



Architetture Dati


Valutazione di Sistema RAG

AA 2024/2025

di

 Andrea Falbo

a.falbo7@campus.unimib.it

 Ruben Tenderini

r.tenderini@campus.unimib.it

Università degli Studi di Milano-Bicocca

Indice

1	Introduzione	1
1.1	Obiettivo	1
1.2	Repository	1
1.3	Capitoli	2
2	Fondamenti Teorici	3
2.1	Retrieval-Augmented Generation	3
2.1.1	Fase di Retrieval	3
2.1.2	Fase di Generazione	4
2.2	Metriche di Valutazione nei Sistemi Q&A	4
2.2.1	Benchmark Standard	5
2.2.2	Benchmark Personalizzati	5
3	Architettura del Sistema	6
3.1	Pipeline di Benchmarking	6
3.2	Pipeline Custom	6
3.2.1	Chunking & Overlap	7
3.2.2	Dataset	7
3.2.3	Librerie e Strumenti Utilizzati	9
3.2.4	Modelli Utilizzati	10
3.3	Pipeline BEIR	10
3.3.1	Dataset	11
3.3.2	Embedding	11
3.3.3	Retrieval	11
3.3.4	Valutazione	12
3.3.5	Librerie e Strumenti Utilizzati	12
4	Metodologia e Configurazioni Sperimentali	13
4.1	Obiettivo	13
4.2	Metriche	13
4.2.1	Metriche Custom	13
4.2.2	Metriche BEIR	14
4.3	Configurazioni	14
4.3.1	Configurazione Custom	14
4.3.2	Configurazione BEIR	16
4.4	Output della Pipeline	18
4.4.1	Output Custom	18
4.4.2	Output BEIR	19
4.5	Plotter	19
5	Analisi dei Risultati	20
5.1	Analisi Custom	20
5.1.1	Embedding Model	20

5.1.2	Chunk Size & Overlap	21
5.1.3	Top K	23
5.1.4	Temperature & Max Tokens	24
5.1.5	Hardware	24
5.2	Analisi BEIR	25
5.2.1	Dataset	25
5.2.2	Embedding Model	26
5.2.3	Score Function	27
5.2.4	Batch Size	28
5.2.5	Hardware	29
6	Conclusioni e Sviluppi Futuri	31
6.1	Conclusioni a livello di sviluppo del processo	31
6.2	Conclusioni a livello di analisi e valutazione	31
6.3	Limiti del sistema attuale	32
6.4	Sviluppi futuri	32
	Bibliografia	34

Capitolo 1

Introduzione

1.1 Obiettivo

Negli ultimi anni, i sistemi di domanda e risposta automatica (*Question Answering*, Q&A) basati su documenti hanno ricevuto crescente attenzione sia in ambito accademico che industriale. In particolare, l'approccio *Retrieval-Augmented Generation* (RAG) si è affermato come soluzione efficace per combinare le capacità di generazione del linguaggio naturale con il recupero mirato di informazioni da una base documentale.

L'obiettivo di questo progetto è sviluppare un sistema Q&A su documenti mediante una *pipeline RAG*, capace di rispondere a domande poste in linguaggio naturale sulla base del contenuto dei documenti forniti. Il progetto prevede anche la costruzione di un *benchmark* di valutazione per misurare le prestazioni del sistema, sia nella fase di recupero dei documenti rilevanti (*retrieval*), che nella fase di generazione delle risposte (*generation*).

In linea con recenti ricerche come [1, 10], il progetto si concentra anche sull'importanza della diversificazione dei dati di test e sulla rilevanza delle risposte in contesti applicativi specifici. Particolare attenzione è stata posta alla qualità dei dati utilizzati sia per la fase di recupero che nella valutazione e alla possibilità di generare in modo controllato scenari realistici e rappresentativi delle interazioni utente.

Lo scopo di questa relazione è descrivere nel dettaglio l'approccio sperimentale adottato, le tecnologie utilizzate, le decisioni progettuali prese, le analisi condotte, le difficoltà incontrate e i risultati ottenuti.

1.2 Repository

Per completezza e trasparenza, tutti i notebook, configurazioni e file di output sono disponibili pubblicamente nella seguente repository GitHub:

<https://github.com/Ruben-2828/rag-system-evaluation>

La repository contiene:

- Notebook del Benchmark Custom;
- Notebook del Benchmark BEIR;
- Plotter del Benchmark Custom;
- Plotter del Benchmark BEIR;
- Cartella contenente le configurazioni standard per entrambi i sistemi;

- Cartella contenente tutti i file di output delle diverse configurazioni e plot riassuntivo;
- file delle dipendenze python;
- il README del progetto in formato Markdown;
- la seguente relazione in formato PDF.

1.3 Capitoli

La relazione è organizzata come segue:

- **Capitolo 2** presenta i fondamenti teorici alla base dell'approccio RAG, descrivendo le tecniche di retrieval e generazione.
- **Capitolo 3** descrive l'architettura del sistema realizzato, ripercorrendo le fasi di sviluppo, le scelte effettuate e le iterazioni svolte per migliorare il sistema.
- **Capitolo 4** descrive in dettaglio le metodologie adottate per la valutazione del sistema RAG.
- **Capitolo 5** analizza i risultati ottenuti attraverso i due sistemi di benchmark costruiti, confrontando le diverse soluzioni sperimentate.
- **Capitolo 6** riassume le conclusioni del lavoro e propone possibili sviluppi futuri.

Capitolo 2

Fondamenti Teorici

2.1 Retrieval-Augmented Generation

Negli ultimi anni, l'approccio Retrieval-Augmented Generation (RAG) [6, 2] ha acquisito una crescente popolarità, in particolare nei contesti specializzati. Questo metodo unisce i punti di forza dei Large Language Models (LLM) con le tecniche moderne di recupero dell'informazione, arricchendo dinamicamente l'input per il modello generativo con contenuti pertinenti estratti da un corpus di supporto. Grazie a questa combinazione, è possibile ottenere risposte più accurate e contestualizzate, superando alcune delle principali limitazioni degli LLM, come la difficoltà nel trattare informazioni specifiche di dominio o soggette a frequenti aggiornamenti.

I sistemi RAG rappresentano una classe di modelli che uniscono due fasi principali:

- **Retrieval:** recupero di documenti rilevanti da una collezione in base a una query;
- **Generation:** generazione di una risposta in linguaggio naturale, condizionata sui documenti recuperati.

2.1.1 Fase di Retrieval

Il recupero dei documenti è basato su tecniche di *embedding* e confronto tra vettori semantici, come proposto nel lavoro [5]. I documenti testuali vengono suddivisi in *chunk* (frammenti), e ciascun chunk viene trasformato in un vettore denso tramite un modello di embedding.

In fase di interrogazione, anche la domanda viene convertita in vettore, e si effettua una *similarity search* per trovare i chunk più rilevanti. Questa operazione viene gestita da un *vector store*, ovvero una struttura indicizzata per la ricerca efficiente di vettori simili.

Similarity Search e Vector Store

Nel contesto RAG, la similarità tra query e documenti è calcolata tipicamente tramite *cosine similarity* o *dot product*. La ricerca può essere effettuata in modalità esatta o approssimata. Quest'ultima permette tempi di risposta ridotti, con una minima perdita in accuratezza, ed è dunque preferita in contesti ad alte prestazioni.

FAISS (Facebook AI Similarity Search) consente di gestire milioni di vettori con efficienza, supportando diverse strategie di indicizzazione e compressione [4].

Chunking & Overlap

Per rendere gestibili testi di grandi dimensioni all'interno di modelli con limite di token in input, i documenti vengono suddivisi in segmenti (*chunk*) di dimensione controllata.

A questa operazione si accompagna spesso un certo grado di sovrapposizione tra chunk consecutivi, per evitare che frammenti informativi vengano troncati in punti critici. Tali strategie di segmentazione risultano essenziali per massimizzare la qualità del recupero [8].

Embedding Models

Gli embedding sono rappresentazioni vettoriali dense che catturano il significato semantico di un testo.

La qualità degli embedding influenza direttamente l'efficacia della fase di retrieval: se la domanda e i documenti non vengono proiettati nello stesso spazio semantico, la ricerca fallisce anche in presenza di documenti pertinenti [5].

2.1.2 Fase di Generazione

Una volta selezionati i documenti più rilevanti, questi vengono inseriti in un prompt insieme alla domanda originale. Il prompt viene passato a un Large Language Model (LLM) che produce la risposta in linguaggio naturale, secondo il paradigma proposto da [7].

Prompt Design

Il design del prompt gioca un ruolo chiave nella qualità della risposta generata. Per evitare fenomeni di allucinazione, cioè l'invenzione di fatti non presenti nel contesto, è importante seguire buone pratiche di prompt engineering, come discusso in [3]:

- Includere istruzioni esplicite;
- Specificare che le risposte devono essere basate solo sul contesto fornito;
- Aggiungere esempi se il modello lo supporta.

Limiti del Contesto e Troncamento

I LLM presentano una finestra di contesto limitata, ovvero un numero massimo di token che possono essere considerati in input. È quindi importante controllare che i documenti inseriti nel prompt non superino tale soglia. Questo vincolo influenza anche il numero di chunk che possono essere forniti al modello in fase generativa. Studi recenti mostrano come il troncamento del contesto possa impattare negativamente sulla coerenza e completezza delle risposte [8].

2.2 Metriche di Valutazione nei Sistemi Q&A

Nel contesto di Retrieval-Augmented Generation, la valutazione può avvenire su due livelli:

- **Valutazione del retrieval:** capacità di recuperare testi realmente utili alla risposta;
- **Valutazione generativa:** qualità linguistica e aderenza semantica della risposta generata.

Tale valutazione può essere effettuata attraverso benchmark esistenti o tramite la costruzione di benchmark personalizzati, in funzione dello scenario applicativo.

2.2.1 Benchmark Standard

I benchmark standard sono raccolte predefinite di dataset ampiamente utilizzati per valutare in modo uniforme le prestazioni di modelli di retrieval e generazione. Tipicamente includono:

- Domande e risposte già annotate manualmente;
- Documenti o passaggi testuali associati;
- Metriche consolidate per il confronto.

Questi benchmark offrono un vantaggio importante in termini di replicabilità e confronto con lo stato dell'arte. Tuttavia, presentano anche alcune limitazioni:

- Coprono domini generici o specifici non sempre allineati al contesto di utilizzo reale;
- La distribuzione delle domande può essere non rappresentativa dell'interazione naturale degli utenti;
- Le risposte target sono spesso singole o sintetiche, il che può penalizzare modelli che offrono varianti semantiche corrette.

2.2.2 Benchmark Personalizzati

I benchmark personalizzati sono creati ad hoc per testare un sistema in un contesto specifico, utilizzando dati reali o generati artificialmente. Questi benchmark permettono di:

- Simulare il comportamento e le esigenze degli utenti target;
- Controllare il grado di difficoltà e la varietà delle domande;
- Allineare il contenuto documentale al dominio applicativo;
- Introdurre valutazioni qualitative più aderenti al caso d'uso.

La costruzione di benchmark personalizzati richiede tempo e risorse, ma risulta fondamentale quando l'accuratezza del sistema è critica o quando i dati pubblici non riflettono la specificità del dominio trattato.

