



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	Preprocessing e Chunking	3
2.2	Generazione degli Embedding	4
2.3	Indicizzazione nel Vector DB	4
2.4	Embedding della Query	4
2.5	Similarity Search	4
2.6	Recupero del Contesto	4
2.7	Generazione della Risposta	4
2.8	Metriche di Valutazione nei Sistemi Q&A	4
2.8.1	Benchmark Standard	5
2.8.2	Benchmark Personalizzati	5
3	Architettura del Sistema	6
3.1	Pipeline di Benchmarking	6
3.2	Pipeline Custom	6
3.2.1	Configurazione	7
3.2.2	Prompt Engineering	8
3.2.3	Fasi della Pipeline	8
3.2.4	Modularità e Funzioni Principali	9
3.2.5	Librerie e Strumenti Utilizzati	9
3.2.6	Dataset	10
3.3	Pipeline BEIR	12
3.3.1	Dataset	13
3.3.2	Embedding	13
3.3.3	Retrieval	13
3.3.4	Valutazione	13
3.3.5	Librerie e Strumenti Utilizzati	14
4	Metodologia e Configurazioni Sperimentali	15
4.1	Obiettivo	15
4.2	Strategia Sperimentale	15
4.3	Metriche	15
4.3.1	Metriche Retrieval	15
4.3.2	Metriche Generation	16
4.4	Configurazioni Sperimentali	16
4.4.1	Configurazione BEIR	17
4.5	Output della Pipeline	18

4.5.1	Output Custom	18
4.5.2	Output BEIR	18
4.6	Plotter e Analisi	18
4.7	Riproducibilità e Tracciabilità	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	24
5.1.4	Temperature & Max Tokens	25
5.1.5	Prompt Engineering	27
5.2	Analisi BEIR	28
5.2.1	Embedding Model	28
5.2.2	Score Function	28
5.2.3	Dataset	29
6	Conclusioni e Sviluppi Futuri	31
6.1	Conclusioni su Sviluppo e Metodologia	31
6.2	Conclusioni Pipeline Custom	31
6.3	Conclusioni Pipeline BEIR	31
6.4	Limiti del sistema attuale	32
6.5	Sviluppi futuri	32
	Bibliografia	33

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, 7], 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.

I test sperimentali hanno evidenziato prestazioni promettenti: modelli di embedding come *all-mpnet-base-v2* e *e5-base-v2* hanno raggiunto precisione fino al 60% e 50% rispettivamente per $k=1$, mentre nella fase di generazione sono stati ottenuti punteggi ROUGE-L fino a 0.83 e similarità coseno fino a 0.91, dimostrando l'efficacia dell'approccio RAG implementato.

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;
- Cartella contenente i file latex per la generazione della relazione;
- file delle dipendenze python;
- il README del progetto in formato MarkDown;
- la seguente relazione in formato PDF.
- la presentazione di tale progetto 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

In questo capitolo vengono presentati i principali concetti teorici alla base dei sistemi Retrieval-Augmented Generation (RAG), seguendo la pipeline tipica illustrata in Figura 2.1.

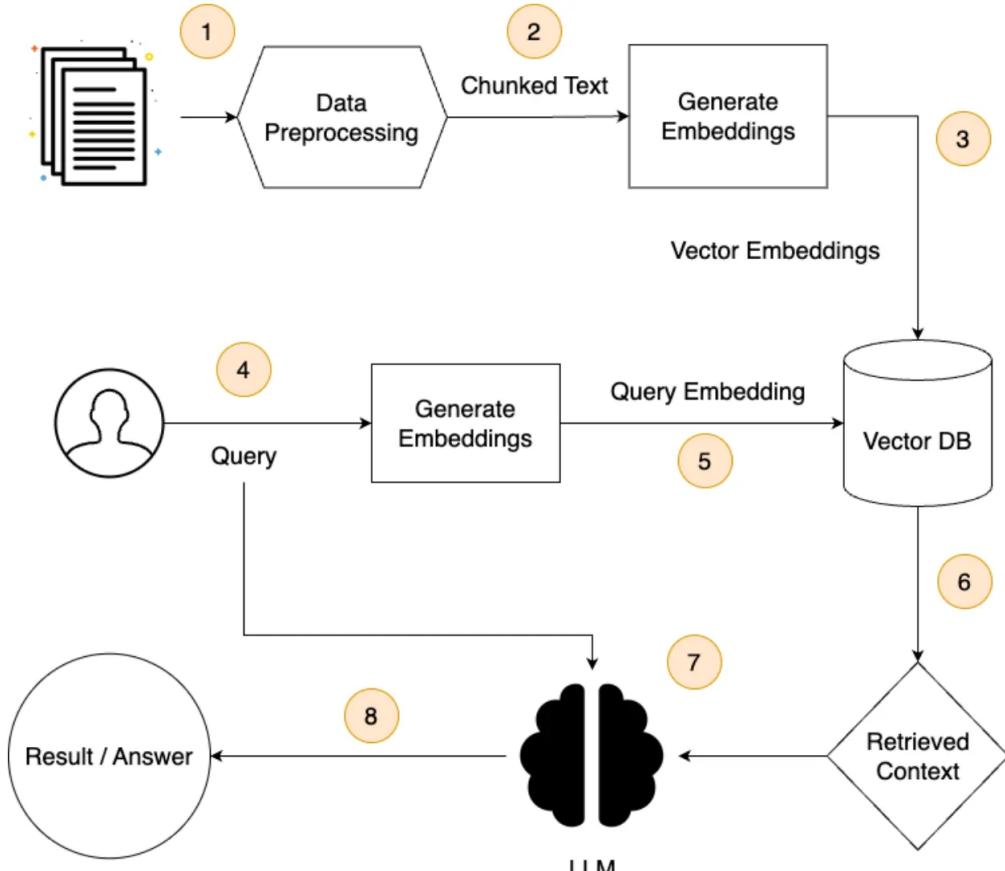


Figura 2.1: Schema generale di una pipeline Retrieval-Augmented Generation (RAG).

2.1 Preprocessing e Chunking

La prima fase di una pipeline RAG consiste nella preprocessazione dei dati e nella suddivisione dei documenti in segmenti (chunk) di dimensione controllata. Questa operazione consente di gestire testi di grandi dimensioni e di adattarli ai limiti di token dei modelli. Spesso si applica una sovrapposizione tra chunk consecutivi per preservare la coerenza informativa [5].

2.2 Generazione degli Embedding

Ogni chunk viene trasformato in una rappresentazione vettoriale densa (embedding) tramite modelli specializzati. Gli embedding catturano il significato semantico dei testi e sono fondamentali per la fase di recupero. La qualità degli embedding influenza direttamente l'efficacia della ricerca [4].

2.3 Indicizzazione nel Vector DB

Gli embedding generati vengono memorizzati in un database vettoriale (Vector DB), come FAISS [3], che consente una ricerca efficiente tra milioni di vettori tramite tecniche di indicizzazione e compressione.

2.4 Embedding della Query

Quando viene posta una domanda, anche questa viene convertita in embedding tramite lo stesso modello utilizzato per i documenti, garantendo la coerenza nello spazio semantico.

2.5 Similarity Search

L'embedding della query viene confrontato con quelli dei chunk memorizzati nel Vector DB tramite misure di similarità (ad esempio, cosine similarity o dot product). Si selezionano così i chunk più rilevanti rispetto alla domanda.

2.6 Recupero del Contesto

I chunk selezionati costituiscono il contesto recuperato, che verrà fornito al modello generativo. È importante controllare che la quantità di testo non superi la finestra di contesto del modello LLM.

2.7 Generazione della Risposta

Il contesto recuperato e la domanda vengono inseriti in un prompt che viene passato a un Large Language Model (LLM). Il design del prompt è cruciale per evitare fenomeni di allucinazione e per ottenere risposte accurate e aderenti al contesto [2].

2.8 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.

2.8.1 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.

2.8.2 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) [6], 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, sebbene un confronto tra le fase di retrieve può essere effettuato. 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 stata sviluppata per offrire massima flessibilità e modularità nella valutazione di sistemi RAG. L'intera pipeline è guidata da un file di configurazione JSON che permette di specificare:

- Modello di embedding e LLM da utilizzare;
- Parametri di chunking (dimensione, overlap);
- Numero di documenti da recuperare (top-k);
- Parametri del modello generativo (temperature, max_new_tokens, tipo di prompt);
- Modalità di esecuzione: solo retrieval o retrieval+generazione.

