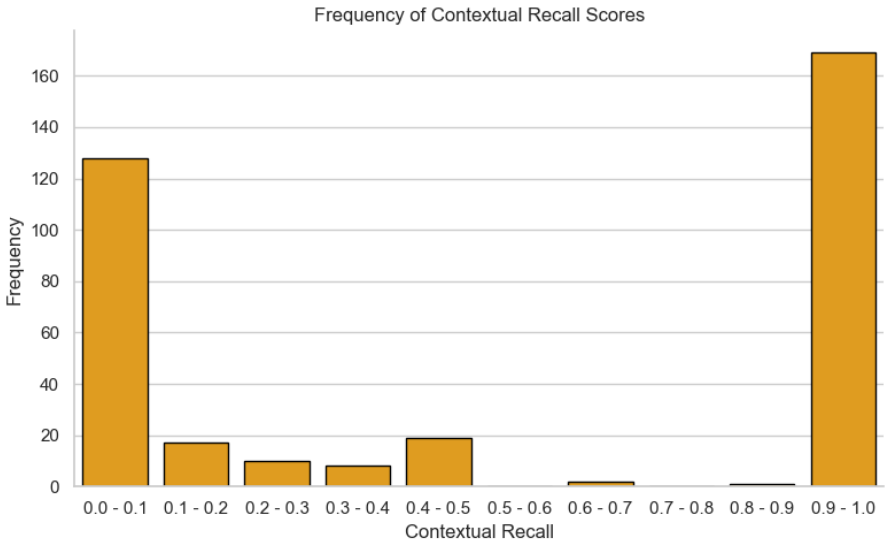

Figura 15: Distribuzione degli score di Contextual Recall

Anche in questo caso, osservando l'immagine 15, si evince che la metrica assume dei valori particolarmente instabili, evidenziando dei picchi in prossimità di 0 ed 1.
A differenza di prima, però, si registra anche un discreto numero di score intermedi. Analogamente a quanto è stato detto precedentemente, anche in questo caso la qualità degli embedding generati è probabilmente alla base dei risultati ottenuti. Infatti, è plausibile che utilizzando dei modelli di embedding studiati appositamente per un linguaggio tecnico-informatico, simile a quello del contesto applicativo di questo progetto, le performance del sistema RAG potrebbero migliorare discretamente.

6.1.4 Contextual Relevancy

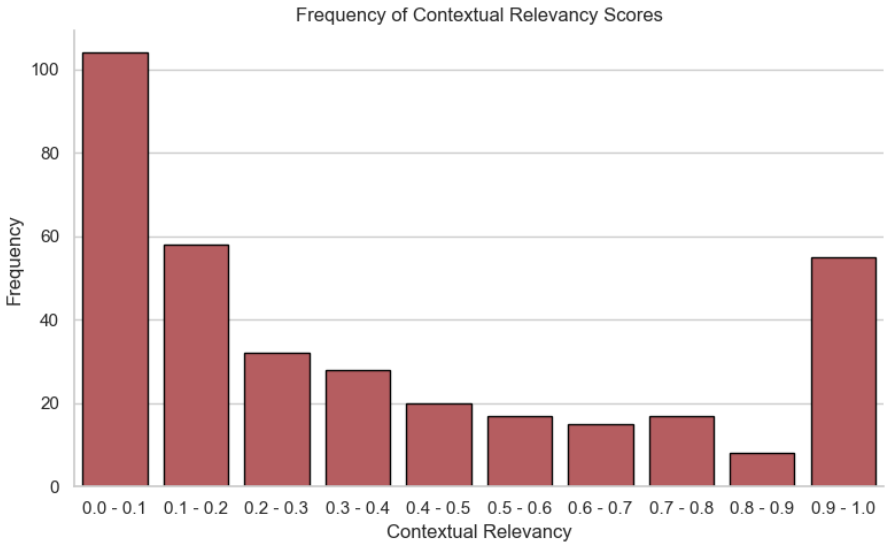

Figura 16: Distribuzione degli score di Contextual Relevancy

Infine, come è possibile notare dalla figura 16 e come anticipato in precedenza, la metrica di Contextual Relevancy ha ottenuto i risultati peggiori. La distribuzione dei risultati è abbastanza uniforme con una densità maggiore nei pressi dei valori inferiori a 0.4. Ancora una volta, è ragionevole pensare che, con un modello di generazione degli embedding più adeguato, sarebbe stato possibile ottenere risultati migliori.