Capitolo 3

Architettura del Sistema

Nel presente capitolo, così come nei successivi, vengono riportati frammenti di codice estratti dai notebook sviluppati durante il progetto. Per una visione completa e ulteriori dettagli, si rimanda alla repository GitHub disponibile alla Sezione 1.2.

3.1 Pipeline di Benchmarking

Per la valutazione delle performance del sistema di retrieval sono state implementate due pipeline distinte:

1. La prima è una **pipeline custom**, sviluppata internamente, che consente di testare il sistema con dataset specifici e metriche personalizzate, offrendo maggiore flessibilità nel controllo dei parametri e nella gestione dei dati.
2. La seconda **pipeline** sfrutta il benchmark **BEIR** (Benchmarking Information Retrieval) [9], un framework standardizzato che fornisce un insieme eterogeneo di dataset di query e documenti, insieme a metriche comuni per valutare la capacità di un sistema di recuperare informazioni rilevanti.

È importante sottolineare che le due pipeline sono state implementate con scopi differenti e non si intende condurre un confronto diretto tra di esse, ma piuttosto un'analisi interna a ciascuna. Questo perché le pipeline operano su paradigmi di retrieval distinti: la pipeline custom utilizza FAISS per eseguire una *similarity search* basata su embedding vettoriali, mentre BEIR si basa su tecniche di *exact search* e retrieval tradizionale.

3.2 Pipeline Custom

La pipeline custom è strutturata in quattro fasi principali:

1. **Preprocessing**: ciascun documento è preprocessato estraendo il testo dai token, rimuovendo gli elementi HTML e frammentandolo in segmenti di lunghezza controllata (chunk).
2. **Indicizzazione**: i chunk vengono convertiti in vettori tramite un modello di embedding e memorizzati in un indice vettoriale FAISS.
3. **Retrieval**: data una domanda, si esegue una ricerca di similarità per recuperare i chunk più rilevanti (top- k).
4. **Generazione**: opzionalmente, il contesto recuperato viene fornito a un LLM per generare una risposta alla domanda.

Nel caso in cui la generazione sia disattivata, il sistema restituisce direttamente il primo documento recuperato, simulando un comportamento ‘extractive’. La modalità generativa è abilitata tramite un flag di configurazione.

```
1  if use_llm:
2      prediction = ask(query, context)
3  else:
4      prediction = context[0]
```

3.2.1 Chunking & Overlap

Nel contesto specifico, le tecniche di chunking e overlap sono state realizzate tramite `RecursiveCharacterTextSplitter` di `LangChain`, con parametri configurabili:

- **Chunk size:** numero massimo di caratteri in un singolo blocco.
- **Chunk overlap:** numero di caratteri condivisi tra chunk consecutivi.

```
1 splitter = RecursiveCharacterTextSplitter(
2     chunk_size=config["chunk_size"],
3     chunk_overlap=config["chunk_overlap"]
4 )
```

3.2.2 Dataset

Per il modello base della pipeline custom è stato utilizzato un sottoinsieme del dataset `Natural Questions (NQ)`, una raccolta di domande reali poste dagli utenti di Google e annotate con risposte brevi e lunghe tratte da Wikipedia.

Parametri Utilizzati

Per la prima versione stabile della pipeline custom, sono stati utilizzati i seguenti parametri:

- **Split utilizzato:** `train[100:1000]`
- **Numero di esempi validi valutati:** 10

Il parametro *numero di esempi validi valutati* è personalizzabile e si rende necessario poiché alcuni elementi presenti nel dataset non contenevano un contesto associato, risultando quindi inutilizzabili ai fini della nostra valutazione. Per questo motivo, durante la selezione dei dati, si procede a scorrere gli elementi nel dataset scartando quelli privi di contesto e mantenendo solamente quelli rilevanti fino al raggiungimento del numero desiderato di esempi validi.

```
1 num_valid_examples = config["num_valid_examples"]
2 use_llm = config["use_llm"]
3
4 results = []
5 i = 0
6
7 ...
```

```

8 while len(results) < num_valid_examples and i < len(dataset):
9     ...

```

Architettura dei Dati

Ogni esempio presente nello split di training del dataset *Natural Questions (NQ)* è rappresentato da una quintupla nella forma:

$\langle \text{id}, \text{document}, \text{question}, \text{long_answer_candidates}, \text{annotations} \rangle$

dove:

- **id**: stringa che identifica univocamente l'esempio.
- **document**: dizionario contenente la rappresentazione HTML completa della pagina web da cui è stata estratta la risposta.
- **question**: stringa contenente la domanda naturale (in linguaggio naturale) posta dall'utente.
- **long_answer_candidates**: lista di candidati long answer, ciascuno definito da un intervallo di token che individua una regione specifica del documento (tipicamente una sezione, paragrafo o div).
- **annotations**: lista di annotazioni manuali, ciascuna comprendente una risposta lunga (**long_answer**) e opzionalmente una o più risposte brevi (**short_answers**), oltre a un attributo **yes_no_answer** nei casi pertinenti.

Per maggiori dettagli sulla struttura del dataset e sul processo di annotazione, si consiglia di consultare la documentazione su Hugging Face disponibile al seguente link:

https://huggingface.co/datasets/google-research-datasets/natural_questions

Suddivisione del Dataset

Il subset di NQ utilizzato, cioè **default**, è suddiviso in due split principali:

- **train split**: composto da 10.600 esempi.
- **validation split**: composto da 7.830 esempi.

Preprocessing del Documento

Il dataset fornisce l'intero contenuto del documento target in forma tokenizzata. Tuttavia, molti di questi token rappresentano markup HTML o elementi strutturali. La funzione seguente rimuove tali elementi e restituisce il testo continuo del documento, utile come contesto per il retrieval e la generazione.

```

1 def preprocess_text(sample):
2     tokens = sample["document"]["tokens"]
3     return " ".join([t for t, html in zip(tokens["token"],
        tokens["is_html"]) if not html])

```

Estrazione delle Risposte Annotate

Per la pipeline custom è stata definita la funzione `extract_answers` che consente di estrarre le risposte brevi e lunghe da ciascun esempio del dataset. Il dataset **Natural Questions** organizza le risposte in base agli indici di inizio e fine token. Alcune annotazioni possono essere mancanti o contenere elementi HTML non desiderati nel testo finale. Per questo motivo, la funzione filtra i token HTML e restituisce solo testo pulito. Infine, il metodo restituisce una coppia di stringhe (long, short), oppure una stringa vuota in caso di assenza di annotazioni.

```
1 def extract_answers(sample):
2     tokens = sample["document"]["tokens"]
3     short_answer = ""
4     start = sample["annotations"]["short_answers"][0]["start_token"]
5     end = sample["annotations"]["short_answers"][0]["end_token"]
6     if len(start) > 0:
7         short_answer = " ".join([
8             t for t, html in
9                 zip(tokens["token"][int(start[0]):int(end[0])],
10                    tokens["is_html"][start[0]:end[0]])
11             if not html
12         ])
13     long_answer = ""
14     if sample["annotations"]["long_answer"][0]["start_token"] != -1:
15         start =
16             sample["annotations"]["long_answer"][0]["start_token"]
17         end = sample["annotations"]["long_answer"][0]["end_token"]
18         long_answer = " ".join([
19             t for t, html in zip(tokens["token"][start:end],
20                                tokens["is_html"][start:end])
21             if not html
22         ])
23     return long_answer or "", short_answer or ""
```

3.2.3 Librerie e Strumenti Utilizzati

Le principali librerie Python utilizzate sono:

- **LangChain**: orchestrazione della pipeline, interfacce per modelli e vectorstore.
- **HuggingFace Transformers e Hub**: accesso a modelli preaddestrati e gestione delle chiamate API.
- **SentenceTransformers**: embedding dei documenti.
- **FAISS**: indicizzazione e ricerca approssimata nel vectorstore.
- **Evaluate**: calcolo delle metriche.
- **Datasets (HuggingFace)**: caricamento e gestione del dataset Natural Questions.

3.2.4 Modelli Utilizzati

Per il modello base sono stati utilizzati i seguenti modelli:

- **Modello di embedding:** `sentence-transformers/all-MiniLM-L6-v2`, compatto e adatto a operazioni di embedding veloce.
- **Indice vettoriale:** FAISS in modalità Flat index L2.
- **Modello generativo:** `microsoft/Phi-3.5-mini-instruct`, accessibile tramite endpoint Hugging Face, configurato con `temperature=0.5` e massimo di 256 token generati.

L'indice vettoriale è stato mantenuto fisso in modalità `IndexFlatL2` per tutte gli esperimenti eseguiti, in modo da garantire coerenza nella fase di retrieval e rendere comparabili le diverse configurazioni testate. Grazie alla normalizzazione L2 dei vettori durante la fase di indicizzazione, la distanza euclidea calcolata dall'indice equivale a una ricerca basata su **similarità coseno**.

Diversamente, il modello di embedding è stato variato nel corso della sperimentazione per valutare l'impatto sulla qualità dei risultati recuperati, con particolare attenzione alla compatibilità con il dataset e alla velocità di inferenza. Maggiori dettagli sono disponibili nel Capitolo 4.

Per quanto riguarda il modello generativo, sono stati condotti alcuni test preliminari con alternative come `Mistral-7B-Instruct` e `HuggingFaceH4/zephyr-7b-beta`, tuttavia questi modelli si sono rivelati troppo onerosi sia in termini di tempo di risposta sia di consumo di crediti API. Per questo motivo, ai fini dell'efficienza e della scalabilità, si è deciso di utilizzare un modello più leggero come `Phi-3.5-mini-instruct`, che ha garantito un buon compromesso tra qualità delle risposte e costi computazionali.

3.3 Pipeline BEIR

La pipeline BEIR è stata adattata per lavorare con modelli `Sentence-BERT`, utilizzando un retrieval denso e valutando le performance con metriche comuni nel campo del document retrieval.

`Sentence-BERT` è una variante di BERT (un modello basato su trasformatori che apprende rappresentazioni contestuali del linguaggio) progettata per produrre embedding vettoriali di frasi, rendendo possibile confrontare semanticamente query e documenti in modo efficiente.

La pipeline BEIR è articolata in due fasi distinte:

1. **Embedding:** viene inizializzato un modello `Sentence-BERT` con i pesi specificati nella configurazione.
2. **Retrieval:** utilizzo di `DenseRetrievalExactSearch` con un retriever configurato tramite `EvaluateRetrieval` per effettuare la ricerca densa.

Oltre alle due fasi principali appena elencate, il benchmark BEIR svolge le seguenti operazioni:

1. **Caricamento del Dataset:** prima dell'embedding, attraverso l'utilizzo di `GenericDataLoader` viene caricato corpus, query e qrels dal dataset selezionato.
2. **Valutazione:** dopo la fase di retrieval, avviene il confronto tra i documenti recuperati e le risposte attese.
3. **Salvataggio dei Risultati:** infine, avviene il salvataggio dei risultati delle metriche in formato CSV e della configurazione utilizzata in formato JSON.

Le seguenti fasi sono analizzate nel dettaglio nelle prossime sezioni.