3.2.1 Configurazione

Tutti i parametri chiave della pipeline sono definiti in un file di configurazione (ad esempio, `custom_benchmark_config.json`). Ecco un esempio e la spiegazione dei parametri principali:

```
1  {
2      "dataset_split": "train[100:1000]",
3      "output_folder": "output/generation/max_new_tokens",
4      "run_name": "max_new_tokens_512",
5      "embedding_model_name":
6          "sentence-transformers/all-MiniLM-L6-v2",
7      "chunk_size": 500,
8      "chunk_overlap": 100,
9      "top_k": 5,
10     "k_values": [1, 3, 5, 10],
11     "num_valid_examples": 10,
12     "llm_model": "HuggingFaceTB/SmolLM3-3B",
13     "temperature": 1.0,
14     "max_new_tokens": 1024,
15     "prompt_type": "basic",
16     "only_retrieve": false
17 }
```

Ogni parametro è pensato per essere facilmente modificabile, consentendo esperimenti rapidi su architetture, modelli e strategie diverse.

dataset_split: sottoinsieme del dataset da usare

output_folder: cartella di output

run_name: nome identificativo del run

embedding_model_name: modello di embedding da usare

chunk_size: dimensione dei chunk di testo

chunk_overlap: overlap tra chunk consecutivi

top_k: numero di documenti recuperati

k_values: valori di k per metriche di retrieval

num_valid_examples: numero di esempi validi da processare

llm_model: modello LLM per la generazione

temperature: temperatura del modello generativo

max_new_tokens: numero massimo di token generati

prompt_type: tipo di prompt da usare

only_retrieve: se true, solo retrieval; se false, retrieval+generazione

3.2.2 Prompt Engineering

Un aspetto chiave della pipeline è la possibilità di selezionare diversi template di prompt per la generazione, definiti in un modulo Python dedicato (`prompt_templates.py`). Questo permette di testare facilmente strategie di prompt engineering, tra cui:

- **basic**: risposta secca basata solo sul contesto;
- **cot**: chain-of-thought, ragionamento passo-passo;
- **few_shot**: esempi di risposte corrette/errate;
- **structured**: output strutturato con quote, reasoning, confidence, answer;
- **constrained**: risposte con vincoli (lunghezza, unità, incertezza);
- **role**: risposta come se fosse un assistente esperto;
- **self_verify**: auto-verifica della correttezza della risposta.

Esempio di selezione e utilizzo del prompt:

```
1 from prompt_templates import get_prompt_messages
2 messages = get_prompt_messages(prompt_type="cot", context=contesto,
3     query=domanda)
4 response = chat_model.invoke(messages)
```

3.2.3 Fasi della Pipeline

1. **Preprocessing e Chunking**: il testo viene estratto dai token del dataset, rimuovendo markup HTML e suddiviso in chunk sovrapposti tramite `RecursiveCharacterTextSplitter`, con parametri configurabili.

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

2. **Embedding e Indicizzazione**: ogni chunk viene convertito in embedding tramite `SentenceTransformers` e indicizzato in un vector store FAISS.

```
1 embedding_model = HuggingFaceEmbeddings(
2     model_name=config["embedding_model_name"])
3 vectorstore = FAISS.from_documents(split_docs, embedding_model)
```

3. **Retrieval**: data una query, viene calcolato l'embedding e viene effettuata una similarity search per recuperare i chunk più rilevanti (`top_k`).

```
1 docs_faiss = vectorstore.similarity_search(query,
2     k=config['top_k'])
```

4. **Generazione (opzionale)**: se abilitata (`only_retrieve: false`), il contesto recuperato viene fornito a un LLM per generare la risposta, usando il prompt selezionato. Il modello, la temperatura, il numero massimo di token e il tipo di prompt sono tutti configurabili.

```

1 llm = HuggingFaceEndpoint(
2     repo_id=config["llm_model"],
3     huggingfacehub_api_token=HF_API_KEY,
4     task="text-generation",
5     temperature=config["temperature"],
6     max_new_tokens=config["max_new_tokens"],
7 )

```

5. **Valutazione:** vengono calcolate metriche di retrieval, cioè precision@k, recall@k, MAP@k, nDCG@k, e se la generazione è attiva, metriche di generazione (ROUGE-L, BLEU, similarità coseno tra embedding, longest match).
6. **Valutazione semantica:** lo stesso LLM viene usato per rileggere la risposta e fornire una valutazione semantica automatica, confrontando la predizione con il contesto e la risposta “golden”.

```

1 rouge = metric_rouge.compute(predictions=[prediction],
2                               references=[golden_context])
2 bleu = metric_bleu.compute(predictions=[prediction],
3                             references=[golden_context])
3 cosine_similarity = util.pytorch_cos_sim(pred_embeddings,
4                                           golden_embeddings)

```

Per ogni predizione, viene anche generata una risposta “golden” usando il contesto ideale, per confronto diretto.

7. **Output e Analisi:** tutti i risultati sono salvati in formato CSV/JSON, separando metriche di retrieval, generazione, risposte, documenti recuperati e configurazione. Un notebook dedicato (`custom_benchmark_plotter.ipynb`) consente di visualizzare e confrontare facilmente le metriche tra diverse configurazioni.

3.2.4 Modularità e Funzioni Principali

Il codice è organizzato in funzioni dedicate per ogni step (preprocessing, estrazione risposte, retrieval, generazione, valutazione, salvataggio output), facilitando la manutenzione e l'estensione della pipeline.

Esempio di funzione di valutazione completa:

```

1 def process_sample(i, sample):
2     # ... retrieval ...
3     # ... generazione ...
4     # Calcolo metriche: precision, recall, MAP, nDCG, ROUGE-L,
5     #                   BLEU, cosine similarity, longest match, valutazione semantica
6     return result

```

3.2.5 Librerie e Strumenti Utilizzati

- **LangChain:** orchestrazione pipeline, interfacce per modelli e vectorstore
- **HuggingFace Transformers/Hub:** accesso a modelli preaddestrati e API

- **SentenceTransformers**: embedding dei documenti
- **FAISS**: indicizzazione e ricerca approssimata
- **Evaluate**: calcolo metriche
- **Datasets (HuggingFace)**: caricamento e gestione dataset
- **TQDM, Pandas, NumPy**: supporto per progress bar, manipolazione dati, calcoli numerici

3.2.6 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:

