A.D. 1308
unipg
DIPARTIMENTO
DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA E ROBOTICA

TESI DI LAUREA

Valutazione comparativa di pipeline di Retrieval-Augmented Generation (RAG): analisi di tecniche tradizionali, di GraphRAG e ibride

RELATORE:
Prof. Fabrizio Montecchiani

LAUREANDO:
Tommaso Martinelli

ANNO ACCADEMICO 2023 - 2024

Indice

Capitolo 1

Introduzione

Negli ultimi anni l'intelligenza artificiale generativa ha avuto un notevole sviluppo e una diffusione sempre più ampia. Le applicazioni di questa tecnologia sono potenzialmente infinite e possono coinvolgere qualsiasi campo. Ad oggi l'applicazione in cui l'intelligenza artificiale generativa ha avuto più successo è la generazione del testo, che si basa su modelli in grado di comprendere e generare testo in linguaggio naturale. La diffusione di questi modelli ha rappresentato una vera e propria rivoluzione, andando a modificare radicalmente l'approccio ai principali task di Natural Language Processing (NLP) e portando a risultati sempre più vicini a quelli umani.

Tra le sfide più importanti in questo campo vi sono tutti i problemi legati alla generazione di testo coerente e di qualità e di integrazione di fonti di conoscenza esterne o private. Questo tipo di compiti si affrontano generalmente con approcci di Retrieval-Augmented Generation (RAG), che combinano tecniche di retrieval con modelli generativi. I sistemi RAG sono in grado di migliorare le capacità di modelli di linguaggio, rendendo le loro risposte più coerenti e accurate su ambiti specifici, combinando fasi di retrieval e generazione. Su questo tipo di sistemi è attiva una serie di studi e ricerche, che mirano a migliorare le performance e l'efficienza, oppure che ne vogliono estendere le funzionalità. Il RAG infatti è specifico per task di tipo question-answering, ma stanno nascendo sempre più varianti che provano ad adattare un approccio simile a task differenti.

In questo contesto si inserisce il progetto di tesi, che si propone di esplorare un approccio di retrieval-augmented generation basato sull'utilizzo di grafi di conoscenza. L'idea di utilizzare grafi per migliorare modelli generativi nasce dalla

grande flessibilità di una struttura di questo genere, che permette di rappresentare in modo efficiente e flessibile conoscenza strutturata, oppure di schematizzare relazioni complesse tra entità. La struttura a grafo inoltre rappresenta anche una soluzione generalmente efficiente per la memorizzazione e la ricerca di dati intercorrelati, oltre a essere comprensibile e interpretabile da un essere umano. L'approccio con grafo di conoscenza si propone quindi di migliorare le capacità dei modelli generativi e, in particolare, si pone l'obiettivo di affrontare problemi di query-focused summarization. Questo tipo di task è finalizzato a generare risposte a domande che richiedono una sintesi di informazioni provenienti da fonti eterogenee. Questo task è particolarmente complesso e richiede la capacità di comprendere in modo accurato e completo le informazioni presenti nelle fonti e di sintetizzarle ed elaborarle in modo coerente e comprensibile.

1.1 Obiettivi della tesi

L'obiettivo principale della tesi è quello di esplorare questo nuovo approccio di retrieval-augmented generation basato su grafi di conoscenza, denominato GraphRAG, e di valutarne l'efficacia su task diversi. In particolare si vuole confrontare l'approccio GraphRAG con quello tradizionale, esplorando le possibilità di ottimizzazione di entrambi i sistemi e analizzando in che misura ciascun approccio sia adeguato ad affrontare certi tipi di sfide. L'intento è comprendere le potenzialità e i limiti di GraphRAG, verificando se esso possa rappresentare una alternativa al RAG tradizionale oppure se possa essere affiancato al RAG tradizionale per generare sistemi più complessi e performanti. Il lavoro di tesi è stato sviluppato in collaborazione con l'azienda *E4 Analytics*, specializzata in soluzioni di intelligenza artificiale private e personalizzate. L'azienda ha permesso di lavorare su sistemi ad alte prestazioni e di testare le soluzioni proposte su dataset complessi, mettendo a disposizione risorse di calcolo adeguate.

1.2 Struttura dell'elaborato

Il lavoro di tesi è strutturato nel seguente modo:

- Nel capitolo 2 verranno presentati i principali concetti teorici e le tecnologie alla base del progetto di tesi. Si parlerà in particolare di modelli generativi, di retrieval-augmented generation, di grafi di conoscenza.
- Nel capitolo 3 verrà presentata l'attività sperimentale al centro del lavoro di tesi, fornendo una descrizione dettagliata degli obiettivi e delle metodologie adottate, dei sistemi implementati e delle relative scelte progettuali.
- Nel capitolo 4 verrà presentata l'infrastruttura utilizzata per lo sviluppo e la sperimentazione dei sistemi implementati.
- Nel capitolo 5 verranno presentati i risultati sperimentali ottenuti, valutando le performance dei sistemi implementati e confrontandoli tra loro, proponendo una soluzione finale che possa rappresentare un compromesso tra i due approcci.
- Nel capitolo 6 verranno presentate le conclusioni e le prospettive future del progetto.

Capitolo 2

Stato dell'arte

Il presente capitolo offre un'analisi approfondita della letteratura scientifica e delle tecnologie più rilevanti per la realizzazione di questo progetto di tesi. Gli studi e le tecnologie analizzati sono stati selezionati per fornire una solida base teorica, necessaria per comprendere le scelte progettuali adottate e il contesto delle sperimentazioni condotte. Alcuni degli strumenti e delle metodologie presentati sono stati utilizzati direttamente durante lo sviluppo del progetto, mentre altri, pur non essendo stati impiegati in maniera diretta, sono stati inclusi per la loro rilevanza teorica e il loro contributo al quadro generale delle tecnologie esistenti. Il capitolo è organizzato in sezioni, ciascuna dedicata a un aspetto specifico del contesto tecnologico e scientifico.

2.1 AI Generativa

L'intelligenza artificiale generativa (GenAI o GAI) è una branca dell'intelligenza artificiale che rientra nella categoria dei sistemi di intelligenza artificiale a uso generale (general-purpose AI systems, GPAI) [1]. Questi sistemi sono in grado di offrire una vasta gamma di servizi applicabili in contesti diversificati, come il riconoscimento di immagini e parole, la generazione di audio, video e immagini, e la rilevazione di pattern complessi. La GenAI si basa su reti neurali generative, che attraverso il processo di addestramento apprendono i pattern e la distribuzione dei dati su cui sono stati istruiti. Questo permette ai modelli di creare contenuti nuovi a partire dalle informazioni apprese, come la generazione di immagini, testi o suoni

originali, che non erano presenti nel dataset di addestramento. Grazie alla capacità di imitare lo stile e la struttura dei dati di addestramento, questi modelli possono produrre contenuti che mantengono una coerenza con le caratteristiche dei dati originari, pur essendo unici e originali. Una delle principali caratteristiche che rendono l'AI generativa particolarmente potente è la capacità di comprendere e riprodurre la complessità dei dati, permettendo applicazioni che spaziano dal completamento di testi alla creazione di immagini artistiche, fino alla generazione di risposte simulate in contesti conversazionali. Altri importanti vantaggi dell'AI generativa sono la velocità e la scalabilità nella creazione di contenuti, il costo relativamente ridotto di accesso, e la possibilità di personalizzare i modelli per adattarli a vari scenari e contesti. Nonostante le sue potenzialità, la GenAI presenta anche alcune criticità, come evidenziato da Bommasani et al. (2021) nel loro studio "On the Opportunities and Risks of Foundation Models" [2]. Tra le sfide principali si annoverano la difficoltà nel comprendere come i modelli generano i contenuti, il rischio di bias e fenomeni di allucinazione [3], e la necessità di garantire la sicurezza e la privacy dei dati impiegati.

2.2 LLM

2.2.1 Caratteristiche e Definizione dei LLM

I Large Language Models (LLM) sono tra i modelli di intelligenza artificiale generativa più noti e utilizzati. Si tratta di una classe di modelli di apprendimento automatico basati su reti neurali profonde, addestrati su ampie collezioni di testi per apprendere la struttura, la distribuzione delle parole e dei concetti. L'obiettivo principale di questi modelli è generare testi coerenti e comprensibili, riproducendo lo stile e il contenuto dei dati di addestramento e rispondendo a una vasta gamma di richieste linguistiche.

Il termine "large" fa riferimento alla dimensione di questi modelli, che è determinata dal numero di parametri di cui sono composti. I LLM possono contare da milioni a centinaia di miliardi di parametri, che rappresentano i "pesi" appresi durante la fase di addestramento. L'elevato numero di parametri permette ai modelli di apprendere una vasta gamma di informazioni e di comprendere relazioni complesse all'interno del linguaggio naturale.

Grazie a queste caratteristiche, i LLM sono estremamente versatili e possono essere impiegati in una vasta gamma di applicazioni nell'ambito del Natural Language Processing (NLP), come la traduzione automatica, il riassunto di testi, la risposta a domande e la generazione di contenuti creativi.

2.2.2 Contesto Storico e Sviluppo Tecnologico

Lo sviluppo di LLM è stato reso possibile dai progressi nella progettazione di reti neurali profonde, in particolare dall'avvento dei Transformer. Questi rappresentano una delle architetture più avanzate per la modellazione del linguaggio naturale e sono stati introdotti per la prima volta da Vaswani et al. nel 2017 nel loro articolo "Attention Is All You Need" [4]. La loro introduzione ha segnato un momento di svolta nel campo dell'elaborazione del linguaggio naturale (NLP), superando le limitazioni delle reti neurali ricorrenti (Recurrent Neural Networks, RNN) [5] e delle reti di memoria a lungo termine (Long Short-Term Memory, LSTM) [6].

Prima dell'introduzione di queste nuove architetture, i modelli di NLP presentavano numerose limitazioni che ne ostacolavano la scalabilità ed efficacia in applicazioni complesse. La principale problematica delle RNN era l'incapacità di catturare le dipendenze a lungo raggio all'interno del testo, processando parola per parola e aggiornando lo stato interno per ogni input. Questo approccio si traduce in una progressiva perdita di informazioni, man mano che il testo diventa più lungo, per via del fenomeno noto come "vanishing gradient problem" [7], che limita la capacità di catturare relazioni che si estendono su lunghe distanze. Le LSTM sono state introdotte per superare questo problema, introducendo una struttura di memoria. Tuttavia, anche le LSTM presentano limitazioni importanti, tra cui il processamento sequenziale e la complessità computazionale. Le LSTM, pur migliorando la capacità di memorizzazione delle informazioni rispetto alle RNN, presentano comunque limiti significativi nella gestione di dataset di grandi dimensioni e nel catturare relazioni complesse all'interno del testo. A causa di queste limitazioni, le RNN e le LSTM non riuscivano a soddisfare le esigenze dei modelli di linguaggio su larga scala, come richiesto da applicazioni di machine translation, text generation e question answering. L'introduzione dei Transformer ha permesso di superare queste barriere, offrendo un'architettura più efficiente e scalabile per la gestione dei dati sequenziali.

I Transformer si basano su un'innovativa architettura che sfrutta un meccanismo di self-attention, che consente ai modelli di prestare attenzione a tutte le parole di una sequenza contemporaneamente. Questo permette di catturare le relazioni tra parole indipendentemente dalla loro distanza all'interno della sequenza, risolvendo il problema della perdita di informazioni su lunghe distanze. Inoltre, i Transformer sono altamente parallelizzabili, rendendo possibile l'addestramento su dataset di grandi dimensioni in tempi ragionevoli.

Lo sviluppo dei Transformer, insieme all'aumento della potenza computazionale, alla disponibilità di dataset di grandi dimensioni e l'efficacia della distribuzione del carico su hardware specializzato, ha reso possibile la creazione di LLM su scala senza precedenti. Grazie a queste innovazioni, i Transformer hanno rappresentato la base per lo sviluppo dei Large Language Models (LLM) moderni, consentendo loro di gestire e comprendere testi complessi su scala globale e aprendo la strada a nuove frontiere nell'ambito dell'intelligenza artificiale generativa.

2.2.3 Architettura e Funzionamento dei LLM

L'architettura dei LLM è strettamente legata ai Transformer, che ne costituiscono la base teorica e tecnologica. La struttura di un LLM è infatti detta "transformer-based" e viene fatto uso di un insieme di moduli e meccanismi per l'analisi, la comprensione e la generazione di testi. Di seguito vengono descritti i principali elementi architetturali e il loro ruolo nel funzionamento dei LLM. In Figura **??** è illustrata una rappresentazione schematica dell'architettura di un LLM.

Meccanismo di Attention

Il meccanismo di attention è una componente fondamentale dei Large Language Models (LLM) basati su architetture Transformer. Questo operatore consente al modello di selezionare le parti più rilevanti dell'input per calcolare l'output, migliorando così la sua capacità di elaborazione del linguaggio naturale. Il meccanismo di attention si sviluppa in tre step principali, che permettono di prestare attenzione solo alle parti più rilevanti, riuscendo ad ignorare quelle meno importanti.

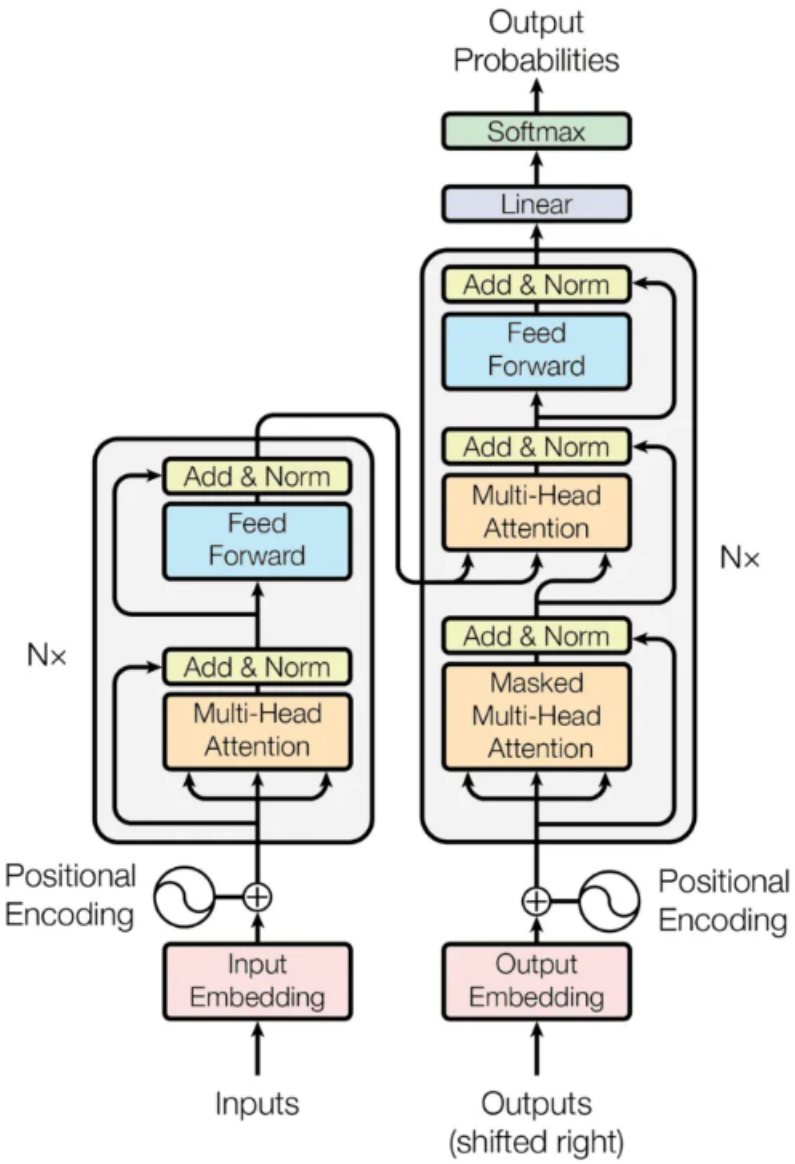

Figura 2.1: Schema architetturale di un LLM [8]

1. **Calcolo dei punteggi di attenzione:** in questa fase, il modello calcola i punteggi di attenzione per ciascuna parola dell'input, determinando la rilevanza di ciascuna parola rispetto alle altre. Questo calcolo si basa su una funzione di similarità tra le parole, che consente al modello di valutare la correlazione tra le diverse parti dell'input. I punteggi vengono calcolati confrontando un vettore di query (Q) con i vettori di chiave (K) delle altre parole nella sequenza.

$$\text{Attention}(Q_i, K_j) = Q_i \cdot K_j^T$$

2. **Normalizzazione dei punteggi:** i punteggi di attenzione vengono normalizzati attraverso una funzione softmax, che trasforma i punteggi in una distribuzione di probabilità, che indica quanto ciascuna parola contribuisce al calcolo dell'output.

$$\alpha_{ij} = \frac{\exp(Q_i \cdot K_j^T)}{\sum_k \exp(Q_i \cdot K_k^T)}$$

3. **Calcolo dell'output:** infine, il modello calcola l'output pesando i vettori di valore (V) delle parole dell'input in base ai punteggi di attenzione normalizzati. Questo passaggio consente al modello di concentrarsi sulle parti più significative nel contesto.

$$\text{Output}_i = \sum_j \alpha_{ij} V_j$$

Tra le varie funzioni che possono essere impiegate per calcolare i punteggi di attenzione, il prodotto scalare è particolarmente efficace. Quando si utilizza il prodotto scalare tra il vettore di query e quello di key, si parla di *self-attention*. Il *self-attention* permette a ogni parola di una sequenza di considerare tutte le altre parole nella stessa sequenza, pesandole in base alla loro rilevanza. Questo approccio consente al modello di catturare le relazioni semantiche tra parole, anche se distanti, rendendo più ricca la comprensione del contesto.