3.3.1 Dataset

Nel contesto del benchmark BEIR, ogni dataset è strutturato attorno a tre componenti fondamentali: **corpus**, **queries** e **qrels**. Il *corpus* è l'insieme dei documenti disponibili, da cui il sistema deve recuperare quelli più pertinenti. Ogni documento può essere un paragrafo, una frase o un breve testo, a seconda del dataset. Le *queries* rappresentano le domande poste. I *qrels* (*query relevance judgments*) indicano, per ciascuna query, quali documenti del corpus sono considerati rilevanti. Le tre componenti sono state caricate utilizzando il seguente comando:

```
1 corpus, queries, qrels =  
    GenericDataLoader(data_folder=os.path.join(data_path,  
        dataset)).load(split="test")
```

3.3.2 Embedding

In questa fase, i documenti del corpus e le query vengono convertiti in rappresentazioni vettoriali tramite un modello preaddestrato.

Per la configurazione di base, è stato utilizzato il modello `sentence-transformers/all-MiniLM-L6-v2`, lo stesso utilizzato nella pipeline custom. Il modello viene integrato nel sistema attraverso l'oggetto `DenseRetrievalExactSearch`, configurato con funzione di similarità `dot` e dimensione del batch pari a 32.

```
1 retriever = EvaluateRetrieval(dres(sbert,  
    batch_size=config['batch_size'],  
    score_function=config['score_function']))
```

Tali impostazioni rappresentano la configurazione di riferimento, soggetta a variazione negli esperimenti descritti nel Capitolo 4.

3.3.3 Retrieval

Una volta calcolati gli embedding, il sistema procede con il recupero dei documenti più pertinenti per ciascuna query, confrontando i vettori nel corpus con quelli delle query. Questo avviene mediante una semplice operazione di similarità, secondo quanto specificato nella configurazione del modello.

```
1 retrieved = retriever.retrieve(corpus, queries)
```

Il risultato è una mappatura tra ciascuna query e un insieme ordinato di documenti, classificati in base alla loro rilevanza stimata.

3.3.4 Valutazione

Per valutare la qualità dei risultati ottenuti, vengono confrontati i documenti restituiti dal sistema con quelli effettivamente rilevanti secondo i `qrels`. La valutazione avviene su diversi valori di `k`, corrispondenti al numero di documenti restituiti.

```
1 k_values = config['k_values']
2 results = retriever.evaluate(qrels, retrieved, k_values=k_values)
3 output_folder = config["output_folder"]
4 run_name = f"run_beir_{datetime.now().strftime("%Y%m%d_%H%M%S")}"
5 out_path = os.path.join(output_folder, run_name)
6 os.makedirs(out_path, exist_ok=True)
7
8 results_df = pd.DataFrame()
9 results_df["k"] = k_values
10
11 for r in results:
12     metric = next(iter(r.keys()))
13     metric = metric.split("@")[0]
14     values = r.values()
15     results_df[metric] = values
```

3.3.5 Librerie e Strumenti Utilizzati

Oltre le classiche librerie Python come Pandas e JSON, sono state utilizzate le seguenti librerie:

- **BEIR**: framework principale per la gestione dataset, retrieval e valutazione.
- **SentenceTransformers**: modelli preaddestrati per embedding semantico.

Capitolo 4

Metodologia e Configurazioni Sperimentali

4.1 Obiettivo

In questo capitolo vengono descritte in dettaglio le metodologie adottate per la valutazione del sistema RAG, con riferimento sia alla componente di **retrieval** che a quella di **generazione**.

Viene definita una configurazione di base per ciascuna pipeline (custom e BEIR), che costituisce il punto di partenza per una serie di esperimenti controllati. A partire da questa base, sono stati variati singolarmente i principali parametri del sistema, con l'obiettivo di isolare e comprendere l'impatto di ciascuno di essi sulle metriche di prestazione.

Le configurazioni sperimentali descritte riguardano entrambe le pipeline sviluppate. Tuttavia, poiché la pipeline BEIR si limita alla fase di retrieval, le valutazioni relative alla generazione sono state effettuate esclusivamente sulla pipeline custom.

4.2 Metriche

4.2.1 Metriche Custom

Le prestazioni della pipeline custom sono state valutate utilizzando le seguenti metriche, scelte per analizzare sia la qualità del contenuto generato che l'efficienza computazionale.

- **ROUGE-L**: misura l'overlap in termini di longest common subsequence tra la risposta generata e la risposta di riferimento.
- **BLEU**: valuta la precisione n-gram della predizione rispetto al riferimento, ovvero quanto le sequenze di n parole nella frase generata coincidano con quelle nella frase corretta. Include anche una penalità per risposte troppo brevi, per evitare che frasi corte ma parzialmente corrette ottengano punteggi elevati.
- **Longest Match Ratio**: metrica personalizzata che calcola il rapporto tra la lunghezza della sottostringa continua più lunga condivisa tra contesto e predizione, e la lunghezza totale del contesto. La differenza principale rispetto a ROUGE-L è che valuta la sottostringa continua, mentre quest'ultimo valuta sottosequenze non necessariamente adiacenti.
- **Tempo di Esecuzione**: per ciascun esempio valido, è stata utilizzata la funzione `time.perf_counter()` per misurare i tempi di esecuzione e permettere un confronto al variare delle configurazioni.

Ogni metrica è stata calcolata in due scenari distinti:

1. **Baseline Retrieval**: senza LLM, usando direttamente il passaggio più rilevante recuperato.
2. **Full RAG**: con generazione della risposta da parte del modello LLM a partire dal contesto.

4.2.2 Metriche BEIR

Le metriche di valutazione utilizzate dalla pipeline BEIR sono le seguenti:

- **NDCG** (Normalized Discounted Cumulative Gain): valuta la qualità del ranking prodotto, penalizzando risultati rilevanti recuperati in posizioni basse.
- **MAP** (Mean Average Precision): media della precisione per ogni query, calcolata fino alla posizione k.
- **Recall**: frazione di documenti rilevanti recuperati entro i primi k.
- **Precision**: percentuale di documenti rilevanti tra i primi k risultati.

4.3 Configurazioni

4.3.1 Configurazione Custom

Per facilitare e standardizzare l'esecuzione degli esperimenti, è stato realizzato un file JSON contenente tutti i parametri fondamentali della pipeline custom.

```
1 {  
2   "dataset_split": "train[100:1000]",  
3   "output_folder": "output/",  
4   "embedding_model_name": "sentence-transformers/all-MiniLM-L6-v2",  
5   "chunk_size": 500,  
6   "chunk_overlap": 100,  
7   "top_k": 5,  
8   "num_valid_examples": 10,  
9   "use_llm": true,  
10  "llm_model": "microsoft/Phi-3.5-mini-instruct",  
11  "temperature": 0.5,  
12  "max_new_tokens": 256  
13 }
```

Listing 4.1: Configurazione Pipeline Custom

La seguente tabella elenca le variabili che sono state testate e analizzate per osservare l'effetto sulle metriche sopra descritte:

Parametro	Valutazione
Embedding model	Variazione del modello di embedding
Chunk size	Dimensione dei blocchi di testo usati per indicizzazione
Chunk overlap	Sovrapposizione tra chunk
Top-k	Numero di documenti recuperati dalla vector store
Temperature	Valore di temperatura per la generazione
Max tokens	Limite massimo di token generabili dal modello
Hardware	Hardware su cui è stato eseguito il test

Tabella 4.1: Parametri variati nei benchmark con la pipeline custom

Di seguito, viene fornita una motivazione e i valori utilizzati per l'analisi di ciascun parametro.

Embedding Model

L'embedding model è stato analizzato per valutare l'impatto della qualità, dimensione e architettura del modello sia sulla fase di **retrieval** che di **generazione**. I modelli più grandi possono fornire rappresentazioni semantiche più ricche, ma a costo di maggiore tempo computazionale.

I valori testati sono:

- sentence-transformers/paraphrase-MiniLM-L6-v2
- sentence-transformers/all-MiniLM-L6-v2
- sentence-transformers/all-MiniLM-L12-v2
- BAAI/bge-small-en-v1.5
- BAAI/bge-base-en-v1.5
- BAAI/bge-large-en-v1.5

Chunk Size & Overlap

La dimensione del chunk e il grado di sovrapposizione influenzano direttamente la granularità del contesto fornito al sistema. Chunk troppo piccoli rischiano di frammentare il significato, mentre chunk troppo grandi possono contenere rumore irrilevante. In ogni esperimento, è stato mantenuto costante il rapporto tra la dimensione del chunk e l'overlap, per garantire una comparabilità coerente tra le configurazioni.

Valori testati:

- **Chunk Size:** 100, 200, 500, 1000, 2500
- **Chunk Overlap:** 20, 40, 100, 200, 500

Questa analisi è stata condotta sia per la fase di **retrieval** che per la **generazione**.

Top-k

Il parametro **top-k** definisce quanti documenti vengono recuperati dal vector store. Questo valore influisce sulla qualità del contesto fornito al generatore ed è stato quindi testato esclusivamente nella fase di **generazione**.

Valori testati:

- **top_k**: 1, 3, 5, 10

Temperature & Max Tokens

Questi parametri influenzano il comportamento del modello LLM durante la generazione. La temperatura regola il grado di casualità della risposta, mentre **max_tokens** limita la lunghezza massima della predizione. Sono stati combinati per osservare le interazioni tra coerenza, lunghezza e qualità. Come per il test precedente, essendo entrambi parametri della configurazione del modello LLM, sono stati testati esclusivamente nella fase di **generazione**.

Valori testati:

- **temperature**: 0.0, 0.3, 0.5, 0.7
- **max_new_tokens**: 128, 256, 512

Hardware

L'analisi delle prestazioni è stata condotta su due distinte configurazioni hardware, al fine di verificare eventuali variazioni nei valori delle metriche e valutare l'impatto corrispondente sulla velocità di esecuzione.

- **Config A**: Intel i7-10700F, 32GB RAM, Windows 11
- **Config B**: AMD Ryzen 5 5600H, 16GB RAM, Windows 11

4.3.2 Configurazione BEIR

Anche per la pipeline BEIR, è stato creato un file di configurazione JSON:

```
1 {  
2     "dataset": "scifact",  
3     "datasets_folder": "datasets/",  
4     "model_name": "sentence-transformers/all-MiniLM-L6-v2",  
5     "batch_size": 32,  
6     "score_function": "cos_sim",  
7     "k_values": [1, 3, 5, 10],  
8     "output_folder": "output/beir/scifact/L6-v2"  
9 }
```

Listing 4.2: Configurazione Pipeline BEIR

Anche in questo caso, sono stati condotti esperimenti variando singolarmente i principali parametri. La seguente tabella mostra quelli presi in considerazione:

Parametro	Descrizione
Dataset	Dataset BEIR usato per la valutazione
Embedding model	Modello Sentence-BERT impiegato per il retrieval
Score function	Funzione di similarità
Batch size	Dimensione del batch durante l'inferenza
Hardware	Hardware su cui è stato eseguito il test

Tabella 4.2: Parametri variati nei benchmark (pipeline BEIR)

Di seguito, viene fornita una motivazione e i valori utilizzati per l'analisi di ciascun parametro.

Dataset

È stato variato per verificare la robustezza del sistema su task eterogenei. I dataset selezionati rappresentano domini e dimensioni differenti:

- **SciFact:** task di fact-checking su contenuti scientifici. Rappresenta il dataset più piccolo, con solo 300 queries e 5k corpus.
- **SCIDOCS:** dataset di documenti scientifici focalizzati su articoli e paper relativi a varie discipline STEM. Rappresenta il dataset di dimensioni intermedie, con 1000 queries e 25k corpus.
- **FiQA-2018:** domande in ambito finanziario. Rappresenta il dataset più grande, con 648 queries e 57k corpus.