`(id, document, question, long_answer_candidates, annotations)`

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 optionalmente 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"],
4                                             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
14     long_answer = ""
15     if sample["annotations"]["long_answer"][0]["start_token"] != -1:
16         start =
17             sample["annotations"]["long_answer"][0]["start_token"]
18         end = sample["annotations"]["long_answer"][0]["end_token"]
19         long_answer = " ".join([
20             t for t, html in zip(tokens["token"][start:end],
21                     tokens["is_html"][start:end])]
22             if not html
23         ])
24
25     return long_answer or "", short_answer or ""

```

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(sbtrt,  
    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**. L'obiettivo è analizzare in modo sistematico l'impatto dei principali parametri della pipeline, sia custom che BEIR, attraverso una serie di esperimenti controllati e riproducibili.

4.2 Strategia Sperimentale

Per ogni esperimento, viene variato un solo parametro alla volta, mantenendo costanti tutti gli altri, secondo il paradigma dell'analisi univariata. Questo approccio consente di attribuire con maggiore precisione eventuali variazioni nelle metriche di performance al parametro in esame.

Tutti gli esperimenti sono eseguiti tramite notebook Python che possono essere eseguiti in modalità **retrieval only**, dove solo la fase di retrieval viene eseguita, e in modalità normale, dove viene eseguita l'intera pipeline. Sono stati utilizzati file di configurazione JSON, che definiscono in modo esplicito i parametri di ciascuna run. Al termine di ogni esecuzione, vengono prodotti file di output strutturati (CSV/JSON) che raccolgono le varie metriche, le risposte generate e la configurazione utilizzata, garantendo la tracciabilità e la riproducibilità degli esperimenti.

4.3 Metriche

4.3.1 Metriche Retrieval

Le metriche di valutazione utilizzate da entrambe le pipeline durante la fase di retrieval 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.

4.3.2 Metriche Generation

Le prestazioni della pipeline custom durante la fase di generazione sono state valutate utilizzando le seguenti metriche, scelte per analizzare le risposte sia a livello sintattico che semantico:

- **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 alla risposta di riferimento, includendo una penalità per risposte troppo brevi.
- **Cosine Similarity**: similarità coseno tra gli embedding della risposta generata e della risposta di riferimento.
- **Longest Match Ratio**: rapporto tra la lunghezza della sottostringa continua più lunga condivisa tra contesto e predizione, e la lunghezza totale del contesto.
- **Valutazione Semantica LLM**: la risposta generata viene riletta dallo stesso LLM, che fornisce una valutazione automatica della correttezza semantica rispetto al contesto.

4.4 Configurazioni Sperimentali

Parametri Analizzati e Motivazione

I parametri scelti per l'analisi sperimentale sono stati selezionati in base alla loro rilevanza nella letteratura e al loro impatto atteso sulle prestazioni del sistema. Di seguito si riassumono le motivazioni per ciascun parametro:

- **Modello di embedding**: la qualità e la natura degli embedding influenzano direttamente la capacità del sistema di recuperare documenti rilevanti. Sono stati testati modelli di diversa dimensione e architettura per valutare il trade-off tra accuratezza e velocità.
- **Chunk size e overlap**: la segmentazione del testo determina la granularità del contesto fornito al sistema. Chunk troppo piccoli rischiano di frammentare l'informazione, mentre chunk troppo grandi possono introdurre rumore e ridurre la pertinenza.