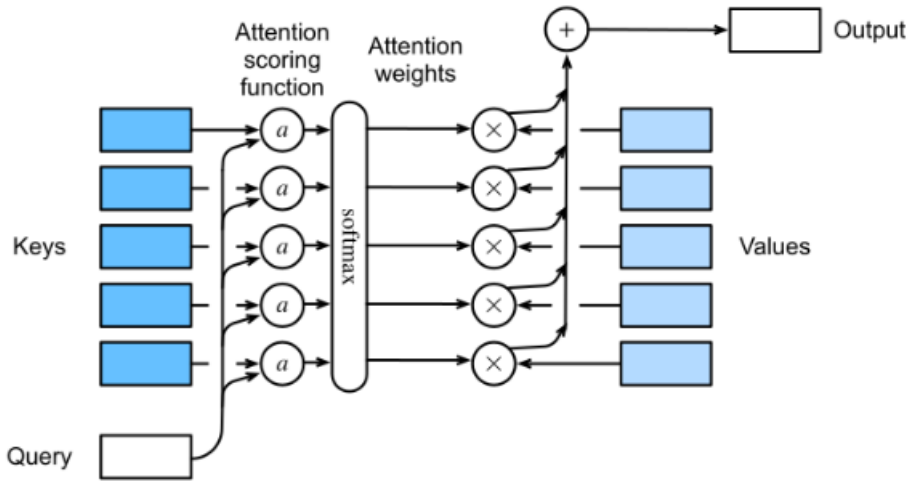

Figura 2.2: Processo di calcolo dell'Attention score [9]

L'uso del prodotto scalare è preferito per la sua semplicità e efficienza computazionale, che facilita l'implementazione e la parallelizzazione del calcolo durante l'addestramento. Inoltre, il *self-attention* rende il modello più adattabile a diverse lunghezze di input, migliorando la capacità di gestione di sequenze variabili e la scalabilità su dataset di grandi dimensioni. Esistono altre varianti del meccanismo di attention, che utilizzano operazioni differenti per calcolare i punteggi di attenzione, ma sono meno comuni rispetto al prodotto scalare.

Multi-Head Attention

Il meccanismo di Multi-Head Attention è una variante del meccanismo di attention che consente al modello di prestare attenzione a diverse parti dell'input contemporaneamente. Rappresenta un'estensione del meccanismo di base, in cui il modello calcola l'attention con più insiemi di pesi, noti come "heads", e combina i risultati per ottenere un output più ricco e dettagliato. Per ogni head, il modello apprende un set di pesi indipendenti, che vengono utilizzati per calcolare i punteggi di attenzione e produrre un output parziale, consentendo al modello di concentrarsi su diversi pattern e relazioni all'interno dell'input.

Dal punto di vista matematico, una attention-head può essere rappresentata come segue:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Dove W_i^Q, W_i^K e W_i^V sono i pesi appresi per la head i, che vengono applicati ai vettori di query, chiave e valore rispettivamente.

Al termine del processo, i risultati di tutti gli head vengono concatenati e combinati attraverso una trasformazione lineare, creando un output finale che incorpora le informazioni da tutte le heads. Lo score di attention finale è calcolato come segue:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O$$

Dove W^O è un ulteriore set di pesi appresi che viene applicato all'output concatenato per produrre l'output finale.

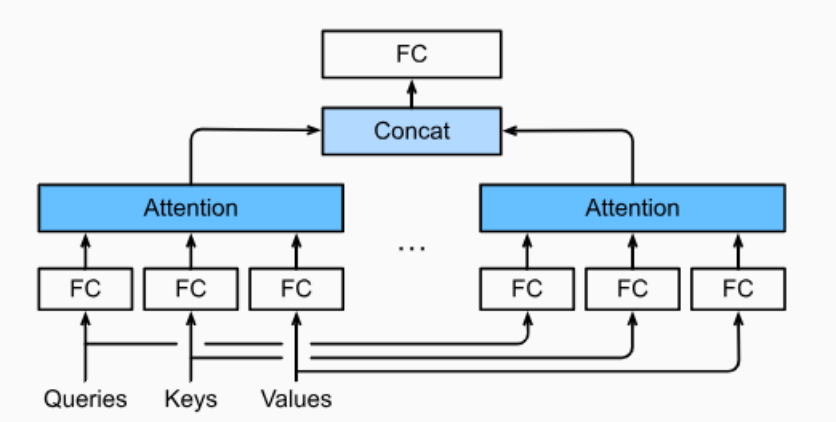

Figura 2.3: Multi-head attention [10]

Oltre al vantaggio di poter considerare più aspetti dell'input contemporaneamente e catturare relazioni complesse, si ha anche un potenziale vantaggio prestazionale, potendo parallelizzare il calcolo delle attention-heads. In Figura 2.3 è illustrata una rappresentazione del processo.

Feedforward Network (FFN)

Il Feedforward Network (FFN) è un componente chiave dell'architettura dei Large Language Models (LLM) basati su Transformer. Questa componente essenzialmente è una rete neurale feedforward, che viene applicata dopo il meccanismo di attention per elaborare ulteriormente l'output, applicando una serie di trasformazioni non lineari che vanno ad arricchire la rappresentazione dell'input. La struttura

di un FNN è semplice, con ogni layer composto da una trasformazione lineare seguita da una funzione di attivazione non lineare, come la ReLU [11] (Rectified Linear Unit) che viene comunemente utilizzata . La funzione può essere espressa matematicamente come segue:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

Dove W_1 e W_2 sono i pesi appresi, b_1 e b_2 sono i bias e x è l'input proveniente dal meccanismo di self-attention. L'aggiunta del FFN consente al modello di apprendere relazioni più complesse e di catturare pattern non lineari all'interno dell'input. Questo è particolarmente importante nel contesto dell'elaborazione del linguaggio naturale [12], dove le relazioni tra le parole possono essere complesse e sfumate.

Positional Encoding

Il Positional Encoding è un componente fondamentale dei Large Language Models (LLM), progettato per fornire al modello informazioni sulla posizione delle parole all'interno della sequenza. Poiché i Transformer non considerano l'ordine delle parole all'interno dell'input, non possedendo una struttura ricorrente o convoluzionale, è necessario fornire al modello un modo per comprendere la posizione delle parole all'interno dell'input. Il Positional Encoding si basa sull'idea di aggiungere vettori di posizione agli embedding delle parole. Questi vettori sono generati utilizzando funzioni sinusoidali, più precisamente la formula per il calcolo del Positional Encoding è la seguente:

$$\text{PE}(pos, 2i) = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$\text{PE}(pos, 2i+1) = \cos(pos/10000^{2i/d_{\text{model}}})$$

Dove pos è la posizione della parola all'interno della sequenza, i è la dimensione del vettore di posizione, e d_{model} è la dimensione del modello.

L'aggiunta del Positional Encoding consente al modello di distinguere la posizione delle parole all'interno dell'input, migliorando la capacità di catturare relazioni spaziali e temporali all'interno della sequenza e contribuendo al miglioramento delle prestazioni del modello. Tuttavia, studi recenti hanno messo in discussione l'efficacia delle tecniche tradizionali di Positional Encoding, evidenziando che approcci alternativi, come quelli senza Positional Encoding esplicito, possono risultare più efficaci in alcuni contesti di generalizzazione [13]

Layer Normalization

Il Layer Normalization [14] è una tecnica di regolarizzazione utilizzata nei Large Language Models (LLM) basati su Transformer per migliorare la stabilità e l'efficienza dell'addestramento. Questa tecnica consiste nell'applicare una normalizzazione ai valori di output di ciascun layer del modello, riducendo la varianza e facilitando il flusso di informazioni all'interno della rete. Il Layer Normalization è particolarmente utile nei modelli di NLP, dove la complessità delle relazioni tra le parole e la lunghezza delle sequenze possono rendere l'addestramento più difficile. La formula matematica per il calcolo del Layer Normalization è la seguente:

$$\mathrm{LN}(x) = \gamma \frac{x - \mu}{\sigma} + \beta$$

Dove x è l'input al layer, μ e σ sono la media e la deviazione standard degli elementi di x, e γ e β sono i parametri appresi dal modello, che permettono di scalare e traslare l'output normalizzato.

Nell'ambito dei LLM, la Layer Normalization viene comunemente utilizzata sia dopo il meccanismo di attention che dopo il Feedforward Network, per mantenere le distribuzioni delle attivazioni coerenti attraverso i vari layer del modello. Questo approccio consente di ridurre l'effetto di covariate shift, migliorando la convergenza dell'addestramento e permettendo al modello di apprendere più velocemente e in modo più robusto, essendo meno sensibile a variazioni nei dati di input.

Residual Connection

Le residual Connection, o connessioni residue, sono un'altra tecnica di regolarizzazione utilizzata nei Large Language Models (LLM) basati su Transformer. Il loro scopo è quello di affrontare della degradazione delle performance che può verificarsi nei modelli di deep learning a causa della presenza di layer profondi. Le connessioni residue permettono di bypassare uno o più layer del modello, consentendo al segnale di propagarsi direttamente attraverso la rete senza subire trasformazioni, permettendo al gradiente di fluire più facilmente attraverso la rete e riducendo il rischio di vanishing gradient. ome evidenziato da He et al. (2016), le residual connections facilitano l'addestramento di reti neurali profonde migliorando il flusso del gradiente [15]. La formula matematica per il calcolo delle connessioni residue è la

seguente:

$$\text{Residual}(x) = x + \text{Sublayer}(x)$$

Dove x è l'input al layer e Sublayer(x) rappresenta l'output del layer, che viene aggiunto all'input per creare l'output finale. Le connessioni residue vengono tipicamente applicate dopo ciascun layer del modello. Secondo Xu et al. (2016), le residual connections permettono una propagazione del gradiente più efficace, simile a quella di un ensemble di reti più superficiali [16]. L'utilizzo di questa tecnica consente di mantenere informazioni importanti all'interno del modello, che altrimenti potrebbero essere perse durante la fase di addestramento, migliorando la capacità di generalizzazione.

Varianti architetturali dei LLM

L'architettura originale di un Transformer è di tipo "encoder-decoder", composta da due parti principali: un encoder, che si occupa di codificare la sequenza di input in una rappresentazione vettoriale compatta, e un decoder, responsabile della decodifica della rappresentazione in un output comprensibile. L'Encoder è composto da una pila di layer, o blocchi, ciascuno dei quali contiene un meccanismo di self-attention e un Feedforward Network, con una connessione residua tra i due, seguita da una normalizzazione. Il Decoder, invece, è simile all'Encoder, ma include un meccanismo di attention multi-head sull'output dell'Encoder e va a modificare il meccanismo di attention per evitare che una parola possa "guardare" le parole successive, evitando così di violare la causalità.

La struttura di un LLM, basato su Transformer, quindi si basa su un'architettura modulare. Esistono numerose varianti architetturali, che si differenziano per la disposizione e la composizione dei vari moduli, allo scopo di affrontare specifiche sfide e problemi. Le configurazioni principali verranno descritte di seguito, presentando le differenze a livello architetturale e fornendo esempi di modelli e specifiche applicazioni.

Encoder-Decoder La variante più comune di un LLM, come già accennato, è quella "encoder-decoder", che si basa sull'architettura originale dei Transformer. Questa architettura è particolarmente adatta per compiti di "Sequence-to-Sequence" (Seq2Seq), come la traduzione automatica, il riassunto di testi e la generazione di testi, task in cui è necessario codificare una sequenza di input in una rappresentazione vettoriale e decodificarla in un output comprensibile. L'esempio più noto di un modello basato su questa architettura è il Transformer di Vaswani [4], che ha introdotto per la prima volta l'architettura Transformer.

Encoder-Only I modelli di tipo "encoder-only" sono una variante semplificata dei LLM, che si basano solo sull'encoder dei Transformer, senza includere il decoder. Questo tipo di architettura è focalizzata esclusivamente sull'analisi e la codifica delle sequenze di input, senza la necessità di generare un output. I modelli di tipo "encoder-only" sono particolarmente adatti per compiti di classificazione di testi, analisi del sentiment, named entity recognition (NER) e altre applicazioni in cui è necessario codificare una sequenza. In modelli di questo genere viene spesso applicata la bidirectional attenction [17], che permette di considerare il contesto sia precedente che successivo ad una parola, migliorando le performance in task di classificazione o riconoscimento di entità.

Questo tipo di architettura è ampiamente utilizzato e i modelli più noti basati su questa configurazione sono BERT (Bidirectional Encoder Representations from Transformers) [18] e le sue varianti, come RoBERTa [19], DistilBERT [20] e ALBERT [21].

Decoder-Only I modelli di tipo "decoder-only" rappresentano una configurazione semplificata dei Transformer, che si basa solo sul decoder, senza includere l'encoder. Questa architettura è particolarmente adatta per compiti di generazione di testi e completamento automatico, in cui è necessario decodificare una rappresentazione vettoriale in un output comprensibile. L'unica componente necessaria è appunto il decoder e il focus è sulla generazione dell'output a partire da una sequenza di input o da un prompt. Il meccanismo di attention utilizzato in modelli di questo tipo è noto come "causal attention", che assicura che il modello consideri solo i token precedenti nella sequenza durante la generazione dell'output, evitando di violare la causalità.

Molti modelli hanno adottato questa configurazione, tra cui GPT (Generative Pre-trained Transformer). I modelli GPT, come GPT-4 [22] e i suoi predecessori, sono basati su un'architettura decoder-only, perchè hanno come obiettivo principale la generazione di testi e la risposta a domande. Questo tipo di modelli sono pre-addestrati su grandi dataset, allo scopo di rendere il modello più generale e adattabile a diversi compiti di generazione di testi.

2.2.4 Addestramento di LLM

L'addestramento dei Large Language Models (LLM) è un processo complesso che richiede una combinazione di metodologie avanzate di ottimizzazione, gestione dei dati e infrastrutture computazionali. L'obiettivo dell'addestramento è fare in modo che il modello apprenda una rappresentazione del linguaggio naturale che gli consenta di generare risposte coerenti e pertinenti in vari contesti. Il processo di addestramento si può vedere come suddiviso in due fasi principali: il pre-addestramento e il fine-tuning.

Pre-addestramento

Il pre-training è la fase iniziale in cui il modello viene addestrato su un ampio dataset non etichettato. Durante questa fase, il modello apprende rappresentazioni generali del linguaggio attraverso tecniche di apprendimento auto-supervisionato. Due delle tecniche più comuni utilizzate per il pre-addestramento sono il "Masked Language Modeling" (MLM), in cui il modello deve prevedere le parole mancanti in una sequenza, e il "Next Sentence Prediction" (NSP), in cui il modello deve prevedere se due frasi sono consecutive o meno, entrambi i task vengono spiegati approfonditamente da Brown et al. (2020) [23]. Il pre-addestramento è un processo intensivo dal punto di vista computazionale, che richiede l'uso di hardware specializzato, come GPU o TPU, e l'ottimizzazione di algoritmi di apprendimento profondo per gestire dataset di grandi dimensioni. Questa fase è fondamentale per ottenere un LLM performante, determinandone in larga parte le capacità generali.