Embedding Model

Sono stati testati gli stessi modelli di embedding della pipeline custom:

- BAAI/bge-small-en-v1.5
- BAAI/bge-base-en-v1.5
- BAAI/bge-large-en-v1.5
- sentence-transformers/all-MiniLM-L6-v2
- sentence-transformers/all-MiniLM-L12-v2
- sentence-transformers/paraphrase-MiniLM-L6-v2

Score Function

Sono state confrontate due funzioni di similarità per valutare il modo in cui la distanza tra query e documenti influenza la fase di retrieval:

- **cos_sim:** similarità coseno;
- **dot:** prodotto scalare.

Batch Size

La dimensione del batch è stata variata per misurare eventuali impatti in termini di prestazioni e tempi di inferenza. I valori testati sono:

- 16
- 32
- 64

Top-k

I valori di k per il calcolo delle metriche di retrieval sono stati fissati per tutti gli esperimenti a:

$$k = [1, 3, 5, 10]$$

Questo consente di osservare la performance del sistema nel recupero dei primi documenti più rilevanti in funzione della loro posizione.

4.4 Output della Pipeline

4.4.1 Output Custom

Al termine dell'esecuzione del notebook, la pipeline custom produce tre file principali, ciascuno con uno scopo preciso legato all'analisi e alla tracciabilità degli esperimenti:

1. **results.csv**: file tabellare contenente i valori delle metriche calcolate per ciascun esempio valutato. Le colonne presenti sono:
 - **index**: indice dello specifico esempio nello split del dataset.
 - **rougeL**: valore ROUGE-L tra la predizione e il contesto corretto.
 - **rougeL_golden**: ROUGE-L calcolato tra la predizione ottenuta usando il contesto "golden" e il golden stesso.
 - **bleu**: valore BLEU tra la predizione e il contesto corretto.
 - **bleu_golden**: BLEU ottenuto usando il contesto "golden".
 - **longest_match**: rapporto tra la sottostringa più lunga in comune tra la predizione e il contesto corretto, rispetto alla lunghezza del contesto.
 - **longest_match_golden**: valore della metrica precedente usando il contesto golden.
2. **used_config.json**: file in formato JSON contenente l'intera configurazione con cui è stato eseguito l'esperimento. Serve a garantire la riproducibilità degli esperimenti e a documentare i parametri di ciascuna run.
3. **responses.json**: file che raccoglie i dettagli completi per ogni esempio processato. Per ogni entry, viene salvato una lista con i seguenti campi:

- `index`: indice dello specifico esempio.
- `query`: domanda da processare.
- `gold_answer`: risposta corretta estratta dal dataset.
- `prediction`: risposta generata (o recuperata) sulla base del contesto prodotto dalla pipeline.
- `prediction_golden`: risposta ottenuta dal modello usando direttamente il contesto di riferimento ideale.
- `elapsed_time`: tempo totale di esecuzione di ciascun esempio processato.

4.4.2 Output BEIR

A differenza della pipeline custom, l'output di BEIR consiste di soli due file, in quanto non si occupa della fase di generazione e dunque non vi è il file `reponses.json`:

1. `results.csv`: contiene le metriche BEIR mostrate nella Sezione 4.2.2.
2. `used_config.json`: con lo stesso scopo del file di configurazione di output della pipeline custom.

4.5 Plotter

Per entrambe le pipeline è stato realizzato uno script di visualizzazione dei risultati, che permette di generare grafici comparativi in automatico a partire dai file `results.csv`.

Per quanto riguarda la pipeline custom, il confronto tra i modelli testati è stato rappresentato mediante *bar plot* delle metriche. Per la pipeline BEIR, che prevede la valutazione delle metriche al variare del parametro k , è stato impiegato un *line chart*.

Capitolo 5

Analisi dei Risultati

5.1 Analisi Custom

Per valutare le prestazioni della pipeline custom su diverse configurazioni di modelli, sono state analizzate e tracciate le principali metriche: BLEU, ROUGE-L e Longest Match. Queste metriche sono calcolate al variare del modello utilizzato, al fine di confrontare l'efficacia delle risposte generate.

Nella fase di *generation*, oltre al calcolo delle metriche standard, ottenute usando il contesto recuperato automaticamente dal sistema, vengono riportate anche le rispettive metriche golden, ovvero i punteggi ottenuti facendo generare il modello a partire dal *golden context*. Nei plot risultanti, le metriche golden appaiono **fisse** e **costanti**: questo accade perché esse non dipendono dal modello di retrieval o embedding. Tali metriche rappresentano quindi un *upper bound teorico* delle performance del sistema.

5.1.1 Embedding Model

Retrieve

La fase di retrieve evidenzia differenze nette tra i modelli di embedding utilizzati. Le metriche si comportano in modo coerente, ma con livelli di performance significativamente variabili.

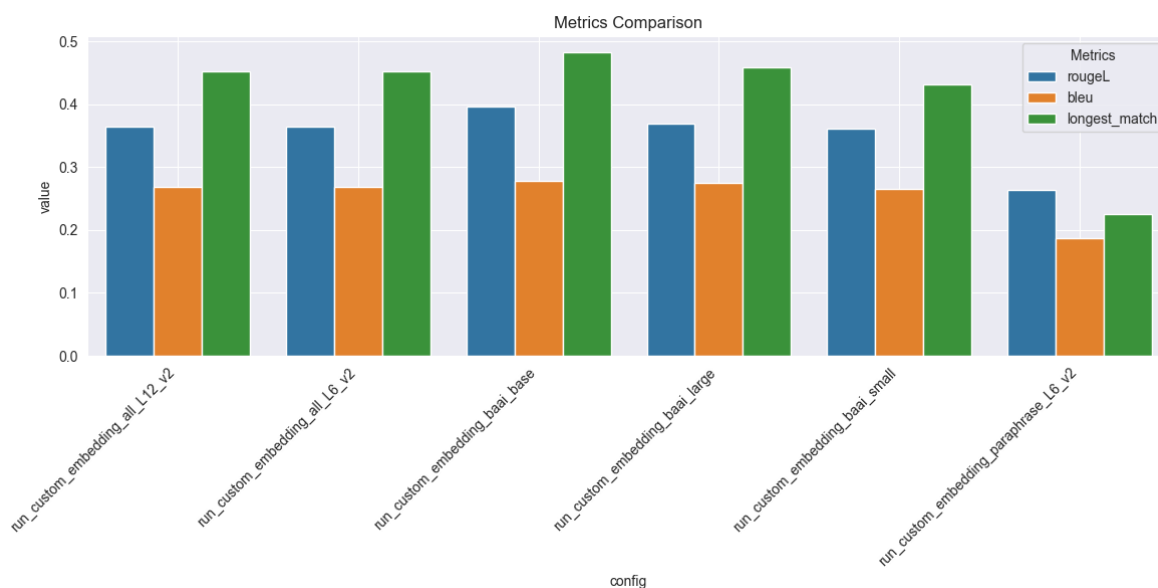


Figura 5.1: Test dei modelli di embedding in fase di retrieve per la pipeline custom

Il modello `baai_base` ottiene i risultati migliori su tutte le metriche principali. Risulta superiore sia a `_baai_small` che a `baai_large`, configurandosi come il miglior compromesso tra qualità e dimensione. Al contrario, `paraphrase_t6_v2` presenta le performance peggiori, verosimilmente a causa della sua estrema leggerezza. Infine, `all_t12_v2` non mostra miglioramenti rilevanti rispetto alla versione base `all_t6_v2`, suggerendo che l'aumento di capacità non si traduce necessariamente in maggiore efficacia nella fase di retrieval.

Generation

In fase di generazione, le metriche `rougeL` e `rougeL_golden` risultano nettamente superiori alle altre, suggerendo una maggiore sensibilità di queste metriche alla qualità del contenuto prodotto.

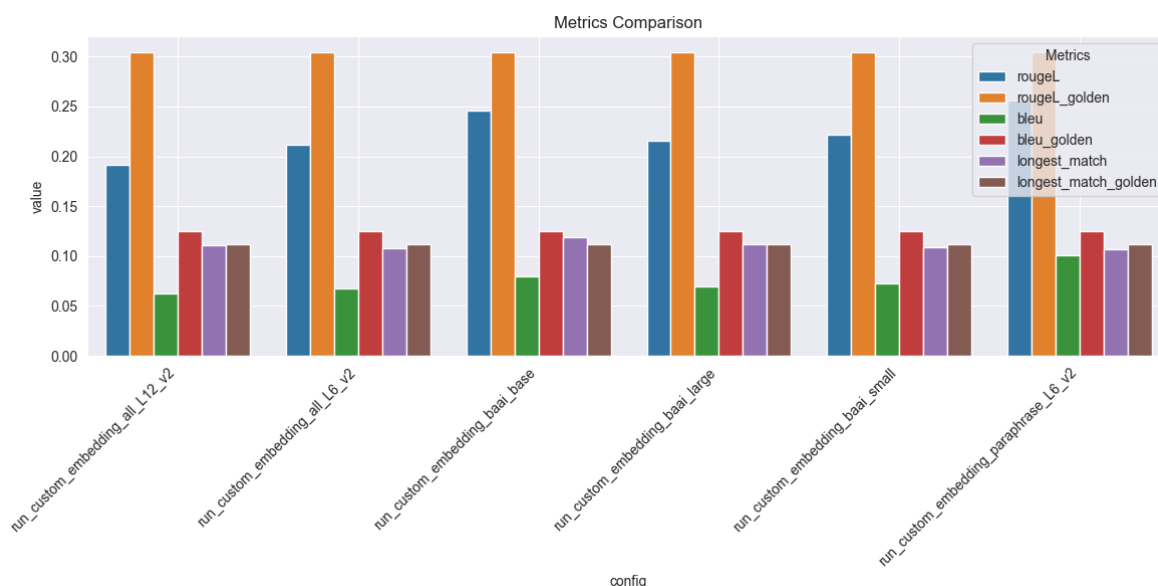


Figura 5.2: Test dei modelli di embedding in fase di generazione per la pipeline custom

È interessante osservare che il modello `paraphrase_t6_v2`, nonostante abbia ottenuto i risultati peggiori in fase di retrieve, produce le performance migliori in fase di generazione. Seguono `baai_base` e gli altri modelli, con `all_t12_v2` che si colloca in fondo alla classifica. Questo comportamento suggerisce che un embedding meno accurato nel recuperare il contesto possa comunque portare a generazioni di qualità, probabilmente per effetto della maggiore sinteticità del contesto selezionato.

5.1.2 Chunk Size & Overlap

Retrieve

La dimensione dei chunk ha un impatto evidente sulla fase di recupero.