- **Top-k**: il numero di documenti recuperati influenza la quantità e la qualità del contesto fornito al generatore. Valori diversi di k permettono di studiare il bilanciamento tra copertura e pertinenza.
- **Temperature e max tokens**: questi parametri controllano la creatività e la lunghezza delle risposte generate dal modello LLM, influenzando sia la coerenza che la completezza delle risposte.
- **Tipo di prompt**: la pipeline consente di testare diversi template di prompt, per valutare l'impatto del prompt engineering sulla qualità delle risposte.

Esperimenti Condotti

Gli esperimenti condotti sulla pipeline custom includono:

- **Variazione del modello di embedding:** sono stati testati diversi modelli (ad es. `all-MiniLM-L6-v2`, `bge-base-en-v1.5`, `e5-base-v2`, ecc.) per valutare l'impatto sulla fase di retrieval.
- **Variazione di chunk size:** sono stati eseguiti esperimenti con chunk di dimensioni diverse: 100, 200, 500, 1000, 2500.
- **Variazione di chunk overlap:** sono stati eseguiti esperimenti su diversi overlap, basati sulla dimensione del chunk: $1/10$, $1/5$, $1/2$, $>1/2$.
- **Variazione di top-k:** sono stati testati valori di k pari a 1, 3, 5, 10 per analizzare la quantità di contesto ottimale da fornire al generatore.
- **Variazione di temperature:** sono stati testati valori di temperatura pari a 0.0, 0.2, 0.5, 0.7, 1.0, 1.2, 2.0.
- **Variazione max tokens:** sono stati testati valori di `max_new_tokens` pari a 32, 64, 128, 256, 512.
- **Prompt engineering:** sono stati testati diversi template di prompt (basic, chain-of-thought, few-shot, structured, constrained, role, self-verify) per valutare l'impatto sulla robustezza e precisione delle risposte.

Tabella 4.1: Riepilogo degli esperimenti condotti sulla pipeline custom

Parametro	Valori Testati	Obiettivo
Modello di embedding	<code>all-MiniLM-L6-v2</code> , <code>bge-base-en-v1.5</code> , <code>e5-base-v2</code> , ecc.	Valutare l'impatto del modello di embedding sulla fase di retrieval
Chunk size	100, 200, 500, 1000, 2500	Analizzare l'effetto della segmentazione del testo sulla qualità del contesto
Chunk overlap	$1/10$, $1/5$, $1/2$, $>1/2$	Analizzare l'effetto della segmentazione del testo sulla qualità del contesto
Top-k	1, 3, 5, 10	Studiare la quantità ottimale di contesto da fornire al generatore
Temperature	0.0, 0.2, 0.5, 0.7, 1.0, 1.2, 2.0	Analizzare l'effetto sulla creatività e coerenza delle risposte
Max tokens	32, 64, 128, 256, 512, 1024	Analizzare l'effetto sulla lunghezza delle risposte
Tipo di prompt	basic, chain-of-thought, few-shot, structured, constrained, role, self-verify	Valutare l'impatto del prompt engineering sulla robustezza e precisione delle risposte

4.4.1 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 4.1: Configurazione Pipeline BEIR

Gli esperimenti previsti sulla pipeline BEIR includono:

- **Variazione del dataset:** SciFact, SCIDOCS, FiQA-2018, per testare la robustezza su domini diversi.
- **Variazione del modello di embedding:** per valutare l'impatto sulla fase di retrieval.
- **Variazione della funzione di similarità:** confronto tra `cos_sim` e `dot`.
- **Variazione di top-k:** fissato a [1, 3, 5, 10] per tutte le metriche.

4.5 Output della Pipeline

4.5.1 Output Custom

Al termine dell'esecuzione del notebook, la pipeline custom produce diversi file di output, ciascuno con uno scopo preciso:

- `generation_metrics.csv`: valori delle metriche di generazione per ciascun esempio valutato.
- `retrieval_metrics.csv`: metriche di retrieval aggregate.
- `used_config.json`: configurazione completa della run, per garantire riproducibilità.
- `responses.json`: dettagli delle risposte generate e delle valutazioni semantiche per ogni esempio.
- `retrieved_docs.json`: documenti rilevanti e documenti recuperati per ogni query.
- `queries.json`: domande, risposte gold e contesti di riferimento.

4.5.2 Output BEIR