Fine-tuning

Il fine-tuning è la fase successiva al pre-training, in cui il modello già addestrato su un vasto corpus di dati viene ulteriormente raffinato su un dataset più specifico e solitamente più piccolo. Durante questa fase, il modello apprende a gestire compiti specifici, come la classificazione del testo, l'analisi del sentiment, o la risposta a domande, migliorando così le sue performance in contesti pratici [24]. Durante il fine-tuning, il modello sfrutta le rappresentazioni già apprese durante il pre-training, adattandole alle nuove informazioni presenti nel dataset di destinazione. Questo approccio consente di risparmiare tempo e risorse computazionali rispetto all'addestramento di un modello da zero. Questa fase non sempre è necessaria e in certi casi il modello pre-addestrato può essere utilizzato direttamente per compiti specifici, senza ulteriori raffinamenti.

2.2.5 Inferenza di LLM

L'inferenza di un Large Language Model (LLM) è il processo mediante il quale un modello pre-addestrato elabora un input e genera un output. Questo processo avviene in vari passaggi, che permettono di trasformare una sequenza di testo in una risposta contestualizzata e adeguata. Gli step fanno uso degli elementi architetturali descritti nelle sezioni precedenti 2.2.3, di seguito vengono descritti i passaggi nel dettaglio.

1. **Input dell'utente:** il processo di inferenza inizia con l'input dell'utente, che può variare di natura e complessità. Prima di essere inviato al modello, il testo può subire delle operazioni di preprocessing, ma ciò dipende dalla specifica tipologia architetturale e dal tipo di input. Il testo viene quindi inviato al modello per il passo successivo.

2. **Tokenizzazione:** è il processo di suddivisione del testo in token, ovvero le unità di base che compongono il testo. Il tipo di tokenizzazione dipende dal modello utilizzato, ma ne esistono varie tipologie, come la *Word-based tokenization*, che suddivide il testo in parole, la *Subword-based tokenization*, che utilizza segmenti di parole, e la *Character-based tokenization*, che considera i singoli caratteri. La tokenizzazione è un passaggio fondamentale per la corretta elaborazione del testo da parte del modello, deve essere in grado di gestire

correttamente spazi e punteggiatura, e i token risultanti devono essere mappati in un vocabolario predefinito. Ogni token viene quindi convertito in un vettore numerico, che rappresenta la sua posizione all'interno del vocabolario.

3. **Embedding:** una volta ottenuti i token, il modello li converte in vettori numerici in uno spazio semantico denso, per catturare le relazioni semantiche tra le parole. Il modello contiene una matrice di embedding pre-addestrata, dove ogni riga rappresenta un vettore numerico corrispondente a un token del vocabolario. Se il vocabolario ha dimensione V e gli embedding hanno dimensione d, la matrice di embedding avrà dimensione $V \times d$. Durante l'inferenza, i token generati dalla tokenizzazione vengono usati per effettuare una lookup nella matrice di embedding, ottenendo così i vettori corrispondenti.

4. **Creazione delle matrici K, Q, V:** una volta ottenuti gli embedding dei token, il modello crea le matrici di chiavi, query e valori, tramite moltiplicazioni matriciali con pesi appresi. Ogni embedding passa attraverso tre matrici di pesi per generare K, Q e V. Queste tre matrici simulano il modo in cui la mente umana elabora le informazioni, permettendo al modello di prestare attenzione a diverse parti dell'input in base alla rilevanza.

5. **Self-Attention:** il modello applica il meccanismo di self-attention alle matrici di K, Q e V, calcolando i punteggi di attenzione e combinando i valori in base a questi punteggi. Il funzionamento di questo meccanismo è stato spiegato nel dettaglio nella sezione apposita, nel capitolo 2.2.3.

6. **Aggregazione dei risultati:** una volta calcolati i punteggi di attenzione, il modello combina i valori pesati per ottenere un output aggregato, che rappresenta l'informazione estratta dall'input. Questo passaggio consente al modello di catturare le relazioni semantiche tra le parole e di generare una rappresentazione contestualizzata dell'input.

7. **Feedforward Neural Network:** i risultati della fase precedente vengono passati attraverso una rete neurale feed-forward, applicando trasformazioni non lineari, per catturare relazioni più complesse.

8. **Norma e residuo:** i risultati sono combinati con l'input originale tramite connessioni residue, seguite dalla normalizzazione del layer. Questo permette di preservare informazioni cruciali, stabilizzando i gradienti.

9. **Stacking dei blocchi di Transformer:** i passaggi precedenti vengono ripetuti per un numero di blocchi di Transformer, che costituiscono la struttura del modello. Ogni blocco è composto da un meccanismo di self-attention, un feedforward network e connessioni residue.

10. **Output layer e decodifica:** l'output dell'ultimo blocco di Transformer viene passato attraverso un layer di output, che converte i vettori numerici in una sequenza di token. Questa operazione è una proiezione lineare, con formula matematica:

$$\text{Logits} = hW + b$$

Dove h è l'output dell'ultimo blocco di Transformer, W e b sono i pesi e il bias del layer di output, e Logits rappresenta i punteggi di probabilità associati a ciascun token del vocabolario. A questo punto i logit vengono trasformati in probabilità attraverso una funzione softmax, che restituisce la distribuzione di probabilità sui token del vocabolario.

$$\text{Probabilità}(w)_i = \frac{\exp(Logits_i)}{\sum_j \exp(Logits_j)}$$

Con w che rappresenta il token e i l'indice del token nel vocabolario.

11. **Generazione della risposta:** infine, il modello genera la risposta in base alla distribuzione di probabilità. Questo processo può variare a seconda del task e del modello, ma comunemente si utilizza una strategia di decoding, che può essere di tipo greedy, beam search o sampling. Il decoding greedy seleziona il token con la probabilità più alta in ogni passaggio.

$$token_{t+1} = \text{argmax}(\text{Probabilità})$$

Il beam search considera più token in ogni passaggio, mantenendo una lista di ipotesi e selezionando le più probabili.

$$beam = \text{top-k}(\text{Probabilità cumulativa})$$

Il sampling seleziona un token in base alla distribuzione di probabilità, introducendo una componente stocastica.

$$token_{t+1} \sim \text{Probabilità}$$

Il processo di generazione della risposta continua fino a quando non viene generato un token di fine sequenza o si raggiunge una lunghezza massima prefissata.

2.2.6 LLM on-cloud vs on-premise

I Large Language Models (LLM) richiedono risorse computazionali significative per l'addestramento e l'inferenza, che possono variare in base alla dimensione del modello, alla complessità del task e alla quantità di dati utilizzati. L'utilizzo di LLM può essere suddiviso in due categorie principali: on-cloud e on-premise, che si differenziano per la modalità di distribuzione delle risorse computazionali. La scelta tra queste due opzioni dipende da vari fattori, tra cui la disponibilità di risorse, i costi, la scalabilità, la sicurezza e le competenze tecniche.

LLM on-cloud

Con questa tipologia ci si riferisce a modelli linguistici di grandi dimensioni, che sono resi disponibili tramite un fornitore di servizi esterni o un cloud provider. L'uso di un modello del genere prevede l'invio di richieste a servizi che espongono API per l'interazione con il modello, oppure l'utilizzo di servizi di cloud computing come AWS, Google Cloud o Azure, che offrono risorse dedicate per l'esecuzione di modelli. Un approccio del genere è particolarmente adatto per applicazioni che richiedono una scalabilità rapida e flessibile, in cui è necessario adattare le risorse in base alla domanda. I modelli on-cloud inoltre vengono generalmente aggiornati e manutenuti dal fornitore di servizi, garantendo un'infrastruttura affidabile e sicura. Questa soluzione però può comportare costi elevati, soprattutto in caso di utilizzo intensivo e continuativo del modello, e può sollevare preoccupazioni sulla privacy e la sicurezza dei dati, inoltre limita la personalizzazione e la flessibilità del sistema che si lega alle API e ai servizi offerti dal fornitore.

LLM on-premise

Con questa tipologia ci si riferisce a modelli linguistici di grandi dimensioni che vengono eseguiti su infrastrutture locali, all'interno di un'organizzazione o di un'azienda. Questa opzione richiede la disponibilità di risorse computazionali dedicate e quindi un investimento iniziale significativo, ma può offrire dei vantaggi significativi. L'utilizzo di modelli on-premise consente un controllo totale sui dati, che rimangono all'interno dell'organizzazione, garantendo la privacy e la sicurezza delle informazioni. Un vantaggio importante è la flessibilità e la personalizzazione del sistema, che può essere adattato alle esigenze specifiche dell'organizzazione, senza dipendere da fornitori esterni. Dal punto di vista dei costi, l'opzione on-premise può risultare molto onerosa in una prima fase, ma nel lungo termine può essere più conveniente rispetto all'utilizzo di servizi cloud, soprattutto in caso di utilizzo intensivo e continuativo del modello. Questo approccio è usato da aziende che danno grande importanza alla privacy dei propri dati ed è particolarmente adatto in contesti in cui si ha grande esperienza e competenze tecniche per la gestione di infrastrutture complesse. La limitazione principale è però ovviamente la richiesta di risorse e competenze apposite, che comportano un investimento iniziale significativo, inoltre la scalabilità può essere limitata e la manutenzione può risultare complessa.

Casi d'uso ibridi

Esistono anche soluzioni ibride, che combinano l'utilizzo di modelli on-cloud e on-premise, per sfruttare i vantaggi di entrambe le opzioni. Un approccio ibrido può prevedere l'utilizzo di modelli on-cloud per task specifici o per picchi di carico, mentre si utilizzano modelli on-premise per task critici o per dati sensibili. Questa soluzione consente di bilanciare i costi, la flessibilità e la sicurezza, adattando le risorse in base alle esigenze specifiche dell'organizzazione.

2.2.7 Limitazioni dei LLM

I Large Language Models (LLM) nonostante abbiano dimostrato una straordinaria capacità di generare testi coerenti e pertinenti, presentano alcune limitazioni intrinseche che ne riducono l'efficacia in determinati contesti.

Mancanza di accesso a informazioni aggiornate

La limitazione principale consiste nell'essere vincolati ai dati di addestramento e non non avere accesso a informazioni aggiornate o a fonti esterne. Dopo la fase di addestramento, il modello dunque non è in grado di aggiornarsi autonomamente e questo può comportare un grave problema di obsolescenza di informazioni. Come evidenziato da Petroni et al. (2019) [25], i LLM non dispongono di una "memoria esplicita" che consenta loro di aggiornarsi con nuove informazioni.

Problemi di allucinazione

Un'altra criticità rilevante è il fenomeno delle cosiddette "allucinazioni". Gli LLM, in mancanza di informazioni specifiche o in presenza di input ambigui, possono generare risposte apparentemente plausibili, ma errate o fuorvianti. Questo avviene per la mancanza di un meccanismo che valuti la veridicità delle informazioni generate. Le allucinazioni possono avere conseguenze gravi in contesti dove l'accuratezza è fondamentale, dove risposte errate potrebbero compromettere decisioni critiche.

Mancanza di specificità

Nonostante la capacità di generalizzare su una serie di task, gli LLM possono soffrire di mancanza di specificità in determinati contesti. Questo può comportare risposte troppo generiche o poco dettagliate, a domande molto specifiche, precise o tecniche, che richiedono una conoscenze approfondite di un settore specifico. Questo limite è dovuto al fatto che gli LLM sono addestrati su una vasta gamma di dati e tendono dunque a produrre risposte che riflettono pattern linguistici generali. Lewis et al. (2020) [26] hanno evidenziato che senza meccanismi specifici per l'accesso a informazioni esterne, gli LLM possono avere difficoltà a generare risposte precise e dettagliate.

Bias e mancanza di trasparenza

Un'altra limitazione significativa dei LLM è la presenza di bias nei dati di addestramento, che possono influenzare le risposte generate dal modello. I LLM potrebbero produrre risposte che contengono stereotipi, pregiudizi o discriminazioni presenti nei dati di addestramento, senza la capacità di riconoscerli o correggerli. Questo

problema è particolarmente critico, considerando anche che non è facile capire il processo di generazione delle risposte, a causa della complessità e dell'opacità dei modelli.

Scalabilità e costi

Gli LLM sono anche molto dispendiosi in termini di risorse computazionali, sia per l'addestramento che per l'inferenza. Richiedono hardware specializzato e una enorme quantità di energia per essere eseguiti, introducendo anche problematiche di sostenibilità ambientale.

Limitata capacità di memoria a lungo termine

I Large Language Models sono in grado di processare sequenze di testo di lunghezza limitata e la loro memoria a lungo termine è spesso insufficiente. Questo comporta che gli LLM tendono a dimenticare informazioni che si trovano nei passaggi precedenti di una conversazione o all'interno di lunghe sequenze di testo. Questo limite può compromettere la coerenza e la coesione delle risposte generate dal modello, in particolare in contesti in cui è richiesta una comprensione a lungo termine del contesto.

2.3 Retrieval-Augmented Generation

Il Retrieval-Augmented Generation (RAG) è una tecnica innovativa che combina i vantaggi dei modelli di generazione del linguaggio con quelli dei modelli di retrieval, in grado di interrogare una base di conoscenza esterna. Viene introdotto da Lewis et al. (2020) [26], come un approccio ibrido che permette di ottenere risposte più accurate, dettagliate e aggiornate, grazie all'accesso a fonti di informazioni esterne. L'idea alla base è quella di superare alcune delle principali limitazioni dei Large Language Models (LLM), come la mancanza di accesso a informazioni aggiornate e la mancanza di specificità. Questa tecnica inoltre permette di estendere i casi d'uso dei modelli di generazione del linguaggio, ad utilizzi con dati privati, specifici o che più in generale non sono presenti nei dataset di addestramento. La Figura 2.4 mostra una rappresentazione schematica del funzionamento di un modello RAG.

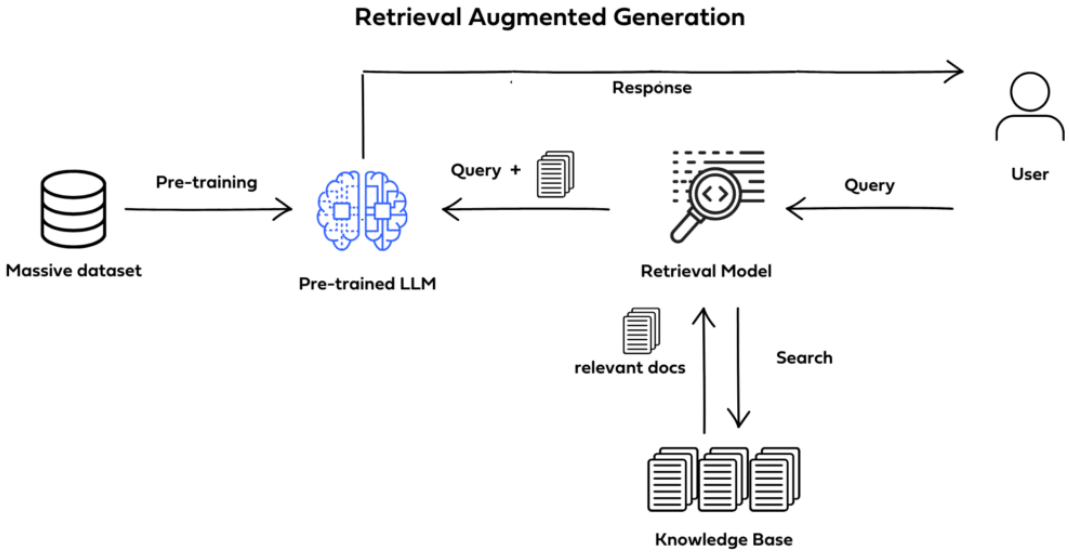

Figura 2.4: Rappresentazione schematica di un modello RAG [27]

2.3.1 Architettura di RAG

L'architettura di un modello RAG è composta da due componenti chiave: il modulo di retrieval e il modulo di generazione, che lavorano in sinergia per recuperare informazioni rilevanti e generare risposte accurate.

Modulo di retrieval

Il modulo di retrieval è responsabile di recuperare informazioni rilevanti da una base di conoscenza esterna, che può essere un database strutturato, un corpus di testi o qualsiasi altra fonte di informazioni. Questo può essere fatto utilizzando una serie di tecniche di recupero testuale basate su modelli di retrieval tradizionali o su modelli più moderni, basati su embedding.

- **Tecniche tradizionali:** le tecniche tradizionali di retrieval testuale si basano su algoritmi di ricerca, come BM25 o TF-IDF (Term Frequency-Inverse Document Frequency), che calcolano la rilevanza tra una query e un insieme di documenti. Queste tecniche analizzano semplicemente la frequenza delle parole nei documenti e nel corpus generale, per determinare la pertinenza di

un documento rispetto a una richiesta. Questi metodi sono efficaci per recuperare informazioni basate su parole chiave, ma possono essere limitati in contesti più complessi, in cui è necessario considerare il contesto e la semantica delle parole, nonostante ciò sono ancora ampiamente utilizzati per la loro semplicità e velocità.