7 Interfaccia grafica

7.1 Panoramica dell'interfaccia

Siccome il sistema RAG prevede l'utilizzo di molti modelli, che richiedono una elevata potenza di calcolo, è stata sviluppata una versione più leggera che fosse utilizzabile anche in locale. Questa versione comprende un'interfaccia grafica funzionale ed intuitiva che richiama l'aspetto dei classici chat-bot.
A differenza del sistema originale, il RAG sviluppato per l'interfaccia grafica utilizza il modello `granite-3.2-2b-instruct` al posto di quello descritto in 5.2; il modello usato appartiene alla medesima famiglia di LLM ma presenta un numero ridotto di parametri ed è stato utilizzato per ovviare al problema delle ridotte risorse di calcolo disponibili; inoltre, non viene effettuato il context pruning e non viene utilizzato l'adattatore LoRA per l'answerability delle domande.

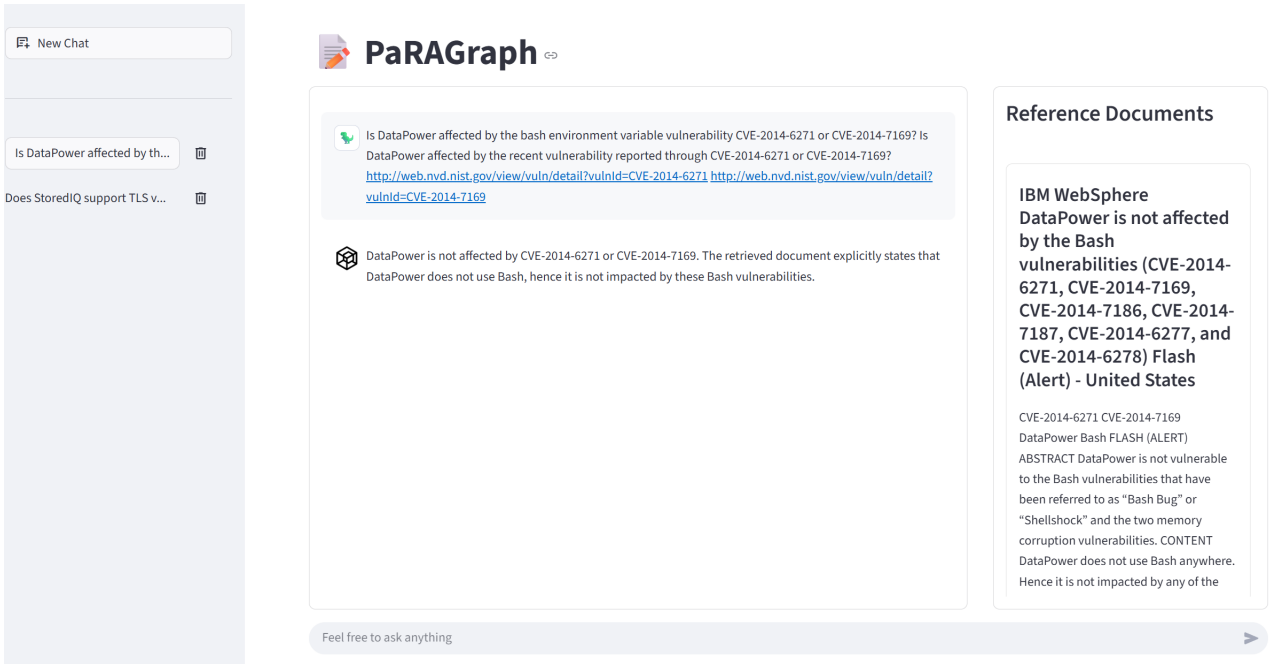

Figura 17: Interfaccia grafica di PaRAGraph

Come si può notare dall'immagine, la web-app sviluppata permette di selezionare una chat esistente, eliminarla o crearne una nuova tramite il menù laterale situato sulla sinistra dello schermo.
Una volta selezionata la chat, l'utente può porre una domanda al sistema, il quale procederà in maniera simile a quanto spiegato nei capitoli precedenti, riscrivendo la query per ottimizzarne la sintassi con l'obiettivo di recuperare i documenti più pertinenti ad essa.
Una volta acquisiti i documenti di interesse, essi vengono mostrati nella sezione a destra della chat; ogni documento presenta anche un link alla pagina originale del forum.
Infine, il sistema RAG genererà una risposta utilizzando le informazioni disponibili.