L'output della pipeline BEIR consiste di:

- `results.csv`: metriche BEIR (NDCG, MAP, recall, precision) per ogni valore di k .
- `used_config.json`: configurazione della run.

4.6 Plotter e Analisi

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 di output. Per

le metriche di generazione della pipeline custom, il confronto tra i modelli testati è stato rappresentato mediante un *bar plot*; per la fase di retrieval della pipeline custom e per la pipeline BEIR, sono stati creati dei *line chart* delle metriche al variare di k .

4.7 Riproducibilità e Tracciabilità

Ogni esperimento è documentato tramite il file di configurazione salvato insieme ai risultati. La modularità del codice e la separazione tra pipeline, metriche e plotter facilitano la ripetizione degli esperimenti e l'estensione a nuovi parametri o strategie.

Capitolo 5

Analisi dei Risultati

5.1 Analisi Custom

In questa sezione vengono analizzati in dettaglio i risultati ottenuti dalla pipeline custom, con particolare attenzione all'impatto dei principali parametri sperimentati sia nella fase di retrieval che in quella di generazione. L'analisi si basa sui file di output prodotti dagli esperimenti (`retrieval_metrics.csv`, `generation_metrics.csv`, `responses.json`, ecc.), che permettono di confrontare in modo oggettivo le diverse configurazioni testate.

5.1.1 Embedding Model

In questo esperimento sono stati confrontati sette diversi modelli di embedding, ciascuno con caratteristiche architettonali e addestramento differenti:

- **all-MiniLM-L6-v2**: modello leggero e molto veloce, ottimizzato per semantic search in inglese.
- **bge-base-en-v1.5**: modello recente, ottimizzato per una migliore generalizzazione e performance su vari dataset.
- **e5-base-v2**: modello addestrato specificamente per compiti di retrieval, con ottime prestazioni su benchmark di domanda-risposta.
- **all-distilroberta-v1**: versione distillata di RoBERTa, più leggera ma comunque efficace per semantic similarity.
- **all-mpnet-base-v2**: modello di dimensioni maggiori, noto per la sua accuratezza nella rappresentazione semantica dei testi.
- **multi-qa-MiniLM-L6-cos-v1**: modello ottimizzato per domande e risposte, con supporto multilingue.
- **paraphrase-multilingual-MiniLM-L12-v2**: modello multilingue, adatto a compiti di parafrasi e retrieval in più lingue.

Come possibile notare dalla Figura 5.1, i modelli più prestanti risultano essere 3 in particolare: **all-mpnet-base-v2** (quello con dimensioni maggiori), **e5-base-v2**, (quello ottimizzato per compiti di retrieval) e **all-MiniLM-L6-v2** (ottimizzato per semantic search).

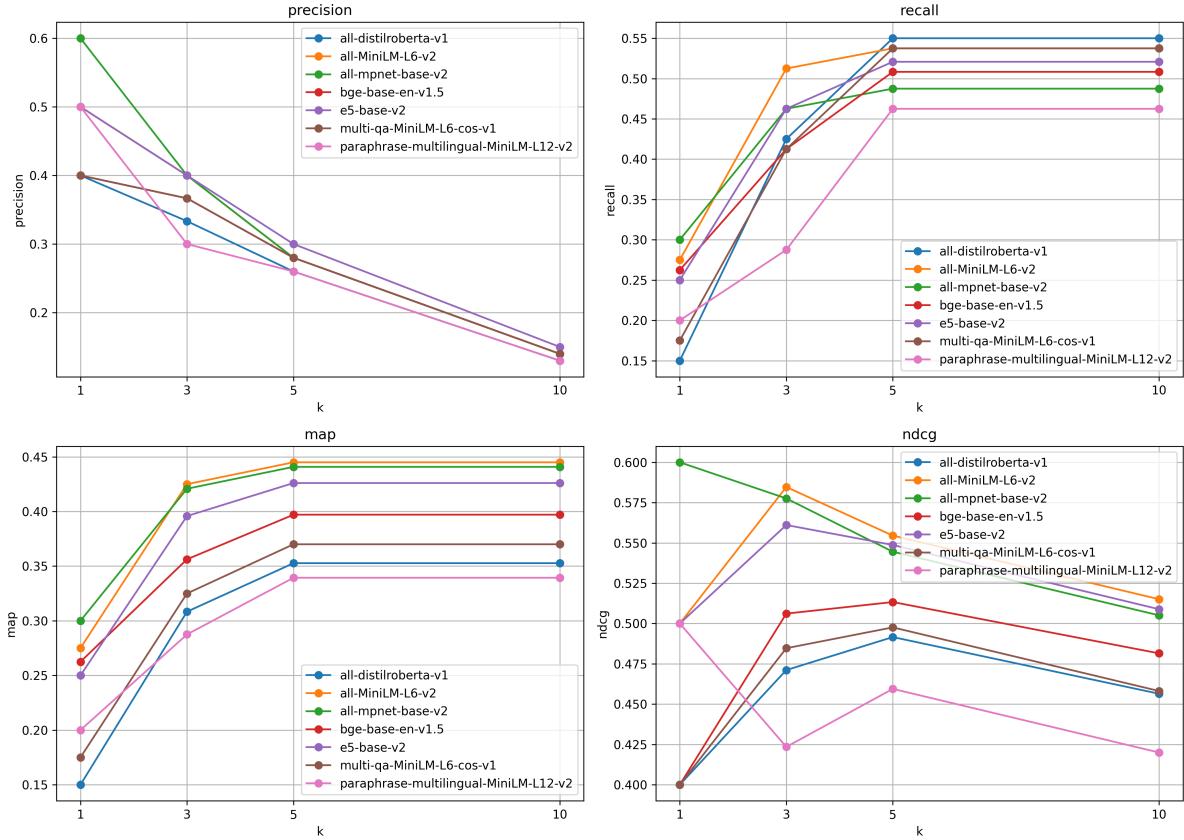


Figura 5.1: Confronto delle prestazioni dei diversi modelli di embedding nella fase di retrieval

5.1.2 Chunk Size & Overlap

In questo esperimento sono state testate cinque dimensioni diverse del chunk size:

- 100
- 200
- 500
- 1000
- 2500

e per ogni chunk size, è stato variato anche il chunk overlap, il quale assume valori proporzionali al primo:

- 1/10
- 1/5
- 1/2
- più di 1/2

Testando diverse combinazioni, abbiamo potuto valutare quale configurazione massimizza la qualità del contesto fornito al generatore, bilanciando granularità e completezza.

In Figura 5.3 possiamo osservare che la precision mostra un comportamento decrescente, poiché aumentando il numero di documenti considerati si inseriscono anche documenti non rilevanti rispetto alla risposta. Anche la NDCG mostra lievi cali, dovuti alle posizioni dei documenti recuperati. Gli altri due parametri, MAP e recall, presentano invece un andamento crescente, in linea con le aspettative.

Per il chunk size 200, il miglior overlap è risultato essere 20, ovvero la frazione più piccola tra quelle considerate.

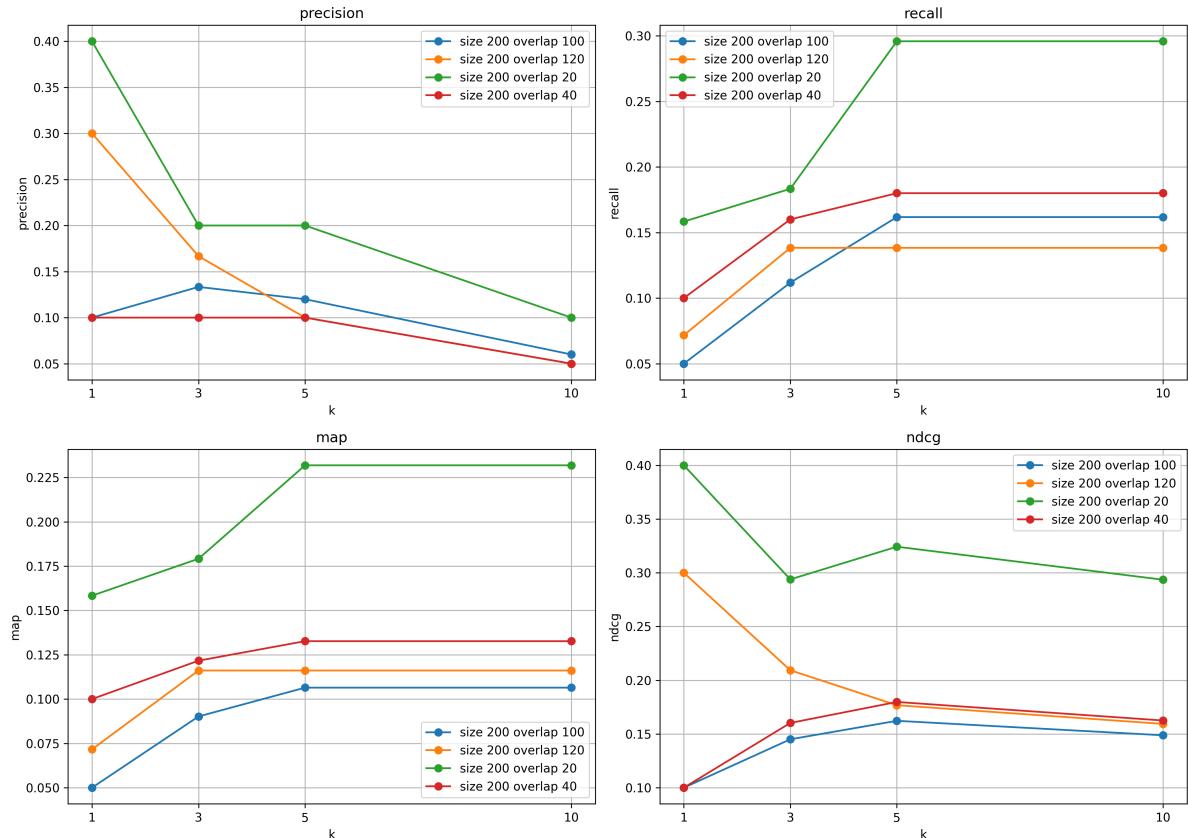


Figura 5.2: Analisi dell'impatto della dimensione dell'overlap su chunk di lunghezza 200