- **Tecniche basate su embedding:** tecniche più moderne, come il Dense Passage Retrieval (DPR) [28], utilizzano modelli basati su reti neurali per rappresentare sia le query che i documenti in uno spazio vettoriale denso, in modo da catturare le relazioni semantiche tra le parole. Questo approccio consente di ottenere risultati più accurati e rilevanti rispetto ai metodi tradizionali, aumentando l'efficacia del retrieval in contesti complessi.

Il processo di retrieval restituisce un insieme di documenti o frammenti, ordinati per rilevanza, che verranno successivamente utilizzati dal modulo di generazione per generare la risposta.

Modulo di generazione

Il modulo di generazione è responsabile di generare la risposta finale, combinando le informazioni recuperate dal modulo di retrieval con il contesto dell'input dell'utente. Si basa su un Large Language Model (LLM) pre-addestrato, che verrà utilizzato per generare la risposta in modo coerente e pertinente, combinando le informazioni estratte con il contesto dell'input. Il modulo di generazione può effettuare il processo di generazione in due modalità principali:

- **Fusion-in-decoder:** in questa modalità, il modello di generazione riceve come input sia la query che i documenti recuperati e utilizza entrambe le fonti per la costruzione della risposta. Questo approccio è particolarmente adatto per task in cui sono necessarie risposte complesse e articolate.

- **Fusion-in-encoder:** in questo caso, la query e i documenti recuperati vengono uniti a livello di encoding, producendo una rappresentazione combinata che viene poi usata dal decoder in fase di generazione. Questa versione è più adatta a contesti in cui l'integrazione di contenuti multipli è meno critica, ma si da maggior importanza alla rapidità e consistenza.

L'obiettivo di questa componente è quello di produrre una risposta che non solo sia grammaticalmente corretta e coerente, ma che sia anche arricchita dalle informazioni recuperate, migliorando la pertinenza e la specificità.

2.3.2 Funzionamento di RAG

Il funzionamento di un modello RAG può essere suddiviso in vari passaggi, che permettono di combinare il processo di retrieval con quello di generazione.

1. **Input dell'utente:** il processo inizia con l'input dell'utente, che può essere una domanda, una richiesta o qualsiasi altra forma di testo. Questo viene utilizzato per la generazione di una query di retrieval, che verrà inviata all'apposito modulo.

2. **Recupero delle informazioni:** la query viene inviata al modulo di retrieval, che esegue una ricerca nel corpus o nella base di conoscenza esterna, restituendo un insieme di documenti o frammenti rilevanti. A seconda dell'algoritmo utilizzato, verranno selezionati i documenti che rispondono meglio alla query e saranno opportunamente ordinati.

3. **Generazione della risposta:** il modello generativo riceve la query dell'utente e i documenti recuperati e utilizzerà entrambe le fonti per generare una risposta completa. Il modello non si limiterà a utilizzare la sua conoscenza pregressa, ma integrerà anche informazioni esterne, specifiche e aggiornate.

Questo approccio consente a RAG di combinare la potenza di un LLM con l'accuratezza dei modelli di retrieval, andando a superare alcune delle principali limitazioni dei modelli di generazione del linguaggio. Nella Figura **??** è mostrato un esempio approfondito, di come un modello RAG funziona.

2.3.3 RAG tradizionale

Il RAG tradizionale, anche detto RAG Naive, è la versione basilare del modello Retrieval-Augmented Generation e segue il framework "Retrieve-Read" spiegato nel paper "Query Rewriting for Retrieval-Augmented Large Language Models" [29]. Il funzionamento di un modello RAG tradizionale è a grandi linee quello descritto

H

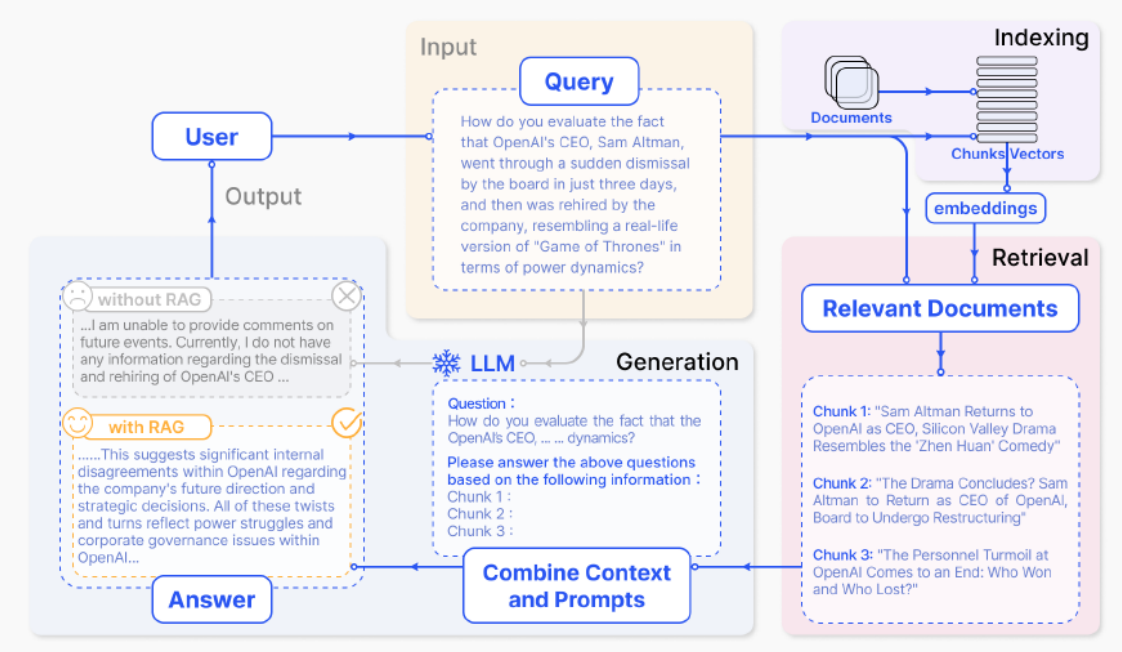

Figura 2.5: Esempio di funzionamento di un modello RAG

nella sezione precedente. Nel dettaglio, il processo inizia con una fase di Indexing, dove i documenti vengono prima convertiti in un formato uniforme, poi vengono segmentati in chunks più piccoli e digeribili, per poi codificarli. Le rappresentazioni vettoriali ottenute vengono memorizzate in un database vettoriale, che è una delle componenti base di un sistema di RAG Naive. Successivamente, quando arriva una query, questa viene codificata, utilizzando lo stesso modello utilizzato in fase di indicizzazione. La query codificata viene confrontata con i documenti memorizzati nel database vettoriale, calcolando la similarità semantica tra il vettore della query e quelli dei chunk. I primi k chunks più simili vengono selezionati e inviati al modello generativo, che li usera come contesto da aggiungere al prompt, per generare la risposta. Nella fase di generazione, creato il prompt finale, il LLM formula la sua risposta.

Vantaggi e limitazioni

Questo approccio è molto efficace in determinati casi e per certi task, riuscendo a migliorare le performance dei modelli di generazione del linguaggio, tramite un meccanismo semplice e diretto. Nonostante i vantaggi, il RAG Naive ha alcune

limitazioni che lo rendono inadeguato per task che richiedono una comprensione contestuale profonda o un'integrazione più sofisticata delle informazioni.

- **Trattamento dei documenti come entità indipendenti:** il RAG Naive tratta i documenti recuperati come entità indipendenti, senza modellare le relazioni tra di essi. In scenari complessi, dove le informazioni devono essere collegate tra più fonti, questo approccio può portare a risposte incomplete o incoerenti.
- **Gestione di relazioni complesse:** il modello fatica a gestire e catturare relazioni complesse tra i documenti. In task come il multi-hop reasoning o la query-focused summarization, dove è richiesto il collegamento di informazioni da più fonti, il modello può non essere in grado di fornire risposte accurate.
- **Perdita di coerenza:** in alcuni casi, l'integrazione di informazioni esterne può portare a risposte incoerenti o fuorvianti, se non gestita correttamente. Questa problematica si verifica soprattutto in caso di informazioni contrastanti tra i vari documenti recuperati.

Queste limitazioni rendono il RAG Naive inadeguato per task complessi che richiedono una comprensione più profonda e una connessione tra informazioni multiple.

2.3.4 Graph RAG

Un approccio alternativo al RAG tradizionale e alle sue varianti evolutive è il GraphRAG, che si basa sull'integrazione di modelli di generazione del linguaggio con grafi di conoscenza (knowledge graphs), al fine di migliorare la qualità e la pertinenza delle risposte fornite dai modelli di linguaggio naturale (LLM). Il GraphRAG è stato progettato per affrontare le limitazioni del RAG tradizionale, specialmente nei casi in cui l'informazione è complessa, distribuita e necessita di una rappresentazione strutturata per essere interpretata correttamente.

L'approccio GraphRAG sfrutta la potenza dei grafi di conoscenza, che offrono una rappresentazione strutturata di entità e relazioni, consentendo un'integrazione più profonda e semantica delle informazioni durante il processo di retrieval e generazione. Questo metodo è particolarmente utile per migliorare l'accuratezza

delle risposte in compiti complessi, come il question answering o l'estrazione di informazioni da contesti altamente specializzati.

Il GraphRAG si distingue dal RAG tradizionale per la sua capacità di utilizzare un grafo di conoscenza come fonte di informazioni aggiuntive nel processo di retrieval. Invece di basarsi esclusivamente su documenti di testo frammentati, il GraphRAG utilizza una rappresentazione strutturata delle entità e delle loro relazioni, fornendo al modello generativo una comprensione più profonda del contesto semantico.

Come descritto da Ontotext [30], il GraphRAG utilizza grafi di conoscenza in tre modalità principali:

- **Graph come store di contenuti:** In questa modalità, i grafi di conoscenza sono utilizzati per estrarre chunks rilevanti dai documenti. Questi chunks vengono inviati al modello generativo come contesto, analogamente a come funziona un RAG tradizionale. Tuttavia, il vantaggio di questa modalità è che il grafo contiene non solo testo, ma anche metadati che descrivono la relazione tra i documenti, migliorando la pertinenza delle informazioni estratte.

- **Graph come esperto del dominio:** In questo caso, il grafo non si limita a fornire testo grezzo, ma piuttosto descrizioni concettuali delle entità e delle relazioni tra di esse. Questo fornisce al modello un contesto semantico molto più ricco, permettendo di rispondere in modo più preciso a domande che richiedono una comprensione profonda di concetti tecnici o di dominio.

- **Graph come database:** Qui, parte della query in linguaggio naturale viene mappata direttamente su una query del grafo, come una query SPARQL, e il risultato viene poi inviato al modello generativo. Questo approccio è particolarmente utile quando il grafo contiene fatti strutturati e relazioni dirette che possono essere utilizzati per rispondere a domande specifiche.

In Figura 2.6 è mostrato un riassunto schematico delle tre modalità di utilizzo dei grafi di conoscenza nel GraphRAG. Questi approcci offrono vantaggi significativi rispetto al RAG tradizionale, in particolare nella gestione di informazioni complesse e nell'integrazione di dati strutturati.

Graph RAG Varieties	Semantic Metadata	Domain Knowledge	Factual Data
Vanilla "chunky" RAG			
Type 1: **Graph as a Metadata Store**	+	–	–
Type 2: **Graph as an Expert**	+	+	–
Type 3: **Graph as a Database**	+	+	+

Figura 2.6: Schema dei tipi di GraphRAG [30]

Questo approccio ad oggi è uno dei più avanzati e promettenti per migliorare le performance dei modelli di generazione del linguaggio. Si tratta però di una tecnica solo parzialmente sviluppata e che richiede ancora molte ricerche e sperimentazioni per essere pienamente applicabile a situazioni differenti. Lo studio più interessante e rilevante ad oggi è quello di Edge et al. (2024) [31], che ha implementato un modello GraphRAG per la Query-focused Summarization, mostrando poi come il sistema sia in grado di superare le performance di un RAG tradizionale. Altre implementazioni e ricerche sono attualmente in corso, per poter estendere l'applicabilità del GraphRAG a task vari e complessi.

Vantaggi e limitazioni

Questo insieme di tecniche offre una serie di potenziali vantaggi, che spaziano dalla maggiore accuratezza e pertinenza delle risposte, alla capacità di gestire informazioni complesse e specializzate. Utilizzando grafi di conoscenza è inoltre possibile integrare informazioni strutturate più facilmente. Anche l'aggiornamento delle informazioni è semplificato, grazie alla struttura flessibile dei grafi, che possono essere facilmente modificati e aggiornati. Un ultimo vantaggio particolarmente importante è la spiegabilità, infatti i grafi di conoscenza offrono una rappresentazione chiara e strutturata delle informazioni, che può essere facilmente interpretata e analizzata. Il Graph-RAG è però soggetto anche esso a limitazioni, in primis la mancanza di soluzioni consolidate e tecnologie mature, che rendono questo approccio ancora difficilmente applicabile in contesti reali e complessi. Le criticità di questa tecnica sono principalmente legate a una scarsa diffusione e alla mancanza di letteratura e

ricerche specifiche. Col tempo e con lo sviluppo di nuove tecnologie e metodologie, il GraphRAG potrebbe diventare uno degli approcci più promettenti per migliorare le performance dei modelli di generazione del linguaggio. Si possono comunque già capire delle possibili problematiche, come la complessità di gestione dei grafi di conoscenza, la necessità di competenze specifiche, l'elevato costo computazionale e la difficoltà di integrazione con sistemi esistenti.

2.3.5 Approcci ibridi

Gli approcci ibridi nel contesto del Retrieval-Augmented Generation (RAG) combinano le caratteristiche del GraphRAG e del RAG tradizionale, cercando di sfruttare i vantaggi di entrambi per migliorare le prestazioni in scenari complessi. L'idea principale di questi approcci è quella di utilizzare contemporaneamente sia la potenza dei grafi di conoscenza (KG) sia i chunk testuali recuperati tramite vettori da database vettoriali, al fine di migliorare la precisione e la coerenza delle risposte generate dai modelli di linguaggio. L'uso di un approccio ibrido risponde alla necessità di superare alcune delle limitazioni sia del RAG tradizionale sia del GraphRAG. Mentre il GraphRAG è particolarmente efficace nel modellare relazioni semantiche complesse tra entità e relazioni, il RAG tradizionale e i suoi sviluppi più moderni, come il VectorRAG, si sono dimostrati più adatti al recupero di grandi quantità di dati non strutturati e di natura testuale. Tuttavia, in compiti particolarmente complessi, entrambi gli approcci mostrano dei limiti se utilizzati singolarmente.

La direzione di ricerca attualmente più promettente è quella di integrare i due approcci, sfruttando i punti di forza dell'uno per compensare i limiti dell'altro. Il paper di Sarmah et al. (2024) [32] introduce il concetto di HybridRAG, un approccio ibrido che combina retrieval da knowledge graphs e da vector databases per migliorare i sistemi di question-answering (Q&A) in contesti come l'estrazione di informazioni finanziarie. Questo tipo di sistema ottiene performance superiori rispetto ai due approcci singoli. Anche Edge et al. (2024) [31] hanno proposto un approccio ibrido, che oltre alla soluzione proposta per task di query-focused summarization, presenta un sistema di ricerca ibrida. La ricerca in questione va a suddividere la finestra di contesto in due parti: una dedicata ai chunk recuperati tramite vettori e l'altra ai nodi e agli archi del grafo di conoscenza. Le informazioni provenienti sia dai chunk che dal grafo vengono poi integrate in un unico prompt,

che viene inviato al modello generativo, generando una risposta più accurata e pertinente.

Questi approcci ibridi sono ancora in fase di sviluppo e sperimentazione, ma promettono di offrire una soluzione più completa e flessibile per superare i limiti attuali. La ricerca in questo campo è in rapida evoluzione, e con il progresso delle tecnologie, ci si aspetta che l'integrazione di approcci ibridi possa essere ottimizzata ulteriormente. L'obiettivo della ricerca è integrare al meglio questi approcci, affrontando le problematiche relative all'elevato costo computazionale e alla complessità di integrare queste tecnologie in sistemi reali. Una volta superati questi ostacoli, gli approcci ibridi potrebbero rivoluzionare il campo del Retrieval-Augmented Generation, rendendoli un'opzione praticabile in un'ampia gamma di settori.