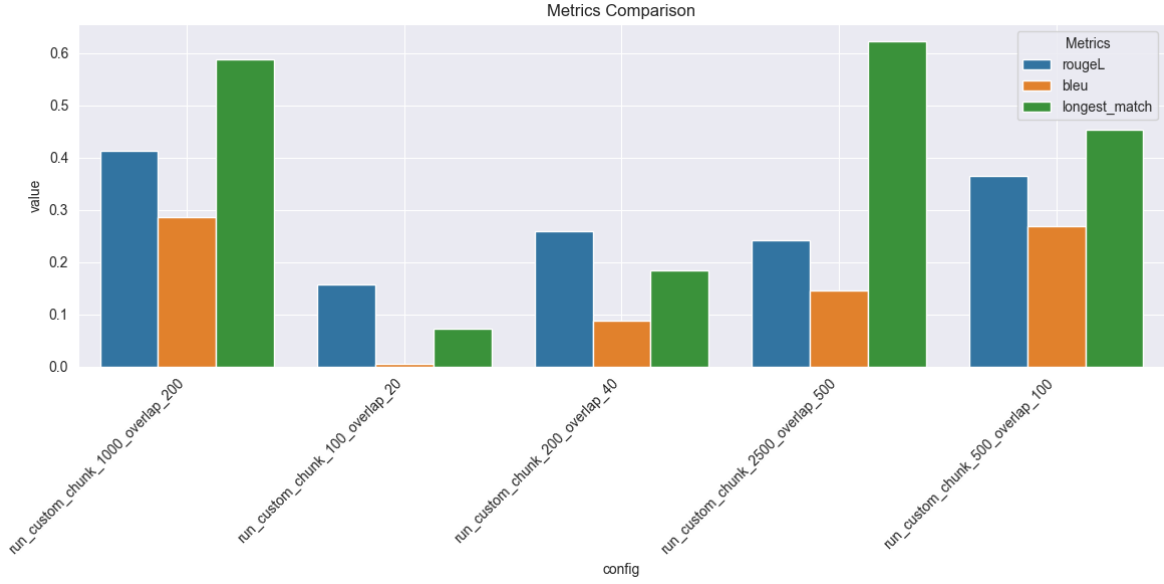


Figura 5.3: Test su dimensione dei chunk e overlap in fase di retrieve

Chunk più ampi (es. 500 o 1000 token) portano a performance significativamente migliori, mentre dimensioni molto ridotte penalizzano il retrieve in modo drastico. In particolare, la configurazione con 100 token e overlap 20 produce metriche molto basse, con valori vicini allo zero per **bleu**. Chunk molto lunghi (es. 1500) mostrano un miglioramento in **longest_match**, dato che la lunghezza aumenta le probabilità di una sovrapposizione con la risposta attesa. Tuttavia, tali configurazioni tendono a includere informazioni superflue, riducendo le performance in **bleu** e **rougeL**. Le configurazioni di dimensioni intermedie rappresentano quindi un buon compromesso tra specificità e copertura informativa.

Generation

I risultati ottenuti in fase di generazione riflettono in gran parte quanto osservato nella fase di retrieve.

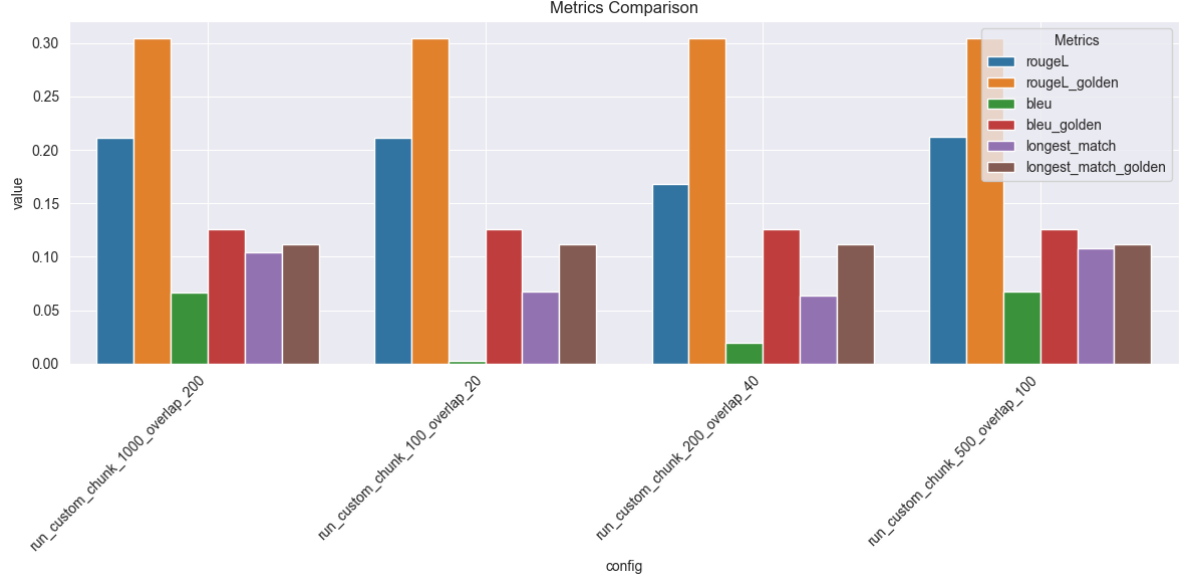


Figura 5.4: Test su dimensione dei chunk e overlap in fase di generazione

La differenza tra metriche standard e golden è evidente per rougeL e bleu, mentre è più contenuta, soprattutto nei modelli con chunk da 500 e 1000 token, per longest_match.

Si segnala che il modello con chunk di dimensione 2500 non è presente nei grafici, poiché non è stato possibile eseguirlo nella configurazione con generazione, a causa delle limitazioni computazionali imposte dalle API gratuite di HuggingFace.

5.1.3 Top K

I test sul parametro top-k in fase di generazione non mostrano variazioni significative nelle metriche.

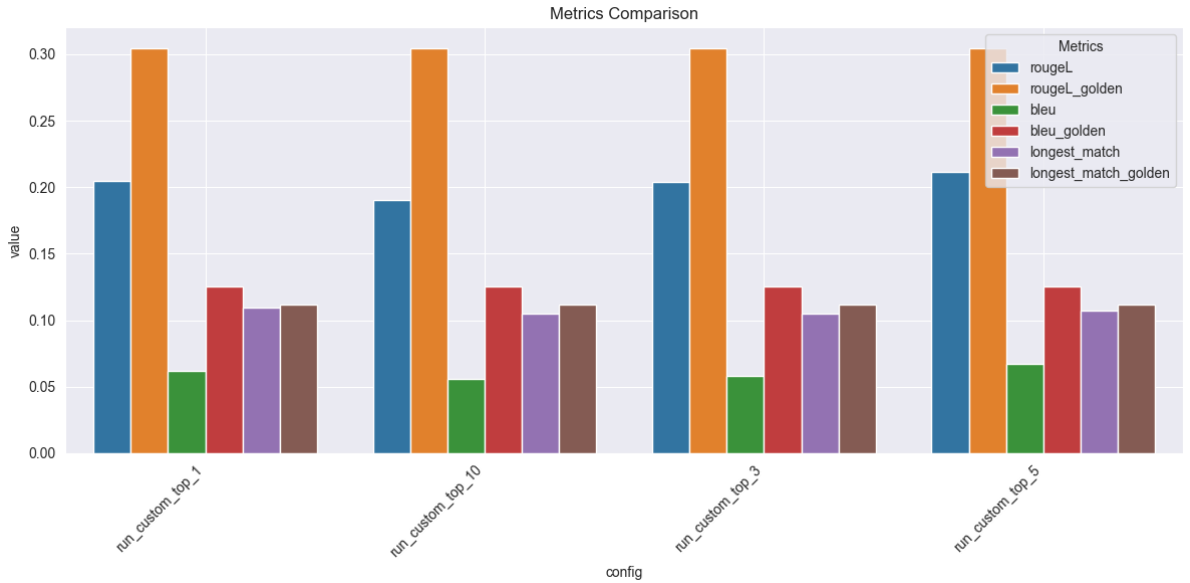


Figura 5.5: Test sul parametro Top-K in fase di generazione

Le metriche restano pressoché identiche sia selezionando un solo documento che dieci. Questo comportamento può essere spiegato dal fatto che la risposta generata si basa principalmente sulle prime informazioni utili nel contesto, rendendo marginale il contributo di documenti ulteriori.

5.1.4 Temperature & Max Tokens

Le variazioni di temperatura non influenzano in modo significativo le performance del sistema.

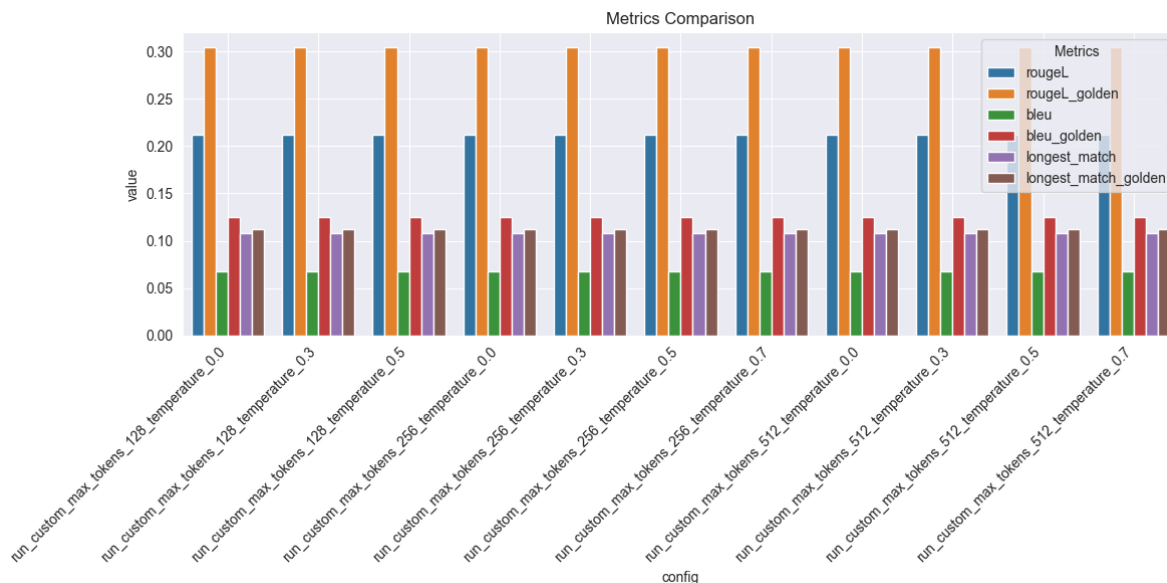


Figura 5.6: Test su temperature e max_tokens in fase di generazione

I valori delle metriche si mantengono costanti, suggerendo una relativa insensibilità alla randomizzazione introdotta dalla temperatura. Analogamente, modificare il valore massimo di token generabili non ha prodotto effetti rilevanti: i testi generati, in media, non raggiungono il limite massimo, e le metriche risultano stabili.