Nella figura 5.3, sono riportate per ogni dimensione analizzata, la combinazione con il chunk overlap migliore. I risultati migliori sono senza dubbio ottenuti con il chunk size più grande, mostrando una tendenza chiara: all'aumentare della dimensione del chunk, le prestazioni migliorano.

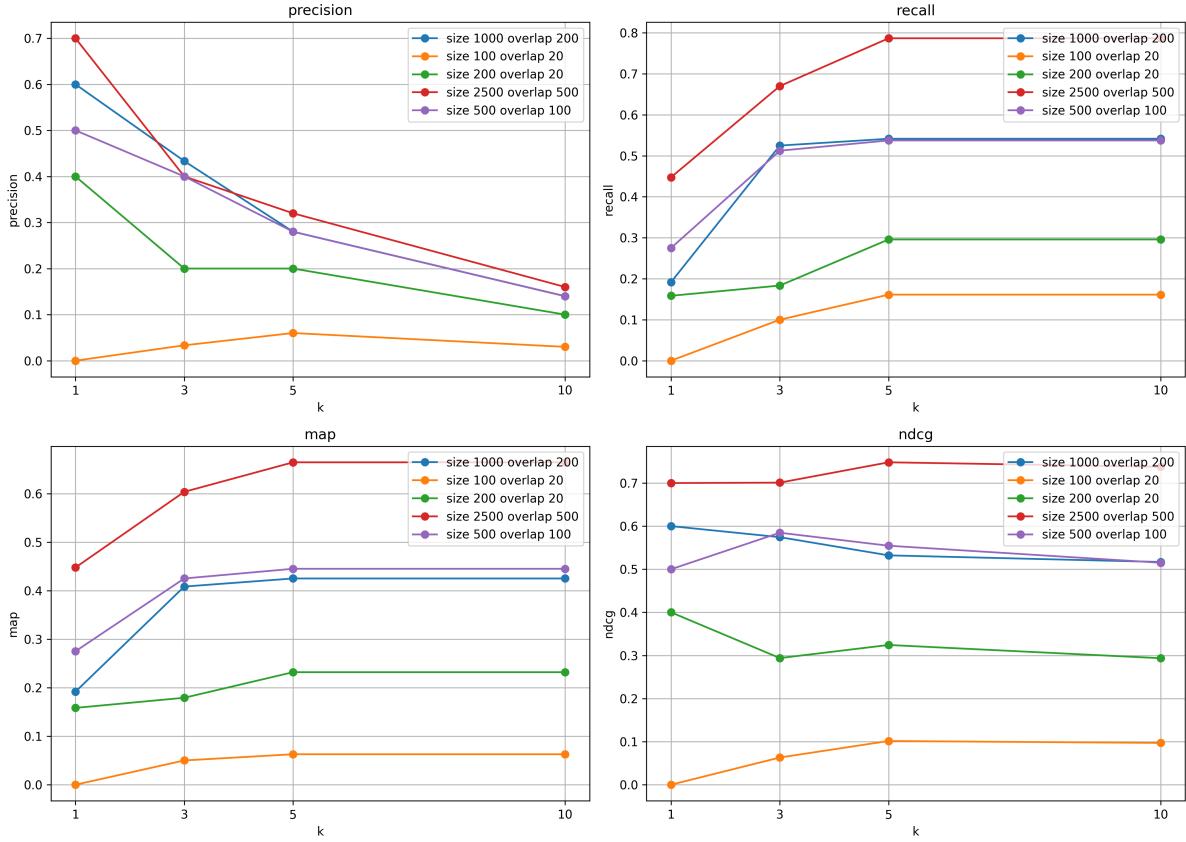


Figura 5.3: Analisi generale dell'impatto della dimensione dei chunk e dell'overlap

La qualità delle risposte è confermata dal plot delle metriche di generazione relative alle migliori esecuzioni per ciascun chunk size, mostrato in Figura 5.4. Si osserva che i valori più alti delle metriche standard si ottengono con chunk di dimensioni maggiori.

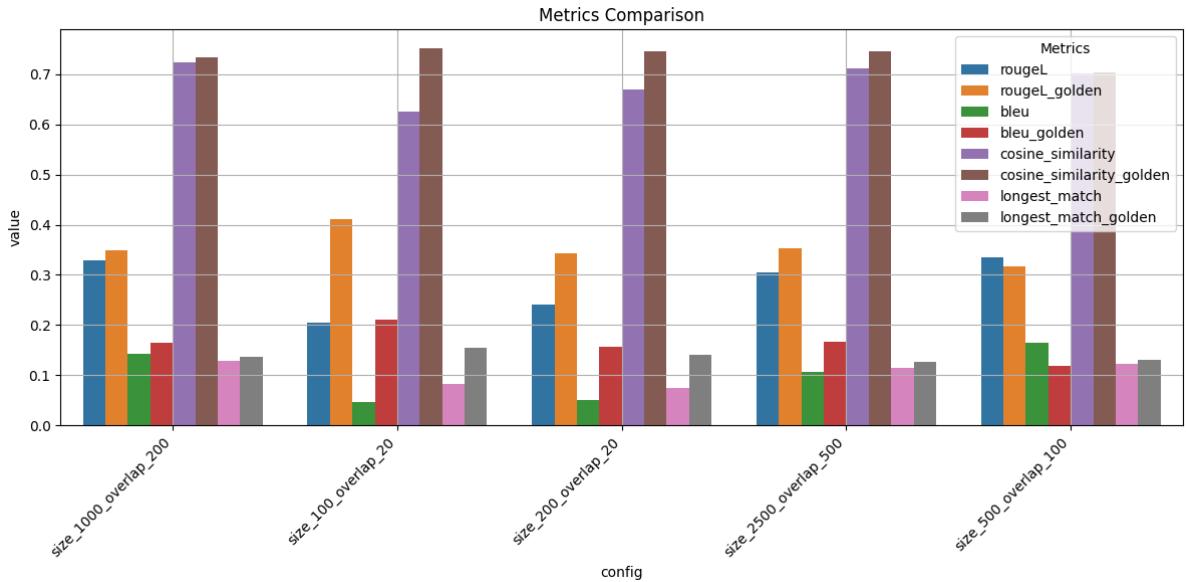


Figura 5.4: Analisi dell'effetto della dimensione e overlap dei chunk sulla fase di generazione

5.1.3 Top K

Analizziamo come il parametro relativo al numero di documenti recuperati influenzi il modello generativo. In particolare, si cerca di individuare la quantità ottimale di contesto da fornire al modello.

Come è possibile osservare in Figura 5.5, i risultati ottenuti al variare del k sono molto simili, indicando che le informazioni rilevanti alle risposte si trovano nei primi documenti recuperati.

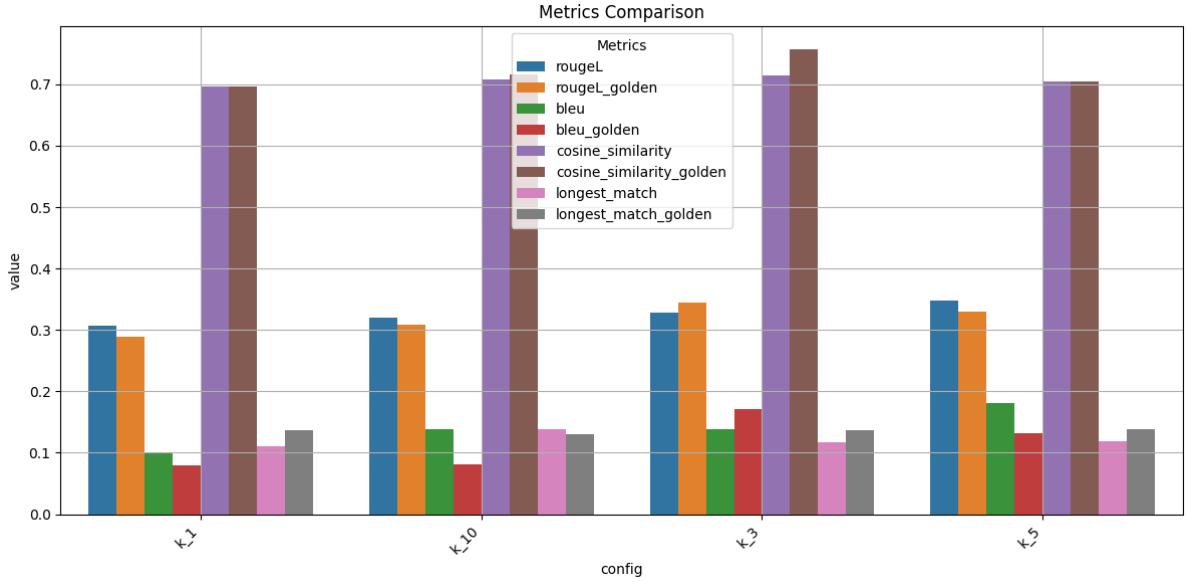


Figura 5.5: Analisi dell'impatto del parametro top-k sulla qualità della generazione

Osservando le metriche di retrieval per i diversi valori di top-k in Figura 5.6, si conferma quanto ipotizzato: ad eccezione di k=1, che risulta troppo basso, gli altri valori offrono prestazioni simili. Pertanto, k=3 rappresenta un buon compromesso tra quantità di documenti recuperati e qualità delle prestazioni.

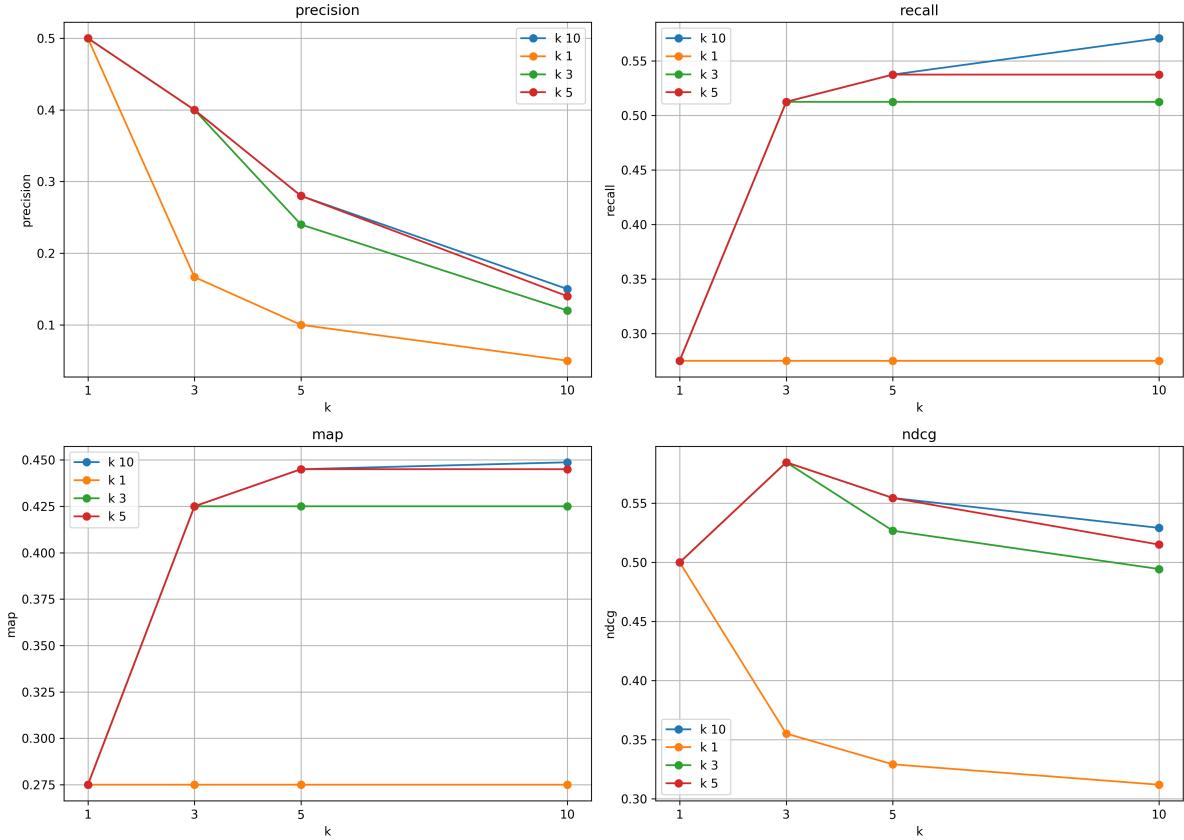


Figura 5.6: Analisi dell'impatto del parametro top-k sulla qualità della generazione

5.1.4 Temperature & Max Tokens

Abbiamo variato la temperatura, che controlla la casualità delle risposte generate dal modello: temperature basse producono risposte più deterministiche e coerenti, temperature alte favoriscono creatività e varietà.

Nel nostro esperimento abbiamo testato anche temperature superiori a 1, precisamente 1.2 e 2.0. Sebbene la temperatura sia comunemente compresa tra 0 e 1, valori più alti sono tecnicamente accettati da molti modelli di language generation. Temperature superiori a 1 rendono la generazione ancora più casuale e imprevedibile, portando spesso a risposte meno coerenti ma potenzialmente più creative.

In Figura 5.7 possiamo confrontare direttamente le metriche di generazione al variare della temperatura del modello. Sebbene non vi siano differenze notevoli, è importante sottolineare come temperature leggermente superiori sembrano offrire prestazioni migliori, anche se usando valori troppo alti tornano a peggiorare.

Il valore ottimale per la metrica ROUGE-L si ottiene con temperatura 0.5, mentre per BLEU e cosine similarity la temperatura ideale è 1.0, e per longest match è 2.0.

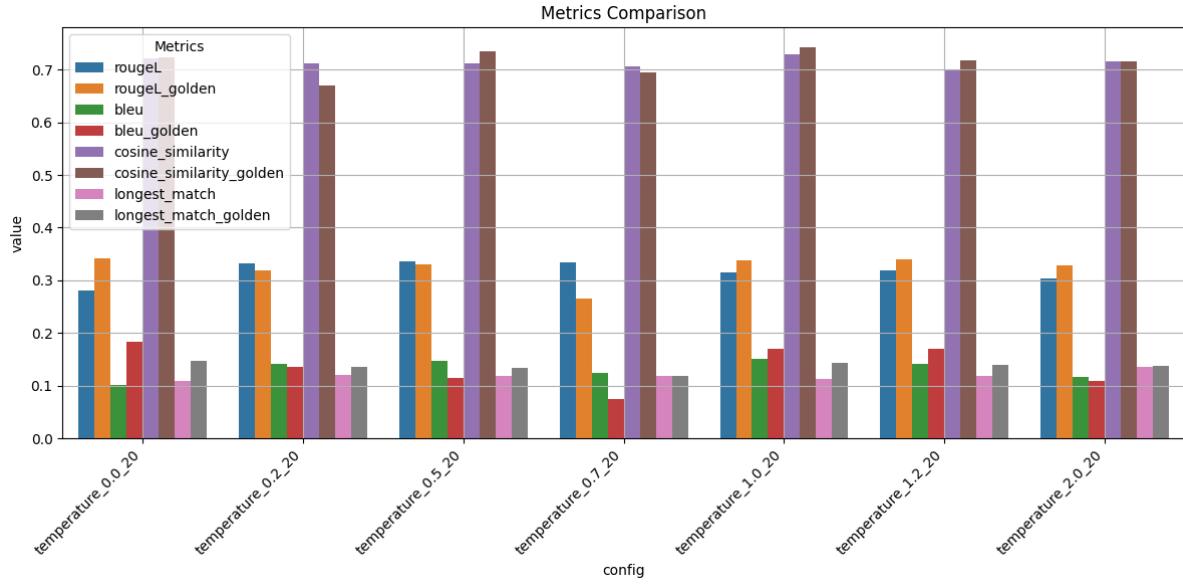


Figura 5.7: Analisi dell’impatto della temperatura sulla qualità della generazione.

Il parametro max new tokens limita la lunghezza massima della risposta generata. Testando diverse combinazioni, abbiamo valutato come questi parametri influenzano la completezza, la coerenza e la qualità delle risposte, cercando il miglior compromesso tra sintesi e dettaglio.

In Figura 5.8 possiamo osservare come il parametro max new tokens non influenzi in modo significativo le metriche di generazione.

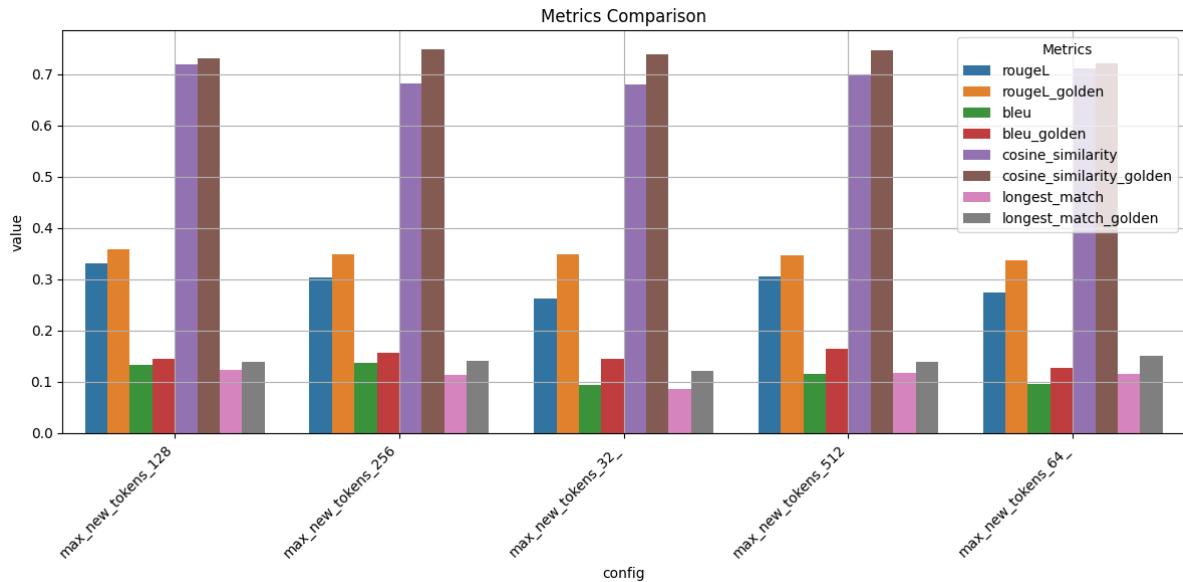


Figura 5.8: Analisi dell’impatto del numero massimo di token generabili sulla qualità della generazione

5.1.5 Prompt Engineering

Successivamente sono stati sperimentati diversi template di prompt, ognuno con una logica diversa:

- Basic: prompt semplice e diretto.
- Chain-of-thought: guida il modello a ragionare passo-passo.