Capitolo 3

Design della sperimentazione

In questo capitolo viene descritto il design della sperimentazione condotta per valutare e confrontare diverse architetture e approcci di Retrieval Augmented Generation. Verranno presentate le scelte progettuali, l'architettura su cui è stata condotta la sperimentazione, le pipeline implementate e le tecnologie utilizzate. Inoltre, saranno descritti i dati impiegati e le metriche adottate per valutare le performance dei modelli. Ogni scelta progettuale viene discussa e motivata, al fine di fornire una panoramica completa del lavoro svolto e mostrare come è stato possibile ottenere i risultati, che saranno presentati nel capitolo successivo.

3.1 Introduzione al design della sperimentazione

Questa sezione fornisce una visione d'insieme del design della sperimentazione, illustrando gli obiettivi e le motivazioni principali che hanno guidato lo sviluppo del sistema. Il progetto è stato svolto in collaborazione con l'azienda *E4-Analytics*, che ha fornito il supporto e le risorse necessarie per lo sviluppo del sistema, rispondendo al proprio interesse nell'esplorazione di approcci innovativi per la generazione di testo automatica. L'azienda ha messo a disposizione un ambiente di sviluppo avanzato, costituito da tecnologie e strumenti appositamente progettati per supportare lo sviluppo di sistemi di intelligenza artificiale. Il progetto è stato supervisionato da un team di Machine Learning Engineers, che ha fornito supporto e consulenza durante il processo di sviluppo.

3.2 Obiettivi della sperimentazione

Il Graph RAG rappresenta un'alternativa al RAG tradizionale che ha acquisito popolarità solo negli ultimi mesi, e le sue potenzialità non sono ancora state completamente esplorate. L'obiettivo principale di questo progetto di tesi è valutare le prestazioni del Graph RAG, partendo da un'analisi delle tecnologie attualmente disponibili per implementare una pipeline di questo tipo. Questo progetto mira a esplorare vari aspetti del Graph RAG, dalla progettazione dell'architettura alla valutazione delle performance, per comprendere come l'approccio possa essere utilizzato in vari contesti applicativi. In particolare, l'obiettivo principale è valutare l'applicabilità del Graph RAG in situazioni differenti e confrontare i risultati con quelli ottenuti attraverso il RAG tradizionale. Attraverso uno studio comparativo, sarà possibile identificare i vantaggi e gli svantaggi di entrambi gli approcci, valutandone l'efficacia in modo individuale e complementare. Durante la sperimentazione verranno affrontati task differenti e tipi di dati diversi, per poter valutare le performance dei modelli in contesti differenti e comprendere come l'approccio possa essere adattato a situazioni specifiche. In particolare verrà affrontato un classico problema di question answering, tipico per testare sistemi RAG, e un task di query focused summarization, più complesso e che richiede una maggiore comprensione del testo di input. L'obiettivo finale del progetto è ottenere un sistema che sappia fondere i punti di forza di entrambi gli approcci, arrivando, se necessario, a una tecnica ibrida che possa integrare le caratteristiche del RAG tradizionale con quelle del Graph RAG. Un sistema ibrido, infatti, potrebbe riuscire a risolvere task complessi sfruttando l'ampiezza delle informazioni non strutturate tipiche del RAG tradizionale insieme alla ricchezza semantica dei grafi di conoscenza del Graph RAG.

3.3 GraphRAG

Il cuore del lavoro di tesi è la ricerca di approcci alternativi alle tecniche di RAG tradizionale, che possano ampliare le funzionalità di sistemi RAG, rendendoli performanti su differenti tipi di dati e task. Graph RAG si presenta come una possibile soluzione, capace di migliorare i sistemi esistenti e di offrire nuove possibilità di ricerca e analisi. Le basi teoriche di questo approccio sono state descritte in dettaglio

nella sezione 2.3.4. Sulla base di questi studi, si è deciso di procedere con l'implementazione di un sistema di Graph RAG, che potesse essere testato e valutato su differenti task. Ciò che si cerca di ottenere con l'introduzione di GraphRAG, è un sistema in grado di elaborare le informazioni in maniera approfondita, estraendo concetti e relazioni tra di essi. Da un sistema di questo tipo ci si aspetta che sia in grado di rispondere a domande complesse, di effettuare dei ragionamenti ed inoltre di aumentare l'interpretabilità dei risultati ottenuti, tramite la visualizzazione delle strutture estratte e generate. Basandosi su queste premesse, si è proceduto con l'implementazione di un sistema di Graph RAG, partendo dallo studio e dall'analisi delle librerie e dei framework disponibili, per poi passare alla fase di implementazione di una pipeline da utilizzare in fase di sperimentazione.

3.3.1 Tecnologie per il GraphRAG

Per l'implementazione di un sistema GraphRAG, è stata svolta una prima fase di ricerca e valutazione delle tecnologie disponibili, volta a identificare i framework e gli strumenti più adatti alle esigenze del progetto. Data la natura emergente di GraphRAG, molte soluzioni sono ancora in fase di sviluppo, e solo pochi framework offrono funzionalità avanzate e pronte all'uso per l'integrazione dei knowledge graph nei modelli di linguaggio. Durante la fase di ricerca, ci si è imbattuti in soluzioni incomplete, non documentate, o disponibili unicamente a pagamento. Alcune tecnologie sono state quindi semplicemente analizzate, mentre altre sono state anche testate. Questa fase di esplorazione è stata essenziale per selezionare la tecnologia più promettente, valutando ogni soluzione sulla base di criteri come la facilità di integrazione, la disponibilità di supporto e le funzionalità offerte. Le soluzioni prese in considerazione si suddividono in framework appositi per il GraphRAG, integrazioni avanzate nei database a grafo e implementazioni con librerie di intelligenza artificiale generica. Nella sezione seguente, verranno descritte tutte le tecnologie esplorate, con un focus particolare su quelle più promettenti.

1. Estensioni di database a grafo

I database a grafo sono una categoria di database ottimizzati per la gestione e l'analisi di dati fortemente interconnessi, che rappresentano informazioni sotto forma di nodi (entità) e relazioni (connessioni tra entità). Questa tipologia di database

utilizza strutture flessibili che consentono di modellare in modo naturale reti complesse di informazioni, rendendo i grafi un'opzione ideale per la gestione di molte tipologie di dati.

Grazie alla loro struttura, si prestano naturalmente al supporto di knowledge graph, che possono essere memorizzati e interrogati in modo efficiente. Queste caratteristiche rendono i database a grafo una tecnologia interessante per sistemi di GraphRAG. Per questo motivo alcuni dei principali database a grafo hanno iniziato a integrare dei moduli specializzati per il supporto di modelli di linguaggio, offrendo strumenti per la creazione di pipeline di GraphRAG. Alcune di queste tecnologie sono state analizzate per il progetto di tesi, e verranno descritte di seguito.

Neo4j Neo4j è senza dubbio il database a grafo più noto e diffuso. Offre una vasta gamma di servizi ed estensioni, e ha integrato nell'ultimo periodo un modulo per l'integrazione di knowledge graph in pipeline di generazione e recupero. Questo modulo consente di combinare la ricerca vettoriale con l'interrogazione di un grafo di conoscenza, migliorando il contesto e la precisione delle risposte fornite dai modelli di linguaggio. L'approccio di Neo4J, presentato da Rathle [33], consiste in una pipeline composta da due fasi principali:

- **Creazione del grafo di conoscenza:** a partire da documenti non strutturati, come PDF o pagine web, viene effettuata una estrazione di entità e relazioni generando un grafo.
- **Ricerca mista vettoriale e basata su grafo:** una query di ricerca vettoriale individua i nodi rilevanti nel grafo, dopodichè Neo4j si occupa di effettuare una ricerca basata su grafo per estrarre ulteriori informazioni e contestualizzare la risposta.

In Figura 3.1 è mostrata la pipeline proposta da Neo4j per l'implementazione di GraphRAG, che combina la ricerca vettoriale con la ricerca basata su grafo. Questo approccio si propone di offrire risposte più accurate e contestualizzate, una comprensione più approfondita del testo, una iterazione più veloce e una maggiore interpretabilità dei risultati, grazie alla struttura a grafo. Questa implementazione rende Neo4j un'opzione potente per integrare knowledge graph con modelli di linguaggio.

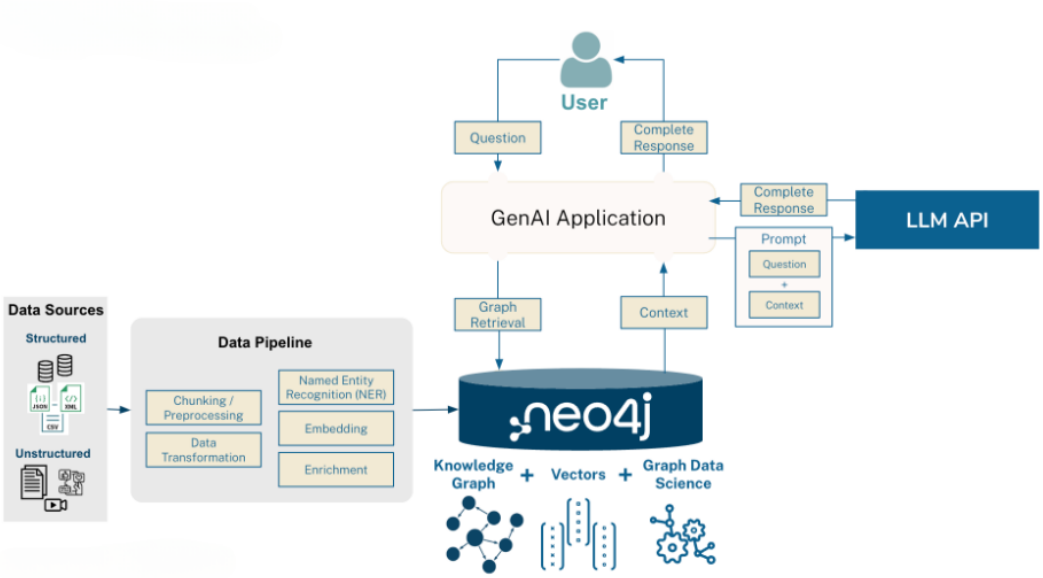

Figura 3.1: Pipeline di Neo4j per GraphRAG

Al momento, l'implementazione di GraphRAG su Neo4j è limitata all'uso di modelli di linguaggio a pagamento, accessibili tramite API esterne, come quelli offerti da provider quali OpenAI. Ciò significa che non è possibile configurare modelli di linguaggio propri e auto-ospitati nel cluster interno. Questa restrizione rappresenta una sfida per chi desidera mantenere l'infrastruttura interamente on-premise e gestire i modelli internamente, limitando la possibilità di integrazione con LLM personalizzati.

Nebula Graph NebulaGraph è un database a grafo distribuito e open-source progettato per gestire efficacemente dati complessi e interconnessi, tipici di knowledge graph estesi. NebulaGraph ha introdotto per primo un modulo dedicato al GraphRAG [34], al fine di estendere le funzionalità del database e offrire un supporto avanzato per l'integrazione di modelli di linguaggio. La tecnica GraphRAG, sviluppata da NebulaGraph, permette di combinare LLM e knowledge graph per fornire risultati di ricerca più contestualizzati e precisi. Questa combinazione è particolarmente utile per le query complesse e lunghe, dove i modelli tradizionali di RAG basati solo su vettori non sono sufficienti a catturare l'intera complessità del contesto. Questa versione funziona integrando knowledge graph con modelli tramite l'utilizzo di librerie come LlamaIndex e LangChain, implementando una versione

di recupero basata sia su vettori che su grafi. Il processo di indicizzazione e ricerca è mostrato in Figura 3.2, dove si evidenzia la fase di creazione degli embedding e conseguente indicizzazione, seguita dalla fase di ricerca basata su grafo e similarità vettoriale.

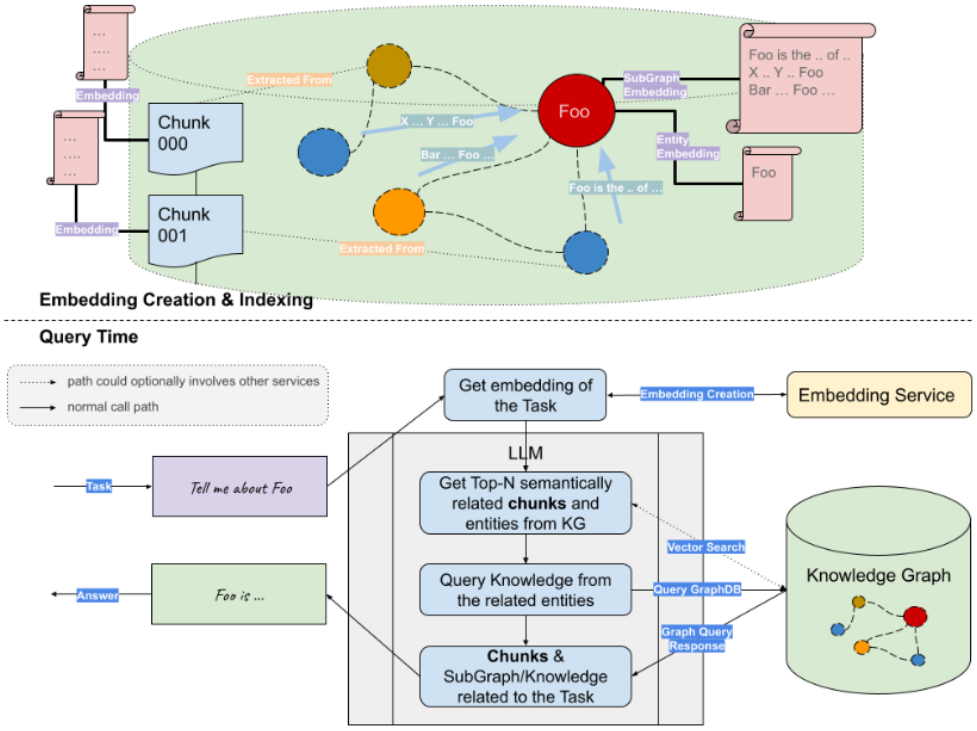

Figura 3.2: Pipeline completa di GraphRAG in NebulaGraph

La soluzione di NebulaGraph è interessante, ma appare ancora incompleta, in quanto si appoggia su altre tecnologie esterne, non specifiche per il GraphRAG.

Memgraph Memgraph è un database a grafo in tempo reale progettato per supportare flussi di dati ad alta frequenza, specialmente in scenari che richiedono l'elaborazione e l'analisi immediata di dati in streaming. A differenza di altri database a grafo che si concentrano su query statiche o su dati archiviati, Memgraph è ottimizzato per casi d'uso dinamici, ma può essere utilizzato anche in casi tradizionali. Recentemente ha integrato il supporto per GraphRAG [35], consentendo la gestione dinamica e l'aggiornamento in tempo reale del grafo di conoscenza. Le principali

feature messe a disposizione per l arealizzazione di pipeline complesse di GraphRAG sono:

- **Algoritmi di deep path traversal:** per l'analisi di cammini complessi e la ricerca di relazioni nascoste o complesse.
- **Algoritmi di community detection:** come Louvain e PageRank, per identificazione di cluster e strutture di comunità.
- **Integrazione con dati in streaming:** per l'aggiornamento in tempo reale del grafo, supportando l'ingestione da sorgenti come Kafka o Redpanda.
- **Funzionalità di query avanzate:** consentendo interrogazioni tramite linguaggio naturale, convertibile in Cypher e integrando LLM con LangChain o LLamaIndex.

Le componenti descritte si vanno a inserire in un sistema architetturale simile a quello mostrato in Figura 3.3.

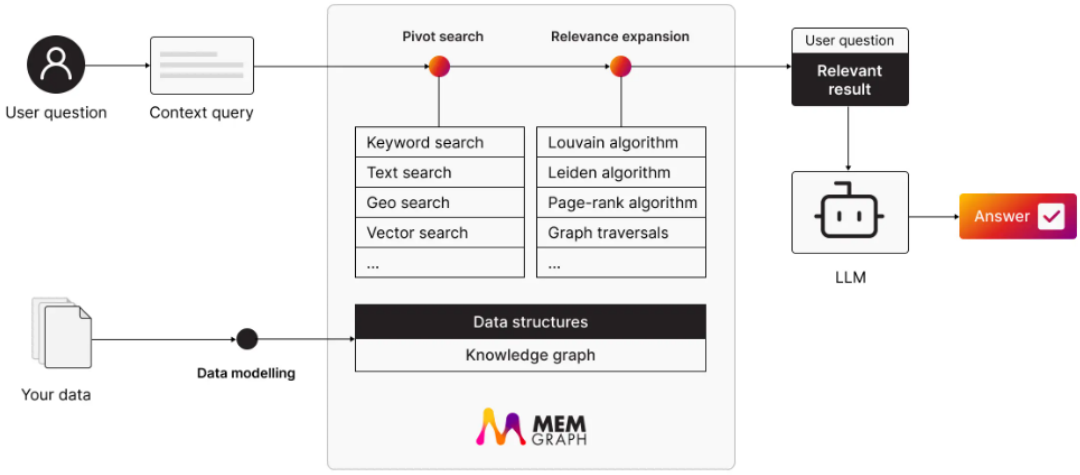

Figura 3.3: Struttura del sistema di GraphRAG in Memgraph

In sintesi, Memgraph offre un'infrastruttura completa per la realizzazione di pipeline di GraphRAG customizzabili, tramite una serie di strumenti pronti per l'uso e funzionalità avanzate per l'analisi e l'aggiornamento dinamico del grafo di conoscenza. L'approccio di Memgraph è particolarmente interessante, ricco di funzionalità avanzate e adatto a scenari anche molto complessi, inoltre i ricercatori hanno

già tracciato un percorso di sviluppo per l'integrazione di numerose funzionalità aggiuntive. La limitazione principale di Memgraph è la mancanza di materiale a supporto, di documentazione dettagliata e di esempi pratici.