5.1.5 Hardware

Il confronto tra le due configurazioni hardware non hanno mostrato differenze a livello di prestazioni nè nella fase di retrieve che nella fase di generation. L'unica differenza tra le due è stata il tempo di esecuzione delle run, con un leggero favore fornito dalla configurazione i7-10700f-32GB

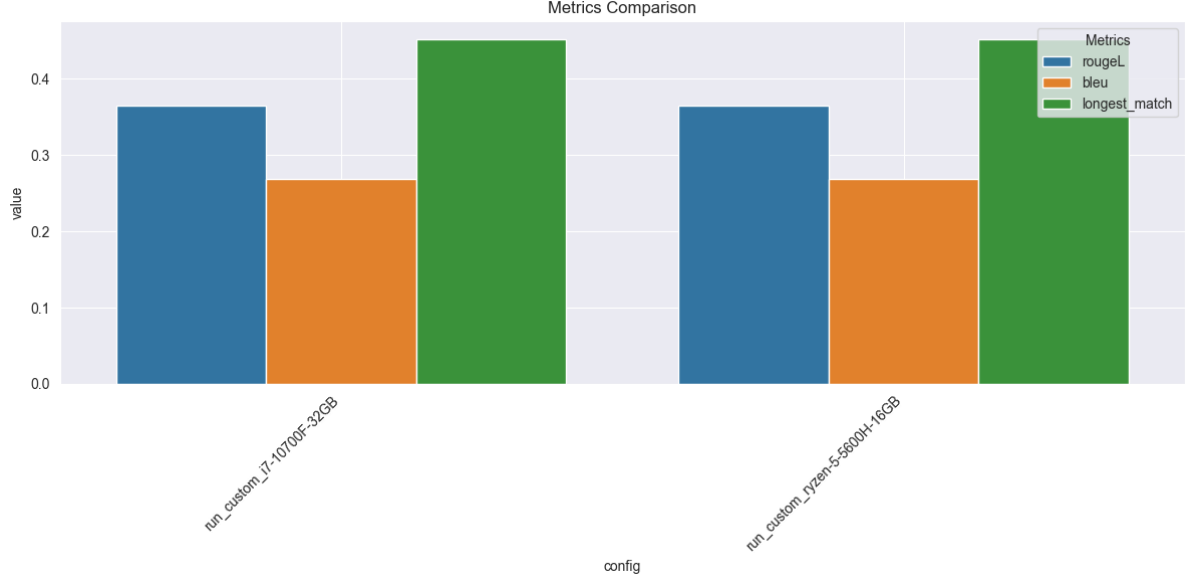


Figura 5.7: Test Hardware in fase di retrieve per la pipeline custom

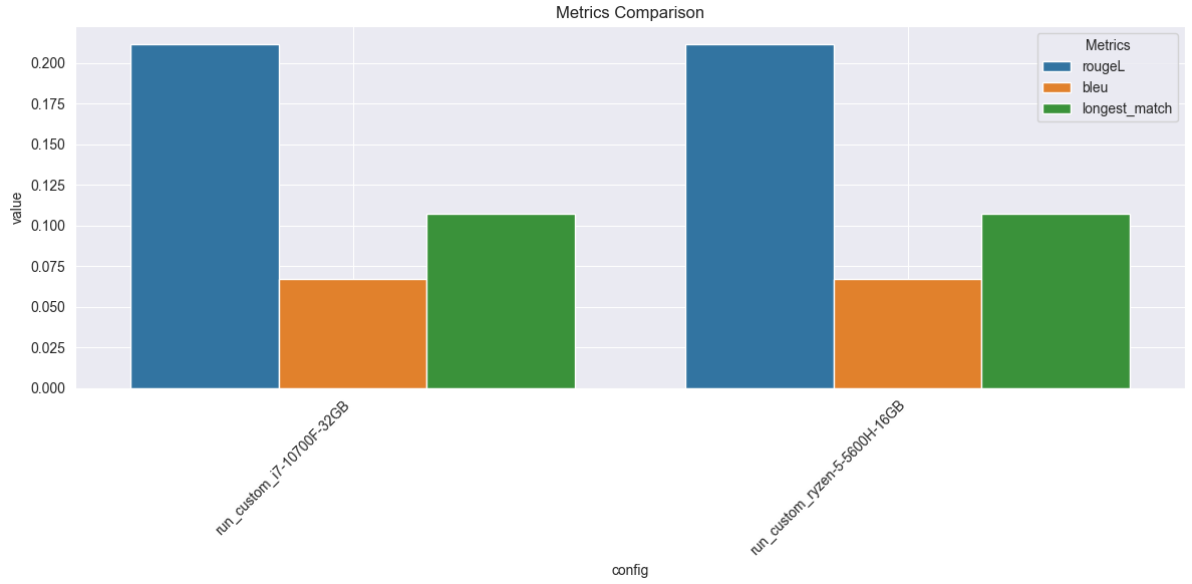


Figura 5.8: Test Hardware in fase di generazione per la pipeline custom

5.2 Analisi BEIR

Per valutare le prestazioni della pipeline BEIR su diverse configurazioni di modelli, sono state analizzate e tracciate le metriche mostrate nella Sezione 4.2.2. Queste metriche sono calcolate all'aumentare del valore di k , che assume i seguenti valori:

$$k = [1, 3, 5, 10]$$

5.2.1 Dataset

I risultati sperimentali evidenziano differenze significative nelle prestazioni tra le diverse configurazioni di dataset testate.

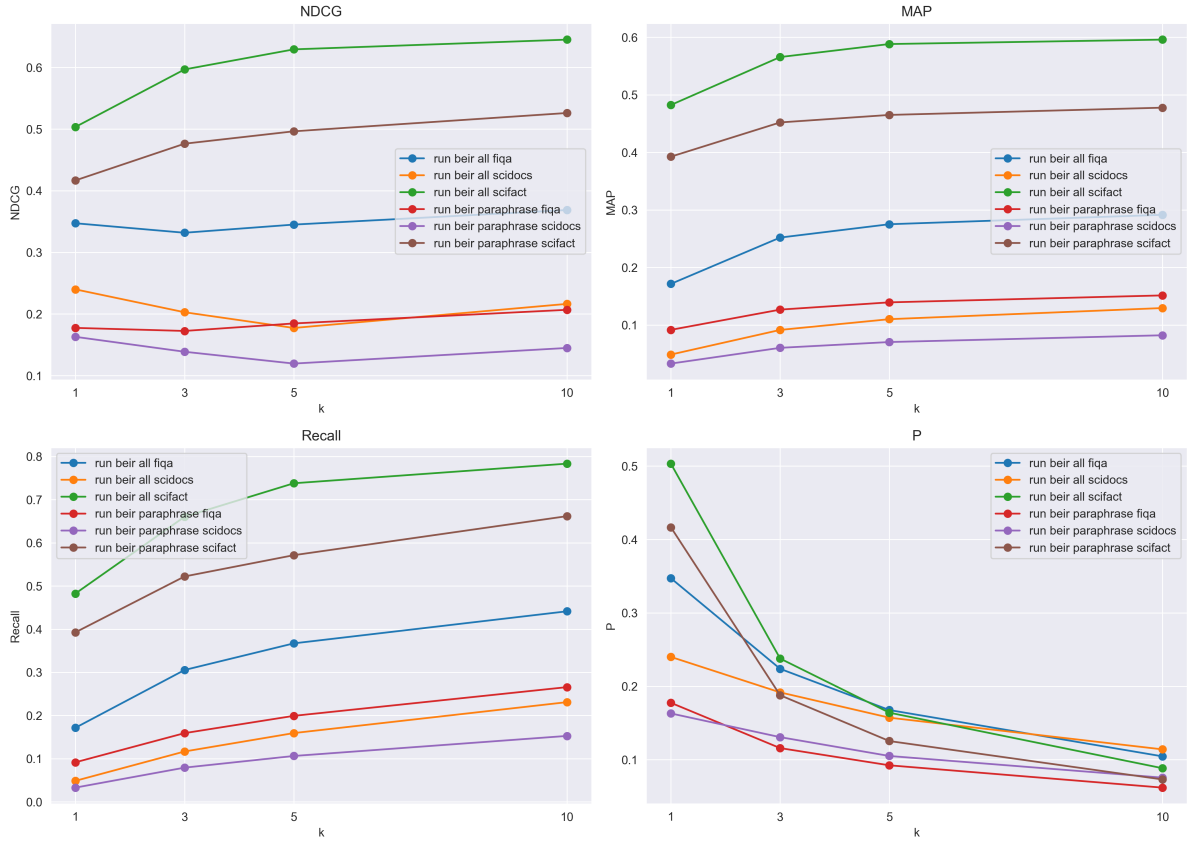


Figura 5.9: Test Datasets in fase di retrieve per la pipeline BEIR

Ogni dataset è stato valutato utilizzando due modelli di embedding: **all-MiniLM-L6-v2** e **paraphrase-MiniLM-L6-v2**. La configurazione ottimale, considerando tutte le metriche analizzate, corrisponde al dataset più leggero, **scifact**, associato al modello di embedding **all**. Tale configurazione è seguita da quella che mantiene fisso il dataset **scifact** passando all'embedding **paraphrase**.

Il dataset **FiQA** mostra risultati intermedi con il modello **all-MiniLM-L6-v2** e leggermente inferiori con il modello **paraphrase-MiniLM-L6-v2**. Le prestazioni peggiori si riscontrano nel dataset di dimensione intermedia, **scidocs**.

L'andamento dei risultati è coerente all'interno di ciascun dataset al variare del modello di embedding, con incrementi e decrementi paralleli su tutte le metriche.

Si rileva che, per le metriche **NDCG**, **MAP** e **Recall**, la gerarchia qualitativa delle configurazioni rimane sostanzialmente invariata. Al contrario, per la precisione (**P**), il modello che mantiene la precisione più elevata all'aumentare di k è quello associato al dataset **scidocs**, seguito da **FiQA**, mentre nel caso del dataset **scifact** si osserva un calo marcato.

5.2.2 Embedding Model

I risultati sperimentali rivelano prestazioni competitive tra i diversi modelli di embedding testati, con alcune configurazioni che emergono come particolarmente efficaci.