- Few-shot: fornisce esempi di domanda-risposta.
- Structured: impone una struttura precisa alla risposta.
- Constrained: come detto prima in temperature, si richiede di rispondere entro un tot di parole, oltre ad altre regole da seguire.
- Role: assegna un ruolo specifico al modello. Nel nostro caso il modello rappresenta un esperto di dominio.
- Self-verify: chiede al modello di verificare la propria risposta.

L'obiettivo era capire come il prompt engineering possa migliorare la robustezza e la precisione delle risposte, adattando la strategia al tipo di domanda o dominio.

In Figura 5.9 si osserva che il migliore dal punto di vista sintattico è structured output, offrendo il migliore rouge e il miglior bleu. Il migliore dal punto di vista semantico, o almeno da quanto possiamo osservare tramite cosine similarity, sembra essere constrained.

Il prompt di base mostra risultati simili a quelli ottenuti con i template few-shot o role-based, indicando che queste strategie non influenzano significativamente la qualità dell'output.

Le strategie che invece peggiorano i risultati sono chain-of-thought e self-verification, con metriche quasi sempre inferiori rispetto al prompt base

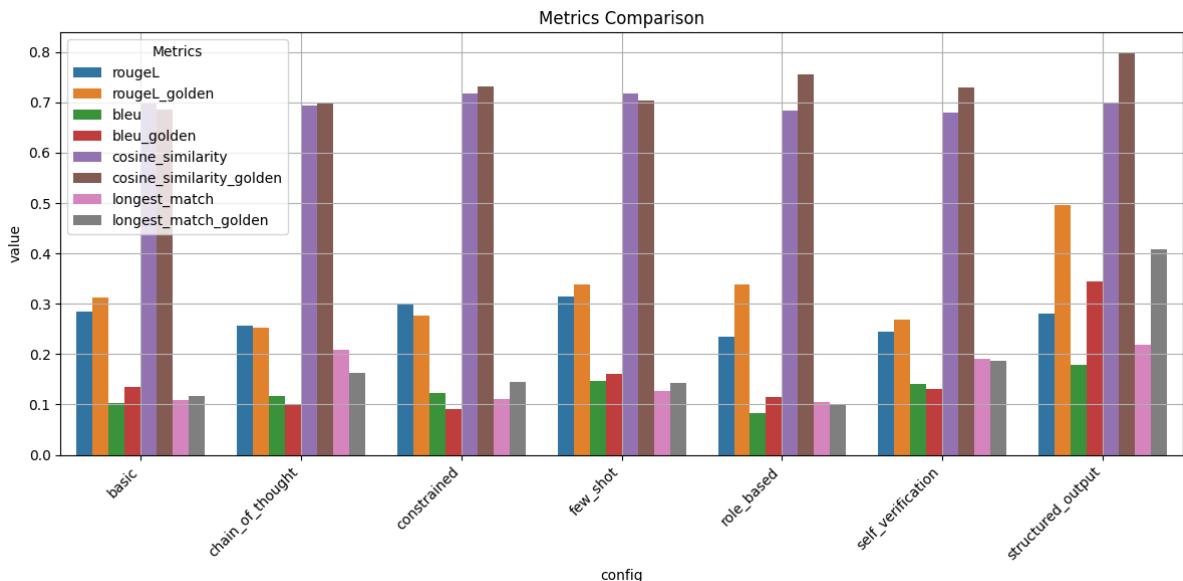


Figura 5.9: Confronto delle prestazioni dei diversi tipi di prompt nella fase di generazione

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 4.3.1. Queste metriche sono calcolate all'aumentare del valore di k , che assume i seguenti valori:

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

5.2.1 Embedding Model

I modelli di embedding testati hanno offerto prestazioni diverse rispetto alla pipeline custom. In particolare, bge-base-en-v5 e e5-base-v2 sono emersi come modelli migliori rispetto a tutte le metriche, come possiamo osservare in Figura 5.10. I modelli migliori della pipeline custom si collocano invece in una posizione intermedia. Anche i modelli meno performanti mostrano risultati coerenti con quanto osservato nella pipeline custom

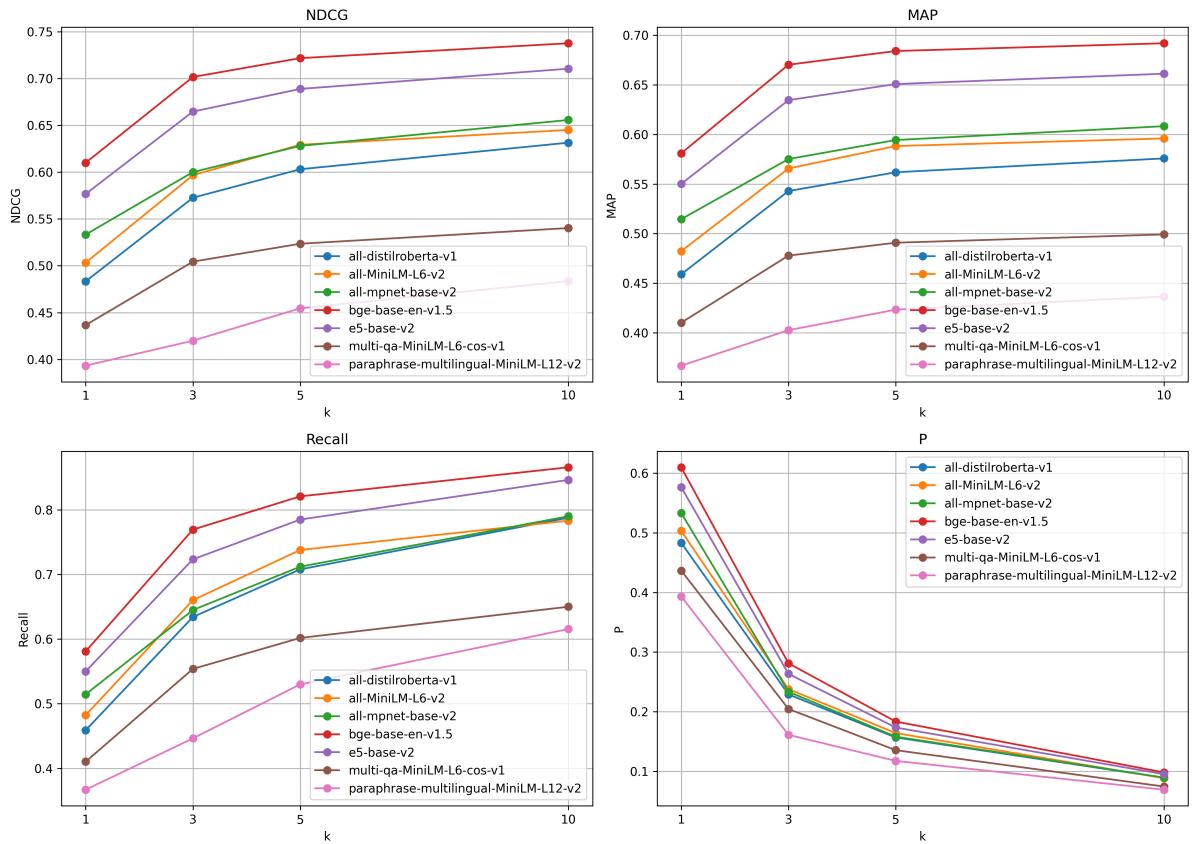


Figura 5.10: Test Embedding Model in fase di retrieve per la pipeline BEIR

5.2.2 Score Function

In Figura 5.11 possiamo osservare come la funzione di scoring influenzi le prestazioni. In particolare, la funzione di scoring `cos_sim` risulta essere migliore rispetto alla funzione `dot`, che offre prestazioni leggermente peggiori.

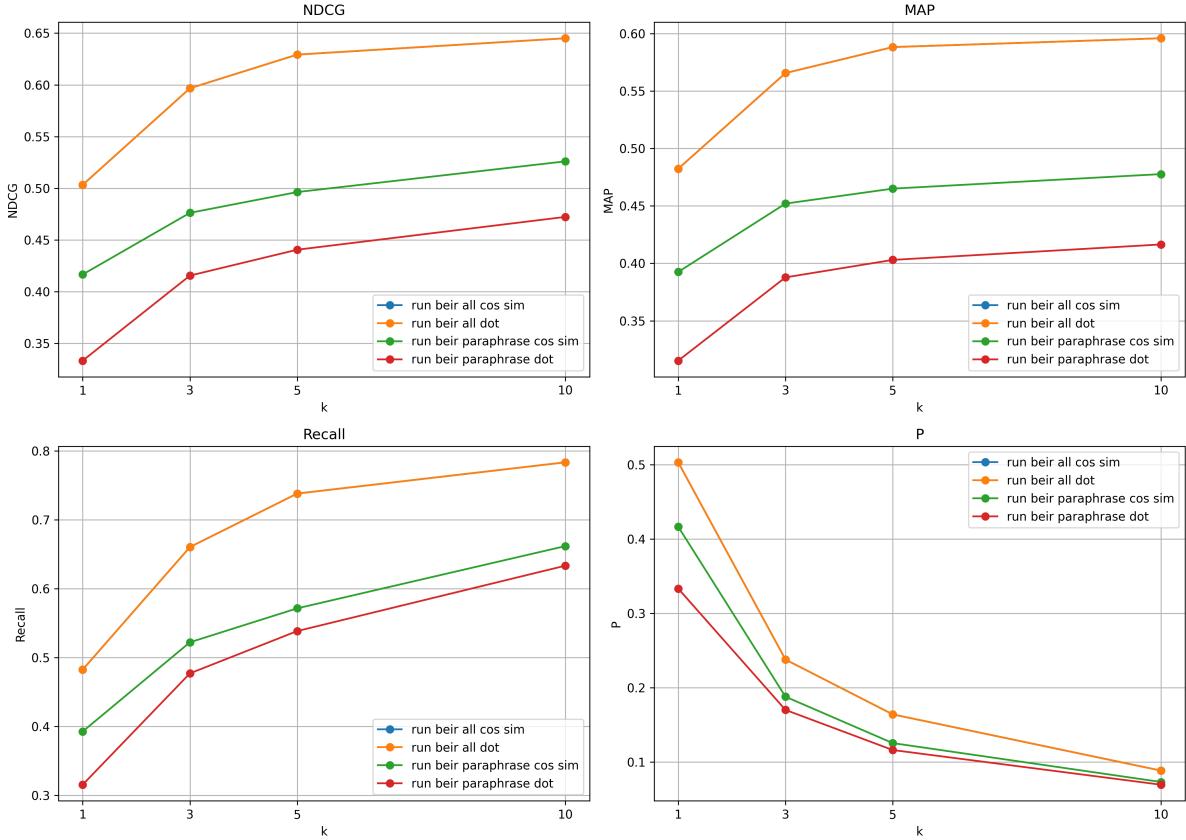


Figura 5.11: Test Score Function in fase di retrieve per la pipeline BEIR

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` 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`.

5.2.3 Dataset

Infine, si può confermare che anche il dataset e le sue caratteristiche influenzano significativamente i risultati. In Figura 5.12 sono presentati i risultati: il dataset con le prestazioni migliori è scifact, che risulta superiore in tutte le metriche eccetto la precision, dove prevale, soprattutto con k grande, scidocs.

Inoltre, oltre al dataset, sono stati utilizzati due modelli di embedding per assicurarsi che il modello di embedding non sia sensibile al dataset scelto, confermando questo fatto: all risulta sempre superiore rispetto al paraphrase.

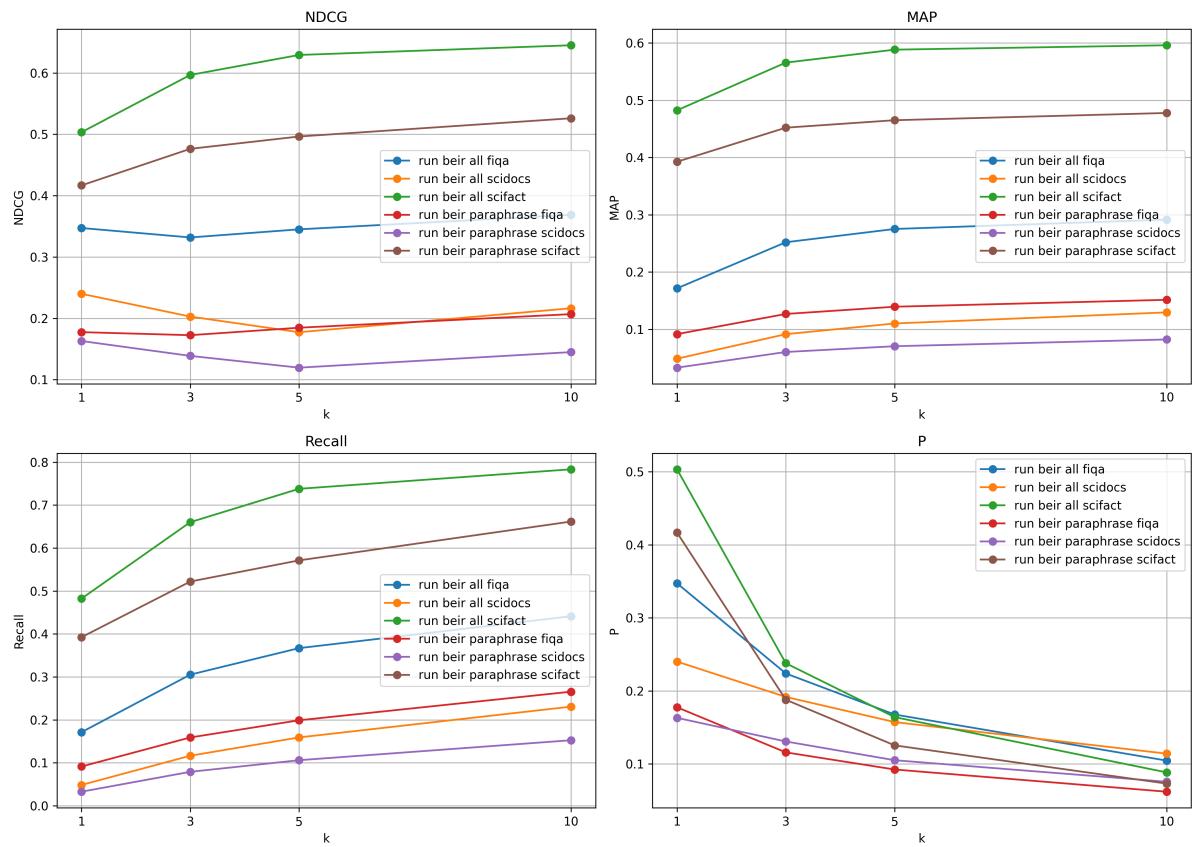


Figura 5.12: Test Datasets in fase di retrieve per la pipeline BEIR

Capitolo 6

Conclusioni e Sviluppi Futuri

6.1 Conclusioni su Sviluppo e Metodologia

Il progetto ha permesso di sviluppare con successo due pipeline complementari per la valutazione di sistemi RAG. La pipeline custom offre controllo granulare sui parametri e analisi approfondite, mentre BEIR fornisce un framework standardizzato per confronti con lo stato dell'arte. L'approccio metodologico di analisi univariata si è rivelato efficace per isolare l'impatto di ogni componente. La strutturazione modulare ha facilitato l'estensibilità e la conduzione di esperimenti sistematici. L'uso di configurazioni JSON ha semplificato la gestione degli esperimenti, garantendo tracciabilità completa. L'implementazione di plotter dedicati ha facilitato l'analisi visiva e il confronto tra configurazioni diverse.