2. Framework per il GraphRAG

Viste le grandi potenzialità dell'applicazione di GraphRAG, oltre all'integrazione sui database a grafo, stanno emergendo numerosi framework specializzati per la realizzazione di sistemi di questo tipo. Molte grandi aziende e centri di ricerca di intelligenza artificiale si stanno muovendo in questa direzione, con l'obiettivo specifico di unire i punti di forza dei knowledge graph e delle tecnologie di retrieval avanzate, evitando i limiti delle semplici estensioni di database a grafo o di librerie di intelligenza artificiale adattate per lo scopo. L'introduzione di framework pensati appositamente per il GraphRAG risponde alla necessità di gestire in modo nativo le complessità legate alla generazione aumentata dal retrieval, permettendo una più agevole integrazione di knowledge graph con modelli di linguaggio. Questi strumenti dedicati puntano ad ottimizzare ogni aspetto per la realizzazione di sistemi performanti, scalabili e fortemente personalizzabili, offrendo grande flessibilità agli utilizzatori, mettendo a disposizione una vasta gamma di strumenti.

Nonostante la grande attenzione che si sta dedicando a questi framework, molti di essi sono ancora in fase di sviluppo e non offrono ancora tutte le funzionalità desiderate. Molti altri limitano l'accesso solo a utenti di piattaforme cloud o modelli a pagamento, rendendo difficile l'integrazione con sistemi on-premise e limitando la possibilità di personalizzazione. Nonostante il mercato non sia ancora maturo, è stato individuato un framework che si è dimostrato particolarmente promettente.

Microsoft GraphRAG Il principale framework per il GraphRAG è stato sviluppato da Microsoft, che ha progettato una soluzione completa per affrontare sfide multiple legate alla generazione e al retrieval di testo. Microsoft GraphRAG è un framework open-source [36] che sfrutta grafi di conoscenza generati da modelli di linguaggio di grandi dimensioni (LLM) per rispondere a query su dati privati o non precedentemente visti. È stato sviluppato per superare i limiti del RAG tradizionale, che utilizza principalmente la ricerca vettoriale per rispondere a domande basate su dati testuali, che non è sufficientemente efficace per alcuni task.

Questo approccio funziona su due fasi principali:

1. **Indicizzazione:** questa fase consiste nella creazione di un grafo di conoscenza a partire da documenti non strutturati, tramite estrazione di entità e relazioni effettuata da modelli di linguaggio. La creazione del grafo ad opera di LLM permette di catturare relazioni complesse e nascoste tra entità, migliorando la comprensione del testo e la qualità delle risposte. In seguito alla creazione del grafo, possono essere applicati dei meccanismi di community detection per identificare cluster e strutture di comunità, che vadano a racchiudere entità simili o correlate. Questa fase è completamente customizzabile, tramite una vasta gamma di parametri, configurazioni e prompt, che permettono di adattare il grafo alle esigenze specifiche del task.

2. **Ricerca:** GraphRAG implementa due tipologie di ricerca, per rispondere a domande di natura diversa.

    - **Ricerca globale:** pensata per task di query-focused summarization, che richiedono una comprensione globale del testo e la generazione di risposte complete e contestualizzate. Per rispondere a questo tipo di domande vengono utilizzate le comunità identificate nella fase di indicizzazione, che riassumono le informazioni principali.
    - **Ricerca locale:** per task più generici di question answering, che richiedono risposte più precise e specifiche. In questo caso, la ricerca è ibrida e combina la ricerca vettoriale con la ricerca basata su grafo, per estrarre informazioni precise.

Questo framework è stato sviluppato per essere usato su dati narrativi [37], ma data la sua grande flessibilità e personalizzabilità, può essere adattato a molteplici task e tipologie di dati. Microsoft GrapRAG rappresenta dunque la scelta ideale per chi desidera realizzare un sistema di GraphRAG complesso che possa affrontare task diversi e mutevoli.

3. Soluzioni basate su librerie di intelligenza artificiale

Oltre ai framework specializzati e alle estensioni di database a grafo, esistono anche soluzioni basate sulle più classiche librerie di intelligenza artificiale, dedicate

all'interazione con LLM. Queste librerie offrono già funzionalità avanzate per task di generazione e retrieval, e stanno aggiungendo dei moduli specifici per l'integrazione con knowledge graph. Questo tipo di approccio, seppur non specifico per il GraphRAG, può essere una soluzione valida per chi desidera integrare rapidamente funzionalità di retrieval basati su grafi, in sistemi già esistenti, che si basano su queste tecnologie. Nel corso della ricerca sono state analizzate diverse librerie, ma è stata presa in considerazione solo una soluzione, che si è dimostrata particolarmente completa.

GraphRAG by LlamaIndex LlamaIndex è una libreria open-source sviluppata da Meta AI, molto diffusa per implementazione di sistemi RAG. La libreria, da poco, ha introdotto due specifici moduli per l'integrazione di knowledge graph nei modelli di linguaggio. L'approccio al GraphRAG di LlamaIndex è simile a quello di Microsoft, con una fase di indicizzazione e una fase di ricerca. Il focus anche in questo caso è sulla creazione di un grafo di conoscenza tramite LLM e il sistema di retrieval è ottimizzato per rispondere a domande complesse e globali, utilizzando anche in questo caso comunità estratte dal grafo. Le due versioni differiscono sulla gestione del grafo, la prima salvando il grafo in memoria e la seconda utilizzando un database a grafo per la memorizzazione. La versione con database a grafo è particolarmente interessante, si appoggia su Neo4j ed ha dimostrato di essere molto performante e scalabile. LlamaIndex è una soluzione completa e flessibile, ad oggi non offre un supporto completo per pipeline ibride, ma questa funzionalità è in fase di sviluppo, insieme ad altre feature avanzate. Questa tecnologia ancora non completa, si propone come una delle soluzioni potenzialmente più valide per il futuro.

4. Scelta del framework

Al termine dell'analisi approfondita delle tecnologie disponibili, si è deciso di puntare sulla soluzione di Microsoft. La scelta è stata dettata dalla completezza e dalla flessibilità del framework, che offre funzionalità avanzate e personalizzabili per la realizzazione di sistemi di GraphRAG. Inoltre un punto a favore è la presenza di una vasta comunità di sviluppatori e di un supporto attivo, che garantiscono un continuo sviluppo e miglioramento del framework. Microsoft GraphRAG si è dimostrato

particolarmente adatto per il progetto in questione, poichè consente di utilizzare modelli open source esposti nel cluster aziendale, permettendo di rispettare i vincoli di sicurezza e privacy dei dati e di non incorrere in costi aggiuntivi.

Le altre tecnologie esaminate, seppur valide, hanno dimostrato diverse limitazioni, come la mancanza di documentazione, la scarsa diffusione, la poca personalizzabilità, o la necessità di utilizzare modelli o sistemi a pagamento. La scelta è stata effettuata tenendo conto delle caratteristiche delle tecnologie e delle esigenze del progetto e dell'azienda. Lo sviluppo del sistema di GraphRAG, sviluppato basandosi sulla tecnologia scelta, sarà descritto nella sezione successiva.

3.3.2 Pipeline di GraphRAG

Dopo aver scelto il framework adatto per la realizzazione di un sistema di GraphRAG, si è proceduto con la progettazione e l'implementazione di una pipeline completa, che potesse essere utilizzata nella fase sperimentale del progetto. Come già descritto, Microsoft GraphRAG permette di creare pipeline personalizzate, che usano come base i suoi schemi di indicizzazione e ricerca predefiniti, ma che possono essere adattate e personalizzate per rispondere a esigenze specifiche.

La libreria nella sua implementazione standard, utilizza come modelli (LLM e embedding-model), quelli di OpenAI, richiedendo l'accesso a servizi esterni tramite API e utilizzo di token. La prima personalizzazione necessaria è stata dunque quella di andare a sostituire i modelli di OpenAI con quelli esposti nel cluster aziendale. Come LLM è stato utilizzato il modello Llama3.1-8B, mentre per l'embedding è stato configurato il modello Nomic-embed-text. Per attuare queste modifiche è bastato creare un'istanza di GraphRAG, specificando i modelli da utilizzare nel file di configurazione. È stato necessario modificare anche alcuni script della libreria, che gestivano la generazione di embedding, per adattarli a modelli esposti con Ollama.

A questo punto si è potuto procedere con la creazione della pipeline di indicizzazione, che verrà descritta nel dettaglio, presentandone ogni singolo step.

Pipeline di indicizzazione: gli step

Per la sperimentazione sono state definite pipeline diverse, ma con una struttura comune. Le differenze tra le varie versioni sono determinate solo dai modelli utilizzati e dai parametri di configurazione. La pipeline di indicizzazione è composta da una serie di passaggi, ognuno dei quali svolge una funzione specifica.

- **Prompt tuning (opzionale):** questa operazione non fa parte della vera e propria pipeline di indicizzazione, ma è una operazione precedente che permette di ottimizzare i prompt utilizzati per l'indicizzazione. Questo step è opzionale, non sempre è necessario, ma in fase di sperimentazione è stato effettuato un test per confrontare una indicizzazione con prompt standard e una con prompt ottimizzati. Questa funzionalità avanzata permette di personalizzare i prompt in due modalità possibili, una manuale, in cui è l'utente a specificare i prompt, e una automatica, in cui il sistema genera prompt ottimizzati in base ai dati. L'opzione manuale è consigliabile solo in casi in cui si ha una forte conoscenza del dominio o si vuole personalizzare fortemente il prompt, altrimenti è preferibile utilizzare l'opzione automatica. In fase di sperimentazione è stata utilizzata l'opzione automatica, il cui funzionamento è mostrato in figura 3.4.

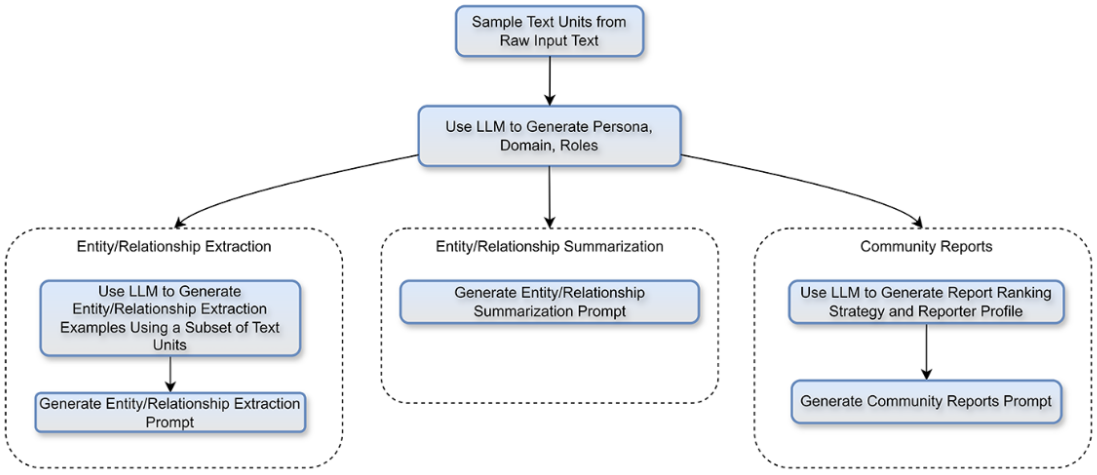

Figura 3.4: Funzionamento del prompt tuning automatico

Il testo in input viene segmentato, alcuni chunks vengono passati al LLM

che sulla base di questi identifica un dominio dei dati e usa questa informazione per generare prompt ottimizzati. L'articolo di Guevara [38] spiega in dettaglio il funzionamento di questa funzionalità, mostrando anche dei test con relativi risultati. Generalmente l'utilizzo di questa tecnica consente di estrarre un maggior numero di entità e relazioni e creare dei sommari migliori. L'applicazione di questo strumento è piuttosto semplice, richiedendo un singolo comando in Bash, come mostrato qui sotto:

```
python -m graphrag.prompt_tune [--root ROOT] /
[--config settings.yaml] [--domain DOMAIN] /
[--method METHOD] [--limit LIMIT] [--language LANGUAGE] /
[--max-tokens MAX_TOKENS] [--chunk-size CHUNK_SIZE] /
[--n-subset-max N_SUBSET_MAX] [--k K] /
[--min-examples-required MIN_EXAMPLES_REQUIRED] /
[--no-entity-types] [--output OUTPUT]
```

1. **Segmentazione in TextUnits:** il testo in input viene suddiviso in TextUnits, ovvero blocchi di testo segmentati e utilizzabili per le tecniche di estrazione del grafo. La dimensione dei chunk è un parametro configurabile, che può essere adattato in base alla lunghezza del testo e alla complessità del task. Aumentare questo parametro può aiutare ad accelerare il processo di indicizzazione, a discapito però della precisione. Questo processo è essenziale per strutturare i dati e garantire che ogni blocco sia associato ai suoi riferimenti e metadati originali, facilitando la tracciabilità e la provenienza delle informazioni. I TextUnits saranno le unità di base su cui verranno effettuate le operazioni di estrazione.

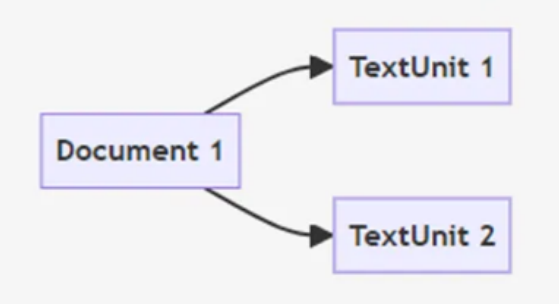

Figura 3.5: Segmentazione in TextUnits

2. **Estrazione di entità e relazioni:** ogni TextUnit viene passato al LLM con apposito prompt, per identificare entità e relazioni presenti nel testo. Per entità si intendono oggetti, persone, luoghi, concetti, eccetera, mentre per relazioni si intendono connessioni tra entità. Ad ogni entità viene assegnato un nome, un tipo e una descrizione prodotta dal modello di linguaggio. Alle relazioni vengono assegnate due entità, una sorgente e una destinazione, e una descrizione. In questo modo vengono generati tanti piccoli grafi, ognuno relativo a un TextUnit.

3. **Aggregazione e creazione dei sommari:** dopo l'estrazione iniziale, i singoli grafi generati per ogni TextUnit vengono combinati in un grafo complessivo. Le entità identiche tra i diversi TextUnit, ossia quelle con lo stesso nome e tipo, vengono fuse. Le descrizioni multiple di ciascuna entità vengono combinate, mantenendo tutte le informazioni rilevanti. In modo analogo, anche le relazioni tra entità che appaiono in diversi TextUnit vengono aggregate, creando una descrizione sintetica per ciascuna connessione. Se presenti, anche le claim (affermazioni verificabili con dettagli temporali o condizionali) vengono identificate e organizzate in modo da facilitare l'interpretazione del grafo finale. In questa fase viene quindi generato un grafo di conoscenza completo che rappresenta l'intero testo in input, con sommari associati a entità, relazioni e claim, generati sempre dal LLM.