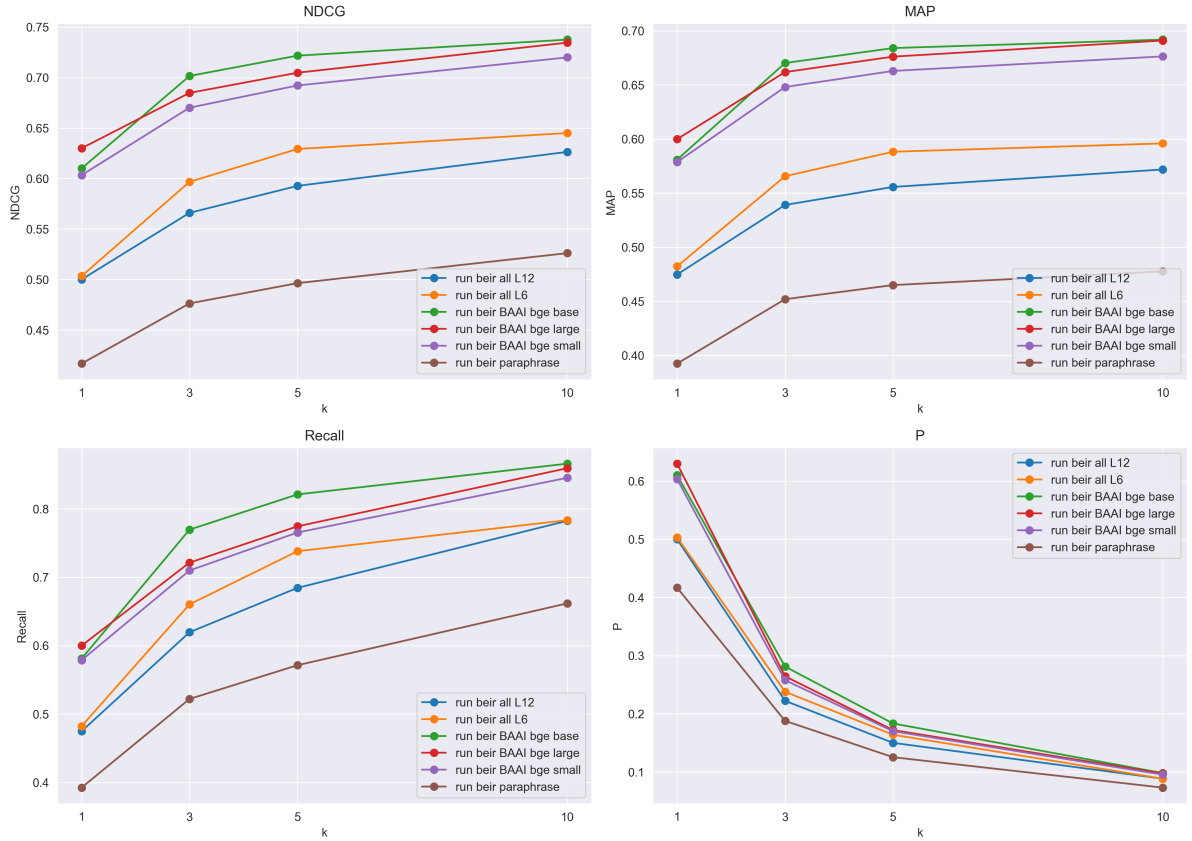


Figura 5.10: Test Embedding Model in fase di retrieve per la pipeline BEIR

Le configurazioni BAAI (bge-base, bge-large, bge-small) mostrano prestazioni superiori, con **bge-base** che raggiunge i migliori risultati complessivi. La configurazione **bge-large** presenta performance leggermente inferiori, ma comunque di livello eccellente, mentre **bge-small** ottiene risultati comparabili nonostante la sua dimensione ridotta rispetto alle altre varianti. La configurazione **all-MiniLM-L6-v2** evidenzia performance intermedie ma solide, mentre **all-MiniLM-L12-v2** registra prestazioni inferiori. La configurazione **paraphrase-MiniLM-L6-v2** conferma le prestazioni più basse tra quelle testate.

L'andamento delle metriche in funzione del parametro k rispecchia il comportamento atteso: un incremento monotono per NDCG, MAP e Recall, e una decrescita per la precisione. È interessante notare come, all'aumentare di k , i valori delle metriche tra configurazioni simili tendano a convergere, risultando più vicini rispetto a quanto osservato per valori intermedi di k .

5.2.3 Score Function

I risultati sperimentali confrontano l'importanza della metrica di similarità nell'ambito del retrieval, evidenziando prestazioni sostanzialmente diverse tra le due configurazioni testate.

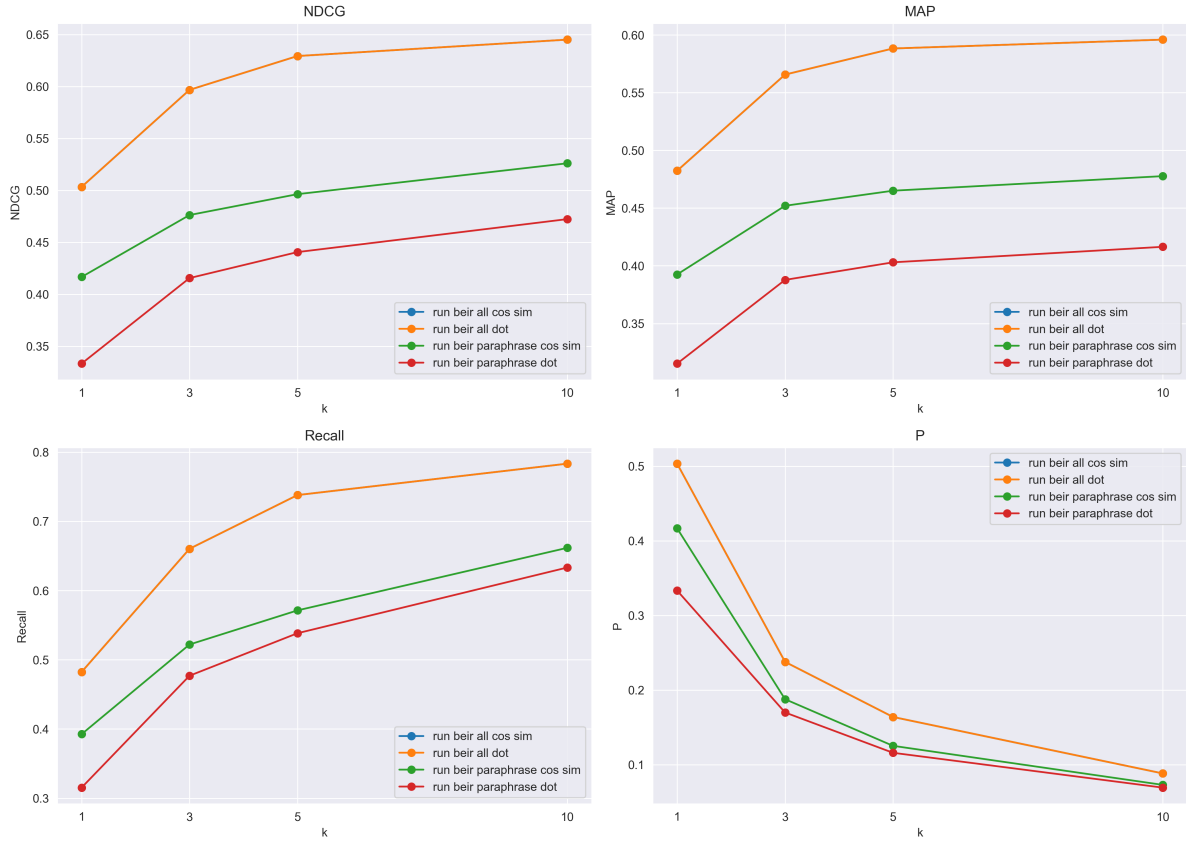


Figura 5.11: Test Score Function in fase di retrieve per la pipeline BEIR

Ciò che non risulta immediatamente evidente dai plot seguenti, ma che può essere osservato ispezionando i file di output, è che la funzione di scoring non determina differenze di prestazioni quando il modello di embedding utilizzato è `all-MiniLM-L6-v2`: infatti, la curva blu, corrispondente alla funzione di scoring `cos_sim`, risulta completamente sovrapposta a quella arancione della funzione `dot`.

Tale comportamento cambia al variare del modello di embedding: con `paraphrase-MiniLM-L6-v2` si ottengono risultati distinti per le due funzioni di scoring, con `cos_sim` che cattura in modo più efficace le relazioni semantiche rispetto a `dot`, sebbene entrambe le performance risultino significativamente inferiori rispetto a quelle ottenute con il modello `all-MiniLM-L6-v2`.

L'andamento delle metriche al variare del parametro k conferma il comportamento atteso: crescita monotona per NDCG, MAP e Recall, e decrescita per la precisione.

5.2.4 Batch Size

Le variazioni della dimensione del batch non influenzano le performance del sistema.

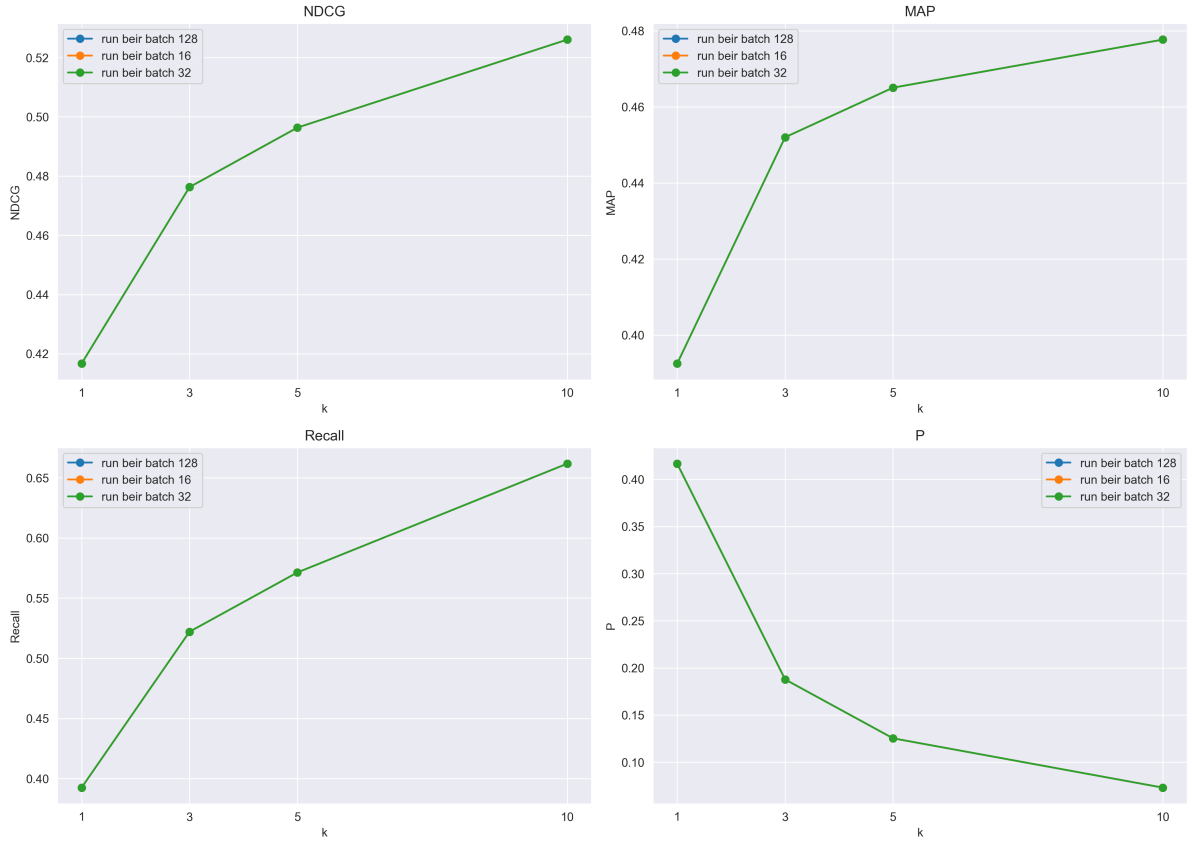


Figura 5.12: Test Batch Size in fase di retrieve per la pipeline BEIR

Per tutte le metriche analizzate, le diverse configurazioni di batch size (16, 32, 128) producono risultati sovrapponibili, suggerendo che tale iperparametro non incide sulle prestazioni del modello nell'intervallo sperimentato, ma impatta principalmente sui tempi di esecuzione della pipeline.

5.2.5 Hardware

I risultati sperimentali ottenuti su due diverse configurazioni hardware sono coerenti con quelli osservati per la pipeline custom.