6.2 Conclusioni Pipeline Custom

I risultati evidenziano aspetti significativi delle performance dei sistemi RAG.

Per quanto concerne la pipeline custom:

Modello di embedding: Il miglior modello di embedding è risultato essere `all-MiniLM-L6-v2`, anche se altri modelli hanno offerto prestazioni simili.

Chunk size: Il chunk size più grande testato (2500 token) ha fornito le migliori prestazioni complessive.

Chunk overlap: Il miglior valore di overlap è risultato essere pari a 1/5 della dimensione del chunk size, in quasi tutte le configurazioni provate.

Fase di generazione: Le prestazioni migliorano con temperature relativamente alte. Il parametro max new tokens non sembra avere impatto significativo sulla generazione, mentre la tipologia di prompt utilizzata sì: i template `constrained` e `structured` sono risultati i migliori.

Numero di documenti recuperati: Il giusto compromesso si ottiene con k=3, che massimizza la qualità delle risposte senza introdurre rumore superfluo.

6.3 Conclusioni Pipeline BEIR

Passando alla pipeline BEIR:

Qualità degli embedding: La qualità degli embedding si conferma un fattore cruciale anche in questo contesto. Il modello `bge-base-en-v1.5` risulta il migliore in termini di

prestazioni. Da notare che `all-MiniLM-L6-v2`, che nella pipeline custom era il migliore, qui si posiziona solo al terzo posto.

Score function: La funzione di similarità `cosine similarity` si è dimostrata superiore rispetto al `dot product`, offrendo risultati migliori nelle metriche di retrieval.

Dominio e dimensione del dataset: Il dominio e la dimensione del dataset influenzano significativamente i risultati. Il dataset `SciFact` è quello su cui si ottengono le prestazioni migliori, mentre `Scidocs` e `FiQA` registrano risultati inferiori.

6.4 Limiti del sistema attuale

Il sistema presenta alcune limitazioni che ne circoscrivono l'applicabilità.

L'utilizzo di un dataset ridotto (solo 10 esempi) limita la significatività statistica dei risultati.

La scelta del modello LLM è stata condizionata dai costi delle API, limitando la sperimentazione al solo `SmollM3-3B`.

Le due pipeline sviluppate utilizzano approcci di ricerca diversi, rendendo difficile un confronto diretto ed equo delle prestazioni.

Manca inoltre un monitoraggio dettagliato delle risorse computazionali (CPU, RAM, GPU) durante l'esecuzione degli esperimenti.

6.5 Sviluppi futuri

Per superare questi limiti, proponiamo i seguenti sviluppi futuri:

- Implementazione del supporto GPU per FAISS, al fine di accelerare le operazioni di similarity search.
- Estensione degli esperimenti a dataset completi, per garantire una maggiore significatività statistica.
- Sperimentazione con modelli LLM più avanzati tramite API premium, per valutare l'impatto di modelli di generazione più potenti.
- Integrazione delle due pipeline, così da consentire confronti equi e standardizzati tra i diversi approcci.
- Implementazione di un profiling dettagliato per CPU, RAM e GPU, per monitorare e ottimizzare l'efficienza computazionale del sistema.

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] 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).
- [3] 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. 4).
- [4] Vladimir Karpukhin et al. «Dense Passage Retrieval for Open-Domain Question Answering». In: *EMNLP 2020*. 2020. URL: <https://arxiv.org/abs/2004.04906> (cit. a p. 4).
- [5] 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. 3).
- [6] 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).
- [7] 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).