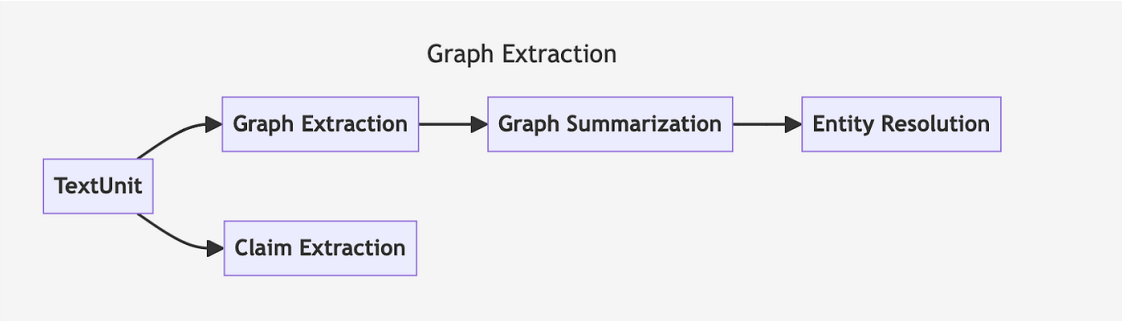

Figura 3.6: Step 2 e 3 della pipeline di indicizzazione

4. **Rilevazione delle comunità:** ottenuto il grafo di conoscenza completo, si procede con l'identificazione delle comunità, ossia gruppi di entità strettamente correlate tra loro. Per fare ciò si utilizzano algoritmi di community detection, che individuano cluster nel grafo. In particolare viene utilizzato

l'algoritmo di Leiden, che effettua una clusterizzazione gerarchica e permette di identificare comunità di entità a differenti livelli di granularità.

5. **Creazione dei report di comunità:** una volta identificate le comunità nel grafo, il modello di linguaggio viene utilizzato per generare un report sintetico per ciascuna comunità. Questi report descrivono le principali entità e relazioni all'interno di ogni comunità, fornendo una panoramica dei temi e dei collegamenti predominanti. L'LLM elabora i dettagli rilevanti e crea un riassunto che evidenzia i concetti chiave e le connessioni più significative all'interno di ciascun cluster.

6. **Generazione degli embedding delle comunità (opzionale):** in questo ultimo passaggio opzionale, si creano degli embedding per ciascuna comunità rilevata. Utilizzando il contenuto del report della comunità, il suo sommario e il titolo del report, si generano rappresentazioni vettoriali che permettono di rappresentare in maniera compatta il contenuto semantico di ciascuna comunità. Questi embedding forniscono una rappresentazione utile per query basate sulla similarità durante la fase di retrieval, facilitando la ricerca e migliorando la precisione nelle risposte.

Figura 3.7: Ultimi step del processo di indicizzazione

La pipeline di indicizzazione descritta rappresenta un processo articolato e completo, finalizzato a trasformare una collezione di documenti in un grafo di conoscenza dettagliato e organizzato. Questa fase di indicizzazione garantisce una rappresentazione dettagliata e comprensibile dei dati in input, inoltre permette di avere una solida struttura per query complesse performanti. Le comunità rilevate e i report sintetici associati rappresentano infatti un livello di organizzazione che facilita la ricerca a diversi livelli di granularità, rendendo il sistema adatto sia per domande specifiche che per analisi di contesto generale.

A questo punto, dopo aver ottenuto il grafo di conoscenza completo e una struttura complessa per facilitare l'aggregazione di informazioni, si può passare alla fase di retrieval.

Pipeline di retrieval

La pipeline di retrieval è la fase successiva alla fase di indicizzazione, e si occupa di effettuare le ricerche e le interrogazioni sul grafo generato dai corpus di dati privati. Come già descritto, Microsoft GraphRAG offre due tipologie di ricerca, una globale e una locale, che si adattano a domande di natura diversa e a task differenti. La differenza fondamentale tra i due approcci sta nel livello di granularità della risposta: la ricerca globale è pensata per rispondere a domande che richiedono una comprensione complessiva del grafo, mentre la ricerca locale è più mirata e adatta a interrogazioni specifiche su entità particolari o relazioni.

Per il progetto di tesi sono state implementate entrambe le tipologie di ricerca, andando a creare due search engine distinti, ognuno con le proprie configurazioni e parametri.

- **Global Search:** questa modalità di ricerca è pensata per domande che richiedono la comprensione dell'intero corpus dei dati in input o di una gran parte di essi. La ricerca globale è utile per task di query-focused summarization, che richie dono una visione d'insieme sui dati o un tipo di ragionamento di alto livello. In Figura 3.8 è mostrato il workflow della ricerca globale, composto da una serie di passaggi complessi che elaborano la query e le informazioni, per restituire una risposta. Il meccanismo di ricerca globale si basa sul map-reduce, che permette di dividere il corpus in batch e di processarli in parallelo, per poi aggregare i risultati e fornire una risposta completa.

    1. Il sistema riceve la query utente e la cronologia conversazionale come input.
    2. I report di comunità vengono mescolati e suddivisi in batch, per essere passati al modello di linguaggio come contesto.
    3. Ogni batch di report viene suddiviso in chunk di dimensioni predefinite e ogni chunk è usato per generare una risposta. La risposta viene generata dal LLM, a cui viene anche chiesto di assegnare un punteggio di

importanza. In questa fase di map vengono dunque generate le risposte parziali, che verranno poi ordinate per punteggio.

4. Le risposte ottenute dai singoli community report, vengono passate al modello di linguaggio ordinate per importanza. Il modello genera a partire da queste una risposta globale finale, tramite un meccanismo di reduce.

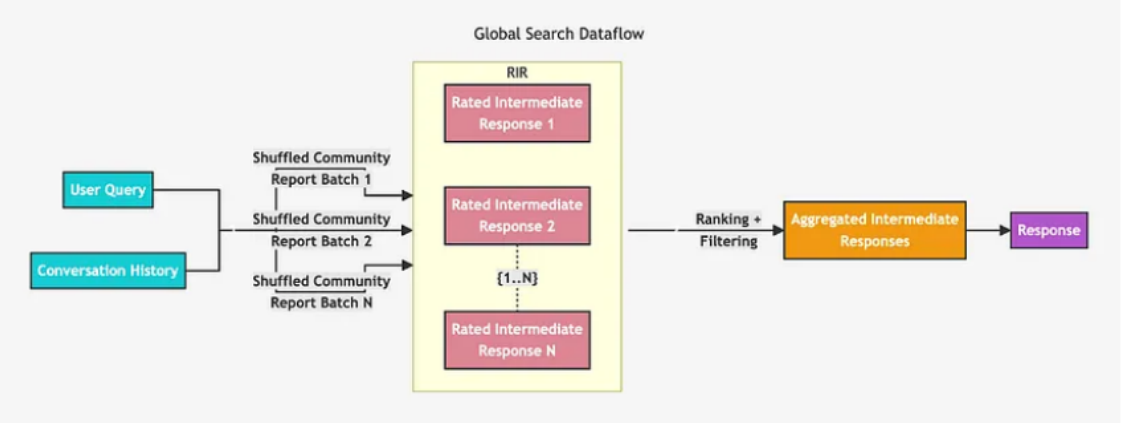

Figura 3.8: Workflow della ricerca globale

Per poter effettuare questo procedimento è stato necessario configurare alcuni parametri e un modello, che in questo caso è stato lo stesso utilizzato nella fase di indicizzazione. Per prima cosa è stato necessario selezionare quali report utilizzare, filtrandoli per livello gerarchico. La scelta è stata quella di utilizzare tutti i livelli gerarchici per mantenere più informazioni rilevanti possibili, senza preoccuparsi del numero di chiamate al LLM. Sono stati impostati dei limiti di token per le fasi di map e di reduce, compatibili col modello scelto e si è scelta la tipologia di risposta desiderata. Anche i prompt delle due fasi sono personalizzabili, ma per i test di questo progetto sono stati utilizzati quelli di default. Questa configurazione della ricerca globale garantisce una risposta che integra tutti i livelli gerarchici del grafo, risultando in una visione d'insieme completa.

- **Local Search:** questa modalità di ricerca è pensata per domande che richiedono informazioni dettagliate su specifiche entità o concetti all'interno del grafo di conoscenza. La ricerca locale è utile per rispondere a domande

mirate e di precisione, come quelle che riguardano singole entità e le loro connessioni dirette o contesti specifici. La ricerca locale è più precisa e dettagliata rispetto alla ricerca globale, integra al suo interno meccanismi classici basati su vettori e meccanismi basati su grafi, per fornire risposte accurate e contestualizzate. In Figura 3.9 è mostrato il workflow della ricerca locale, che si compone di una serie di passaggi che elaborano la query e le informazioni, per restituire una risposta.

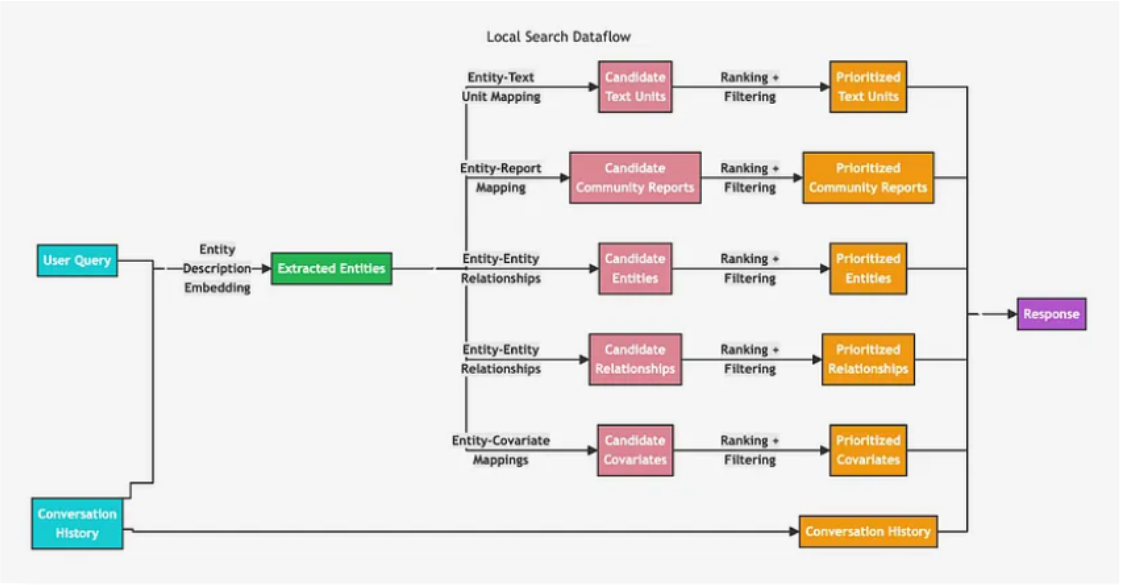

Figura 3.9: Workflow della ricerca locale

1. Il sistema riceve la query utente e la cronologia conversazionale come input.
2. Viene effettuata una ricerca delle entità più rilevanti e semanticamente simili alla query dell'utente, utilizzando un database vettoriale per individuare le connessioni più pertinenti nel grafo.
3. Vengono ricercate le TextUnits collegate alle entità individuate, per estrarre informazioni aggiuntive e contestualizzare la risposta.
4. Per ogni entità identificata, il sistema estrae le relazioni più significative e le entità collegate, creando una rete di informazioni direttamente collegata alla query dell'utente. Stessa cosa viene fatta per le covariate, se presenti.

5. Viene effettuato un mappaggio delle entità per recuperare eventuali report di comunità pertinenti.

6. Tutte le informazioni recuperate ai passi precedenti vengono raggruppate, ordinate e filtrate e fungeranno da contesto per il modello di linguaggio, che genererà la risposta finale.

Per la ricerca locale, oltre al LLM è stato necessario specificare anche un modello di embedding, che in questo caso è stato il Nomic-embed-text. Anche in questo caso sono stati configurati alcuni parametri, molti dei quali sono comuni alla ricerca globale, come i limiti di token e i prompt, mentre altri sono specifici per la ricerca locale e richiedono una attenzione maggiore. In particolare è stato necessario configurare i parametri relativi alla composizione della finestra di contesto, che viene suddivisa in parti dedicate alle unità di testo, alle entità e relazioni del grafo e ai report. Assegnando un peso diverso a ciascuna parte della finestra di contesto, è possibile personalizzare la ricerca, per renderla più legata al testo o al grafo, a seconda delle esigenze. Questa possibilità di utilizzare contesti diversi e di personalizzare la ricerca, rende la ricerca locale particolarmente adatta per domande specifiche e dettagliate, che richiedono una risposta precisa e contestualizzata.

Le due procedure distinte possono essere viste come parte di un unico flusso di lavoro. L'intero processo, che comprende sia la fase di indicizzazione che di retrieval, può essere sintetizzato con gli step mostrati nell'immagine 3.10 Quello nell'immagine è il procedimento generale, che rappresenta fedelmente gli step affrontati dalla pipeline realizzata per questo progetto. Come già citato, è possibile personalizzare fortemente le pipeline di GraphRAG e quindi il worklow in immagine è da intendersi come una idea di base e non sempre come l'effettivo procedimento.

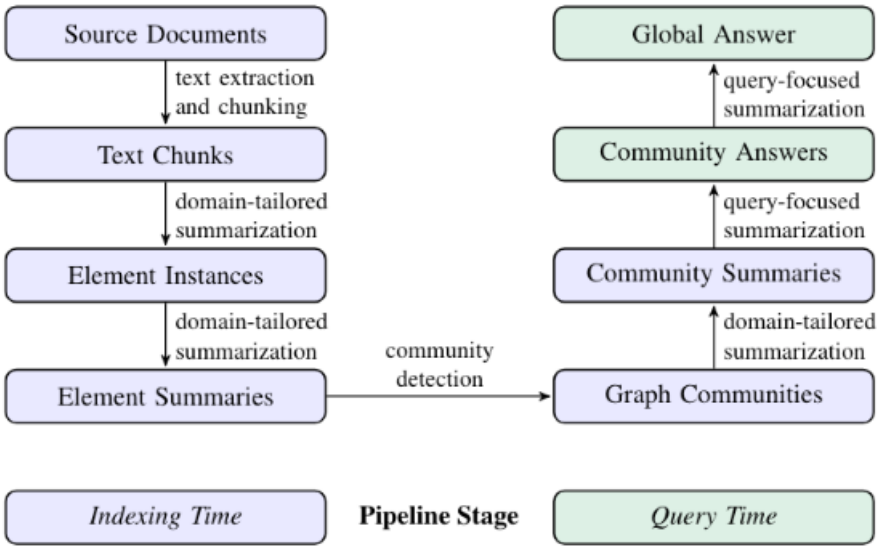

Figura 3.10: Pipeline di GraphRAG completa

3.4 RAG tradizionale

Il sistema di GraphRAG appena descritto deve essere confrontato con un sistema di RAG tradizionale, per valutarne le differenze e le performance. A differenza del GraphRAG, il RAG tradizionale rappresenta un approccio più classico e consolidato, per cui sono stati sviluppati molti strumenti altamente performanti e testati. La scelta è stata dunque quella di utilizzare tecnologie ben note e già utilizzate in altri progetti aziendali, per garantire una comparazione realmente significativa.

Questa pipeline di RAG tradizionale si basa su un processo di indicizzazione dei documenti tramite tecniche di embedding e indicizzazione vettoriale, seguita da una fase di retrieval in cui un modello di linguaggio di grandi dimensioni (LLM) risponde alle query dell'utente basandosi sui chunk indicizzati.

3.4.1 Scelte progettuali e implementative

Ogni fase della pipeline del RAG tradizionale è stata attentamente progettata e configurata per garantire prestazioni ottimali e affidabilità, adattandosi al contesto e ai dati specifici del progetto. Le principali scelte implementative riguardano:

- **Framework e librerie:** per la realizzazione del sistema di RAG tradizionale sono state utilizzate librerie Python open-source, per la gestione dei modelli e dell'indicizzazione vettoriale. La scelta è ricaduta su LangChain, una libreria di intelligenza artificiale tra le più diffuse e performanti, che offre funzionalità avanzate per la gestione di pipeline di retrieval e consente di personalizzare ogni aspetto del processo. LangChain ha moduli specifici per ogni fase della pipeline e non necessita di servizi o librerie esterne, garantendo implementazioni veloci e performanti.

- **Modelli di embedding e di linguaggio:** la pipeline sviluppata utilizza il modello *Multilingual-e5-large* per generare gli embedding e *Llama3.1-8B* come modello di linguaggio. Si è scelto di utilizzare lo stesso LLM del GraphRAG, per garantire una comparazione equa tra i due approcci.

- **Database vettoriale:** è stato utilizzato un database vettoriale basato su Faiss, libreria sviluppata da Meta per il retrieval di dati vettoriali. La scelta di Faiss è stata dettata dalla sua velocità e scalabilità, che lo rendono adatto a grandi moli di dati e a query complesse.