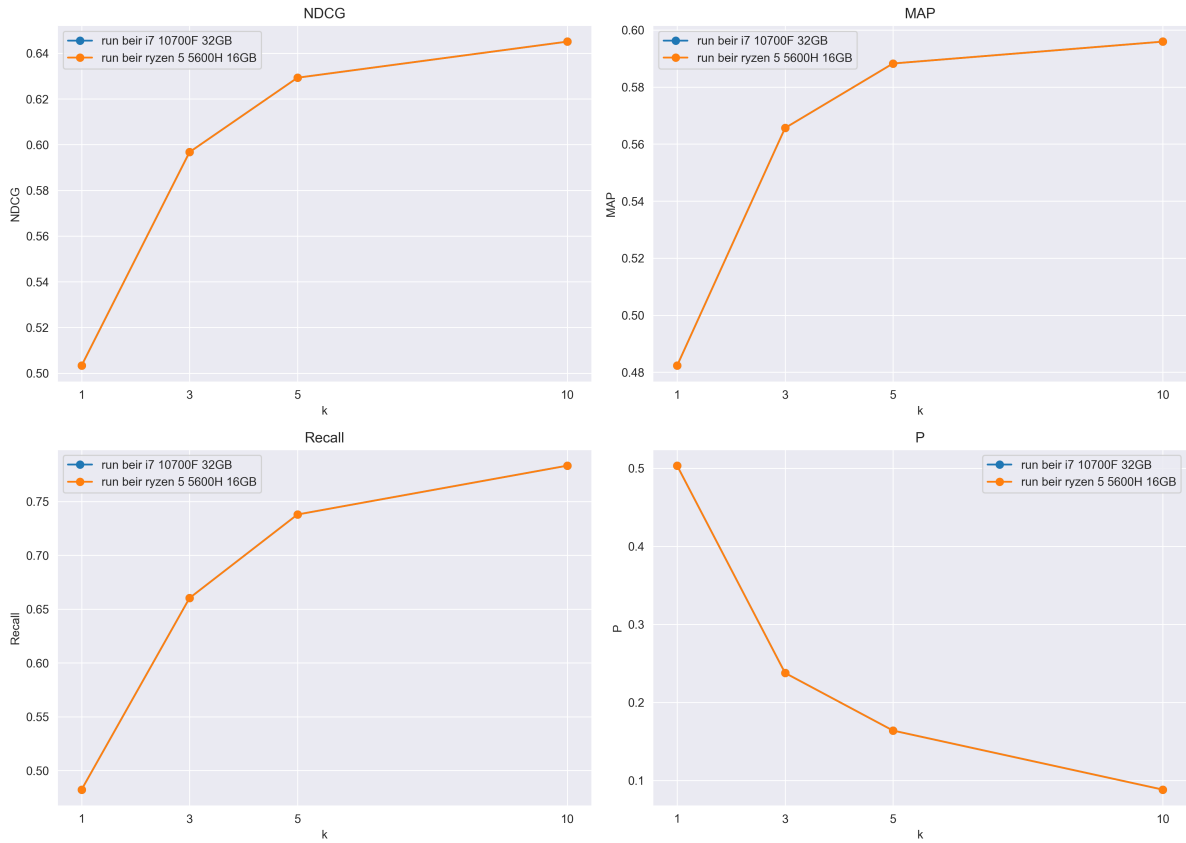


Figura 5.13: Test Hardware in fase di retrieve per la pipeline BEIR

I parametri di valutazione non risultano influenzati dalla configurazione hardware utilizzata, la quale incide invece principalmente sui tempi di esecuzione della pipeline.

Capitolo 6

Conclusioni e Sviluppi Futuri

6.1 Conclusioni a livello di sviluppo del processo

Il progetto ha permesso di sviluppare con successo due pipeline complementari per la valutazione di sistemi RAG, ciascuna con caratteristiche e finalità specifiche. La pipeline custom ha consentito un controllo granulare sui parametri del sistema, permettendo analisi approfondite sull’impatto di ogni componente sulla qualità finale delle risposte. La pipeline BEIR ha fornito un framework standardizzato per il confronto con lo stato dell’arte, garantendo riproducibilità e confrontabilità dei risultati.

Dal punto di vista metodologico, l’approccio adottato di variare singolarmente i parametri mantenendo gli altri costanti si è rivelato efficace per isolare l’impatto di ciascuna componente. La strutturazione modulare delle pipeline ha facilitato l’estensibilità del sistema e la conduzione di esperimenti sistematici.

La scelta di utilizzare file di configurazione JSON ha semplificato notevolmente la gestione degli esperimenti, consentendo una rapida iterazione tra diverse configurazioni e garantendo la tracciabilità completa dei parametri utilizzati in ogni test.

6.2 Conclusioni a livello di analisi e valutazione

I risultati ottenuti evidenziano diversi aspetti significativi delle performance dei sistemi RAG.

Importanza della qualità degli embedding: L’analisi conferma che la scelta del modello di embedding ha un impatto cruciale sulle prestazioni del sistema. In particolare, nella valutazione BEIR, le configurazioni BAAI (bge-base, bge-large, bge-small) hanno mostrato prestazioni superiori rispetto ai modelli MiniLM. Questo risultato è coerente con le osservazioni sulla pipeline custom, dove il modello bge-base si è dimostrato il più efficace.

Dimensione ottimale dei chunk: i risultati dell’analisi relativa alla dimensione dei chunk evidenziano come dimensioni eccessivamente ridotte compromettano in modo significativo le prestazioni, mentre dimensioni troppo elevate tendano a includere informazioni non rilevanti, determinando un peggioramento delle performance complessive.

Influenza del dataset e della configurazione di retrieval: La valutazione BEIR ha evidenziato differenze significative nelle prestazioni tra i dataset testati, con il dataset più leggero (scifact) associato al modello di embedding all che ottiene i risultati migliori su tutte le metriche, seguito da FiQA con performance intermedie e scidocs che registra le prestazioni peggiori. All’interno di ciascun dataset, i risultati sono coerenti al variare del modello di embedding, con variazioni parallele nelle metriche.

Insensibilità ad alcuni iperparametri: La valutazione BEIR ha evidenziato che la configurazione del batch size (testato con valori 16, 32 e 128) non influisce in modo significativo sulle metriche di prestazione, limitando il proprio impatto ai tempi di esecuzione della pipeline. Inoltre, i test su diverse configurazioni hardware confermano che la qualità delle prestazioni non è influenzata dall'hardware utilizzato, che invece determina variazioni principalmente nei tempi di calcolo.

Differenze tra retrieval e generazione: Dai risultati preliminari sulla pipeline custom emerge che modelli con prestazioni inferiori nel retrieval possono comunque produrre generazioni di qualità accettabile, sottolineando la complessità e la non linearità dell'interazione tra le due fasi del processo RAG.

6.3 Limiti del sistema attuale

Il sistema sviluppato presenta alcune limitazioni che ne circoscrivono l'applicabilità e la completezza dell'analisi:

Limitazioni del dataset: L'utilizzo di un sottoinsieme ristretto del dataset Natural Questions (solo 10 esempi validi) può non essere rappresentativo della variabilità reale delle domande e delle performance del sistema. Questa scelta, dettata da vincoli computazionali e di costo API, limita la significatività statistica dei risultati.

Vincoli computazionali: La scelta del modello generativo è stata fortemente condizionata dai costi delle API, limitando la sperimentazione a **Phi-3.5-mini-instruct**. Modelli più performanti come **Mistral-7B** o **GPT-4** potrebbero offrire risultati significativamente diversi.

Manca di integrazione: Le due pipeline sviluppate utilizzano approcci di ricerca diversi (FAISS con similarity search vs. exact search BEIR), rendendo difficile un confronto diretto e equo delle prestazioni.

Prompt engineering limitato: Il sistema utilizza un prompt fisso senza esplorazione di strategie alternative di prompt engineering, che potrebbero migliorare significativamente le performance generative.

Monitoraggio delle risorse: L'assenza di metriche dettagliate sull'utilizzo di CPU e RAM limita la comprensione dell'efficienza computazionale del sistema in scenari reali.

6.4 Sviluppi futuri

Per superare i limiti identificati e migliorare l'efficacia del sistema, si propongono le seguenti direzioni di sviluppo:

- **Ottimizzazione hardware:**
 - Implementazione del supporto GPU per FAISS per accelerare significativamente le operazioni di similarity search.
 - Confronto sistematico delle performance CPU vs GPU per quantificare i benefici dell'accelerazione hardware.
- **Prompt engineering avanzato:**

- Sperimentazione con diverse strategie di prompt engineering.
- Sviluppo di prompt adattivi che si modificano in base al tipo di domanda o al dominio specifico.
- **Configurazioni sperimentali estese:**
 - Superamento del rapporto fisso 1:5 tra chunk overlap e chunk size per esplorare configurazioni più granulari.
 - Estensione degli esperimenti a dataset completi con migliaia di esempi per garantire significatività statistica.
- **Diversificazione dei modelli:**
 - Utilizzo di API key premium per testare modelli generativi più avanzati senza limitazioni di rate limiting.
 - Confronto sistematico tra diverse famiglie di LLM (GPT, Claude, Llama, Mistral) per identificare i modelli più adatti a task specifici.
- **Integrazione e standardizzazione:**
 - Integrazione di FAISS come backend per la pipeline BEIR per garantire confronti equi tra i sistemi.
 - Implementazione di metriche comparative standardizzate tra le due pipeline.
- **Monitoraggio avanzato:**
 - Implementazione di profiling dettagliato per CPU, RAM e GPU durante l'esecuzione.
 - Sviluppo di dashboard real-time per il monitoraggio delle performance del sistema.
 - Analisi del trade-off tra qualità dei risultati e risorse computazionali utilizzate.

Bibliografia

- [1] Simone Filice et al. *Generating Diverse QA Benchmarks for RAG Evaluation with DataMorgana*. 2025. arXiv: 2501.12789 [cs.CL]. URL: <https://arxiv.org/abs/2501.12789> (cit. a p. 1).
- [2] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL]. URL: <https://arxiv.org/abs/2312.10997> (cit. a p. 3).
- [3] Zexuan Ji, Nayeon Lee, Jason Fries et al. «Survey of Hallucination in Natural Language Generation». In: *ACM Computing Surveys* (2023). URL: <https://arxiv.org/abs/2302.03629> (cit. a p. 4).
- [4] Jeff Johnson, Matthijs Douze e Hervé Jégou. «Billion-scale similarity search with GPUs». In: *IEEE Transactions on Big Data* 7.3 (2019), pp. 535–547. URL: <https://arxiv.org/abs/1702.08734> (cit. a p. 3).
- [5] Vladimir Karpukhin et al. «Dense Passage Retrieval for Open-Domain Question Answering». In: *EMNLP 2020*. 2020. URL: <https://arxiv.org/abs/2004.04906> (cit. alle pp. 3, 4).
- [6] Patrick Lewis et al. «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks». In: *Advances in Neural Information Processing Systems*. A cura di H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf (cit. a p. 3).
- [7] Patrick Lewis et al. «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks». In: *NeurIPS 2020* 33 (2020), pp. 9459–9474. URL: <https://arxiv.org/abs/2005.11401> (cit. a p. 4).
- [8] Nelson F. Liu et al. «Lost in the Middle: How Language Models Use Long Contexts». In: *arXiv preprint arXiv:2307.03172* (2023). URL: <https://arxiv.org/abs/2307.03172> (cit. a p. 4).
- [9] Nandan Thakur et al. *BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models*. 2021. arXiv: 2104.08663 [cs.IR]. URL: <https://arxiv.org/abs/2104.08663> (cit. a p. 6).
- [10] Sriram Veturi et al. *RAG based Question-Answering for Contextual Response Prediction System*. 2024. arXiv: 2409.03708 [cs.CL]. URL: <https://arxiv.org/abs/2409.03708> (cit. a p. 1).