Per lo sviluppo dell'interfaccia grafica è stata utilizzata la libreria Streamlit, che permette di creare delle web-app semplici e intuitive in maniera rapida utilizzando Python.

7.2 Salvataggio delle conversazioni

Siccome il progetto non era destinato ad un ambiente di produzione, le conversazioni sono salvate in un cluster locale che utilizza `MongoDB` come database.
É stata scelta una struttura semplice per il mantenimento delle informazioni; infatti, ogni conversazione viene archiviata nel database utilizzando il seguente formato:

```
[
    {
        "_id": {
            "$oid": "687653909ef02eddd50eafca"
        },
        "last_modified": {
            "$date": "2025-07-15T15:11:44Z"
        },
        "chat_history": [
            {
                "role": "user",
                "content": "Why do I get an exception ..."
            },
            {
                "role": "assistant",
                "content": "The exception \"MQRC_NOT_AUTHORIZED\" ..."
            }
        ],
        "title": "Why do I get an exception ...",
        "retrieved_documents": [
            {
              "document_id": "swg21666383",
              "document_title": "IBM Authentication alias ...",
              "document_text": " TECHNOTE (TROUBLESHOOTING) ..."
            }
        ]
    }
    // ...
]
```

Listing 6: Struttura di una conversazione salvata nel database MongoDB

Questa struttura permette di mantenere l'ordine cronologico delle chat create dall'utente, consentendo di accedere direttamente alle conversazioni più recenti intraprese con il RAG.
Inoltre, salvando la conversazione con la stessa struttura utilizzata dai `chat_template` dei vari modelli, è semplice dividere i messaggi inviati dall'utente (contrassegnati con il ruolo `user`) dai messaggi inviati dal RAG (contrassegnati con il ruolo `assistant`).

8 Conclusioni

8.1 Osservazioni finali

Come riportato anche dall'articolo originale di IBM che descrive il dataset e i primi test effettuati su di esso, Tech-QA è un dataset impegnativo siccome sia le performance riportate nell'articolo, sia quelle ottenute in questo progetto non sono ottimali.

Questo è dovuto anche al fatto che il contenuto delle pagine web non utilizza un linguaggio quotidiano, ma un linguaggio tecnico tipico del settore informatico, ricco di codice, messaggi di errore e formule.

Questo genere di documenti potrebbe creare problemi ai modelli se non sono addestrati per lavorare su questa tipologia di dati, sia per quanto riguarda la creazione degli embedding sia per la generazione della risposta, ma potrebbe risultare problematico anche per le fasi di query rewriting e validazione. Infatti, modelli con un numero maggiore di parametri o fine-tunati potrebbero migliorare i risultati ottenuti considerevolmente.

8.2 Sviluppi futuri

Oltre alla scontata implementazione di modelli con un maggior numero di parametri, come possibili sviluppi futuri potrebbero essere prese in considerazione diverse opzioni.

Inizialmente, potrebbe essere una buona idea implementare un sistema di context pruning più efficace di quello testato in questo progetto; ciò dovrebbe garantire una maggiore precisione nella generazione della risposta e un risparmio in termini di tempo, avendo a disposizione meno dati ma più specifici.

Un'ulteriore modifica da prendere in considerazione consiste nell'implementazione di un algoritmo che, presi i documenti recuperati dal retriever, cerchi i link (o i metadati) presenti nell'articolo e li utilizzi per aggiungere al contesto altri documenti potenzialmente utili. Tuttavia, senza l'aggiunta di una componente che effettui context pruning, potrebbe diventare computazionalmente onerosa la fase di generazione della risposta, in quanto servirebbe un modello in grado di gestire un contesto più ampio.

Inoltre, si potrebbe aggiungere anche una fase di document reranking dopo il recupero dei documenti, con lo scopo di fornire al LLM designato per la generazione delle risposte un contesto che sia ordinato, completo e preciso.

Infine, potrebbe essere interessante valutare il sistema impiegando altre metriche di valutazione per ottenere una visione globale e più accurata delle sue prestazioni.

Bibliografia

[1] V. Castelli, R. Chakravarti, S. Dana et al., *The TechQA Dataset*, 2019. arXiv: 1911.02984 [cs.CL]. indirizzo: https://arxiv.org/abs/1911.02984 (cit. alle pp. 5, 18).