- **Sistema di retrieval:** per rispondere in modo preciso alle query, è stata implementata una catena di RetrievalQA configurata con l'LLM selezionato e il database vettoriale. La catena utilizza i chunk di testo indicizzati come contesto per il modello, permettendo di generare risposte dettagliate e basate sui risultati più rilevanti. La configurazione avanzata del retriever e della tipologia di catena garantisce risposte coerenti e ben contestualizzate, ottimizzando la qualità e la pertinenza delle risposte fornite dal sistema. Sono stati testati valori diversi per il parametro k, che indica il numero di risultati da considerare per la risposta.

3.4.2 Pipeline di RAG tradizionale

Le scelte progettuali e implementative descritte hanno portato alla realizzazione di una pipeline di RAG tradizionale completa, che si compone di una serie di passaggi ben definiti, ognuno dei quali svolge una funzione specifica.

1. **Indicizzazione dei documenti:** il primo passaggio della pipeline consiste nell'indicizzazione dei documenti, che vengono trasformati in embedding vettoriali e indicizzati per la fase di retrieval. Prima della trasformazione in embedding, i documenti vengono suddivisi in chunk di dimensioni predefinite, per facilitare l'elaborazione e l'indicizzazione. I chunk vengono poi passati al modello di embedding, che genera rappresentazioni vettoriali per ciascun chunk. Questi embedding vengono poi indicizzati nel database vettoriale, per essere utilizzati nella fase di recupero. La fase di indicizzazione è essenziale per garantire una rappresentazione efficiente e compatta dei documenti, che permetta di rispondere in modo rapido e preciso alle query dell'utente.

2. **Ricezione della query:** una volta indicizzati i documenti, il sistema è pronto a ricevere le query dell'utente. Le query possono essere di diversa natura e complessità, e il sistema è in grado di gestirle in modo flessibile e personalizzato.

3. **Recupero delle informazioni:** elaborata la query, il sistema passa alla fase di recupero delle informazioni, che consiste nell'interrogazione del database vettoriale per trovare i chunk più rilevanti rispetto alla query. Il database vettoriale restituisce i chunk più simili alla query, che vengono poi passati al modello di linguaggio per generare la risposta.

4. **Generazione della risposta:** l'ultimo passaggio della pipeline è la generazione della risposta, che avviene tramite il modello di linguaggio. Il modello riceve in input, insieme ad un apposito prompt, i chunk indicizzati e genera una risposta dettagliata e contestualizzata, basata sui risultati più rilevanti. La risposta viene poi restituita all'utente, completando il processo.

In figura 3.11 è mostrata la pipeline descritta, in modo schematizzato. La pipeline di RAG tradizionale è stata progettata per garantire prestazioni ottimali e risposte precise e contestualizzate, verrà utilizzata per confrontare le performance durante la fase sperimentale del progetto.

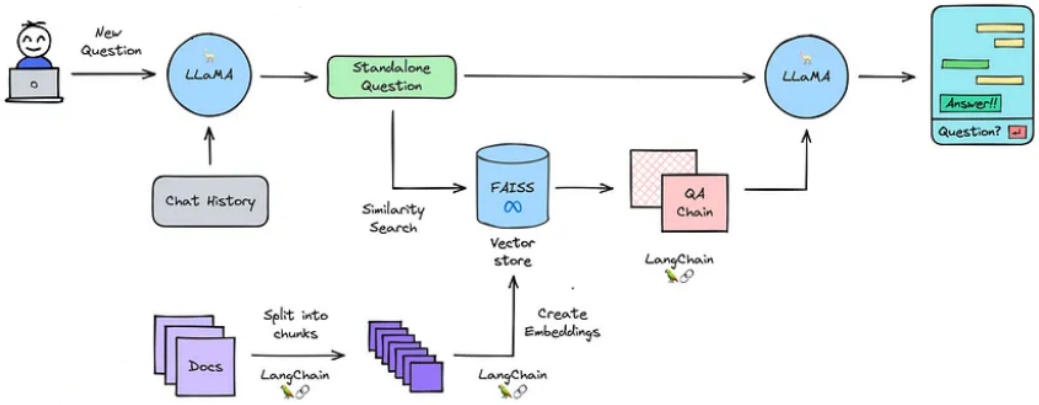

Figura 3.11: Pipeline di RAG tradizionale

3.5 Dati utilizzati

Dopo aver progettato e implementato le pipeline da testare e confrontare, è stato necessario definire un set di dati su cui effettuare le sperimentazioni. I dati utilizzati sono un insieme di nove paper scientifici in formato PDF che trattano il tema dell'analisi di serie temporali con tecniche innovative. Questi articoli rappresentano le più recenti pubblicazioni nel settore e forniscono una panoramica sui nuovi approcci e sulle tecnologie di punta, come modelli basati su LLM e Transformer, applicati alla risoluzione di problemi non legati al linguaggio naturale, ma focalizzati su task specifici per le serie temporali, come l'anomaly detection e il forecasting. Dietro la scelta di questo tipo di dati ci sono due motivazioni principali:

- **Rilevanza aziendale:** il tema dell'analisi di serie temporali è di grande interesse per l'azienda, che sta affrontando anche problemi legati all'analisi di serie temporali. Questi paper possono fungere da base di conoscenza per progetti aziendali futuri, favorendo il trasferimento e la capitalizzazione del know-how.

- **Conoscenza del dominio:** il dominio degli articoli è noto e le soluzioni presentate all'interno delle pubblicazioni sono state studiate. Questo permette

di valutare con maggiore precisione la qualità delle risposte fornite dai sistemi di GraphRAG e di RAG tradizionale, confrontandole con le conoscenze pregresse.

Oltre le motivazioni precedentemente elencate, i paper scientifici sono stati scelti per la loro complessità e eterogeneità, che permettono di testare le pipeline su un problema reale, considerato più indicativo rispetto ad un classico toy dataset.

I paper utilizzati sono stati estratti da archivi scientifici online, come ArXiv e IEEE Xplore. Di seguito è riportata la lista completa dei paper utilizzati, con relativa breve descrizione:

1. **AnomayBERT:** presenta un modello che applica l'architettura BERT, originariamente ideata per il linguaggio naturale, all'ambito delle serie temporali per il rilevamento di anomalie. Grazie alla capacità di BERT di rappresentare relazioni tra sequenze, AnomalyBERT è in grado di identificare pattern anomali all'interno di flussi temporali complessi. L'approccio si distingue per l'efficacia nel trattamento di dati con variazioni sia a breve che a lungo termine, rendendolo adatto a contesti in cui le anomalie possono apparire su differenti scale temporali [39].

2. **Chronos:** rappresenta un framework completo per l'analisi e la previsione su serie temporali, sfruttando l'architettura Transformer. Il sistema è progettato per adattarsi a vari tipi di serie temporali, da quelle finanziarie a quelle industriali, ed è capace di cogliere pattern di dipendenza temporale grazie all'attenzione multi-testa del Transformer. Chronos è ottimizzato per scenari di alta variabilità e complessità, supportando task sia di breve che di lungo periodo in modo flessibile [40].

3. **Foundation Models for Time Series Analysis:** paper che offre una panoramica sui modelli fondamentali per l'analisi delle serie temporali. L'articolo esplora l'evoluzione di questi modelli, evidenziando l'introduzione di LLM e architetture Transformer nell'analisi temporale. Vengono discussi i vantaggi e le sfide di questi approcci avanzati, con un'analisi critica delle capacità di previsione, di adattamento alle serie storiche e delle potenzialità di apprendimento da flussi temporali complessi [41].

4. **LagLLama:** modello che adatta la famiglia dei modelli Llama all'analisi delle serie temporali, con un focus particolare sulla gestione dei ritardi temporali. LagLLAMA riesce a cogliere pattern di dipendenza a vari livelli temporali, migliorando la qualità delle previsioni in contesti di forecasting dove la successione temporale delle osservazioni gioca un ruolo cruciale [42].

5. **ReSTAD:** modello ricorrente, specializzato per il rilevamento di anomalie su serie temporali con variazioni di ampiezza e frequenza. Grazie alla sua struttura a strati (stacked), ReSTAD riesce ad adattarsi a serie temporali in cui l'anomalia può essere camuffata da rumore o da variazioni fisiologiche [43].

6. **TimeGPT:** modello basato su GPT, che applica l'architettura Transformer all'analisi delle serie temporali. TimeGPT rappresenta l'approccio di OpenAI ai task riguardanti le serie temporali [44]

7. **TimeLLM:** presenta un approccio alternativo, che non va a modificare l'architettura per adattarla a dati di tipo serie temporale, bensì applica un meccanismo per trasformare i dati in input da time serie a una rappresentazione comprensibile dal LLM [45].

8. **TimeSFM:** utilizza un'architettura Transformer specializzata per l'attenzione spaziotemporale, progettata per la gestione di serie temporali multidimensionali. Presenta un modello Transformer con architettura decoder-only, specificamente pensato per il forecasting [46].

9. **TranAD:** sfrutta l'architettura Transformer per rilevare anomalie su serie temporali in maniera efficiente e precisa. Grazie all'attenzione multi-head e alle capacità di gestione delle lunghe dipendenze temporali tipiche dei Transformer, TranAD riesce a individuare variazioni anomale in flussi di dati complessi. È particolarmente efficace in contesti dove l'anomalia può essere facilmente confusa con normali fluttuazioni, come nei dati industriali o finanziari [47].

3.6 Valutazione delle performance

Prima di procedere con la fase sperimentale, è necessario definire un meccanismo di valutazione delle performance, per quantificare i risultati che emergeranno nella fase successiva. La valutazione delle performance delle pipeline sviluppate mira a confrontare in modo approfondito il sistema di GraphRAG con la pipeline di RAG tradizionale, basandosi sulle risposte generate dai due sistemi per un set di query. Per garantire un confronto esaustivo, le domande formulate sono di natura eterogenea: alcune sono specifiche e puntuali, mirate a testare la precisione e l'accuratezza delle risposte, mentre altre sono più generiche e richiedono una comprensione globale dei documenti. Questo approccio permette di osservare come i due sistemi rispondano a task di varia complessità e comprensione del contesto. Per questi task non è possibile utilizzare metriche standard come l'accuracy o la precision, in quanto le risposte testuali non si prestano a valutazioni semplici. Per valutare la qualità delle risposte fornite dai due sistemi, sono stati adottati due metodi di giudizio:

- **Valutazione umana:** come prima modalità di valutazione è stata scelta la valutazione umana, che consiste nel sottoporre le risposte generate dai due sistemi a un gruppo di esperti del dominio delle serie temporali. Questa valutazione preliminare è stata utilizzata principalmente durante i test per verificare se i modelli fossero in grado di produrre risposte di qualità, e si è rivelata utile per identificare se le risposte fossero coerenti, sensate e se rispecchiassero adeguatamente le informazioni contenute nei documenti di riferimento. Essendo chi sviluppa il sistema ben informato sui paper usati, questa valutazione intuitiva e qualitativa consente di rilevare aspetti delle risposte non facilmente quantificabili, come la varietà lessicale e la pertinenza tematica. La valutazione umana, pur limitata, è preziosa perché permette di osservare con occhio critico la coerenza generale delle risposte e identificare possibili margini di miglioramento per ciascun sistema.
- **LLM as a Judge:** metodo che impiega un modello di linguaggio avanzato, in questo caso il modello *Llama3.1-70B*, per giudicare la qualità delle risposte delle due pipeline in base a criteri specifici. Il modello,

grazie alla sua capacità di valutare il linguaggio e confrontare risposte complesse, è stato utilizzato in due modalità distinte, descritte di seguito.

- **Con golden dataset:** nel primo approccio di LLM as a Judge, il modello è utilizzato in modalità tradizionale: le risposte dei sistemi vengono confrontate con un set di risposte di riferimento, chiamato appunto golden dataset. Con questo approccio ogni risposta è valutata singolarmente, inviando tramite apposito prompt la domanda, la risposta generata e la risposta di riferimento al modello giudice, che ha il compito di attribuire un punteggio a metriche prestabilite, su una scala da 1 a 5. Le metriche selezionate sono:
  - **Correctness:** misura la correttezza fattuale della risposta rispetto ai documenti di riferimento, verificando l'aderenza delle informazioni alla conoscenza del dominio;
  - **Completeness:** valuta se la risposta è completa rispetto alla richiesta dell'utente e contiene tutti gli elementi necessari per rispondere alla domanda senza omissioni rilevanti;
  - **Relevance:** giudica quanto la risposta sia pertinente alla query posta e adeguata al contesto.

  Queste metriche, selezionate per valutare la qualità delle risposte in modo esaustivo, coprono aspetti diversi e complementari. Questo approccio è applicabile solo in presenza di un golden dataset, che è stato generato appositamente per il progetto. Tuttavia, dato che la costruzione del golden dataset comporta margini di errore e potrebbe non essere esaustivamente accurata, si è deciso di utilizzare anche un secondo metodo di valutazione per una verifica più robusta.
- **Senza golden dataset:** il secondo approccio di LLM as a Judge prevede una valutazione senza golden dataset. In questo caso, Llama3.1-70B confronta le risposte dei due sistemi tra loro direttamente, senza fare riferimento a risposte predefinite, ma valutando quale delle due risposte sia più efficace in termini di determinate metriche. Per ciascuna coppia di risposte (una per ciascun sistema), viene chiesto al modello Llama3.1-70B di scegliere la migliore rispetto a quattro metriche *user-centric*, mirate a valutare l'esperienza e

l'usabilità della risposta dal punto di vista dell'utente:

- **Comprehensiveness:** misura quanto la risposta sia esaustiva nel rispondere alla query, ovvero se offre una panoramica completa e soddisfacente;
- **Diversity:** valuta la varietà delle informazioni incluse nella risposta, ossia se essa integra prospettive o dettagli diversi per una visione ampia dell'argomento;
- **Directness:** indica la chiarezza e la concisione della risposta, ossia se risponde in modo diretto alla query;
- **Empowerment:** misura quanto la risposta sia in grado di fornire all'utente le informazioni necessarie per comprendere l'argomento e proseguire la ricerca in modo autonomo.

Questo metodo di valutazione è particolarmente utile per evidenziare differenze qualitative tra i due sistemi, poiché considera non solo la correttezza della risposta, ma anche la sua utilità dal punto di vista dell'utente finale, rispondendo quindi anche a criteri più orientati all'usabilità.

In sintesi, la valutazione delle performance si basa su una combinazione di giudizi umani e valutazioni automatiche, fornendo una panoramica completa della qualità delle risposte generate. La scelta di integrare più modalità di valutazione nasce dalla consapevolezza che nessun singolo metodo fornisce risultati perfetti; il golden dataset, per esempio, potrebbe contenere errori o non rispecchiare esattamente la totalità delle risposte possibili. Questo approccio permette quindi di compensare i limiti di ciascuna metodologia e di ottenere un risultato complessivo più robusto, offrendo un quadro affidabile sui punti di forza e di debolezza delle pipeline GraphRAG e RAG tradizionale.

3.7 Considerazioni finali

Il design della sperimentazione descritto in questo capitolo è stato progettato per rispondere all'obiettivo principale del progetto: confrontare le performance e la versatilità di due approcci di Retrieval Augmented Generation

(RAG), GraphRAG e RAG tradizionale. L'implementazione del sistema si basa su scelte architetturali e metodologiche che sfruttano strumenti di intelligenza artificiale avanzati e tecnologie all'avanguardia offerte dall'ambiente di *E4-Analytics*. La pipeline di GraphRAG è stata sviluppata per testare come l'integrazione di knowledge graph possa migliorare la qualità e la coerenza delle risposte nei task di retrieval e generazione, permettendo di gestire dati complessi e di fornire risposte più ricche di contesto. Parallelamente, la pipeline di RAG tradizionale, implementata con tecnologie consolidate, offre un efficace sistema di retrieval basato su vettori che consente di confrontare l'approccio a grafo con uno più tradizionale. Infine, la definizione di un meccanismo di valutazione, strutturato sia in base a giudizi umani sia in termini di metriche automatizzate attraverso LLM as a Judge, rappresenta un approccio bilanciato per misurare in modo accurato la qualità delle risposte generate. La scelta di combinare metodi con e senza golden dataset, oltre a diverse metriche per la valutazione user-centric, contribuisce ad aumentare la robustezza dei risultati, riducendo l'influenza dei limiti specifici di ciascun metodo. In sintesi, le scelte di design e metodologia adottate pongono le basi per un confronto solido e affidabile tra GraphRAG e RAG tradizionale, che sarà oggetto di analisi nel prossimo capitolo.

Capitolo 4

Infrastruttura di calcolo e ambiente di sviluppo

Il progetto è stato sviluppato all'interno dell'ambiente di sviluppo avanzato di *E4-Analytics*, azienda specializzata nella realizzazione di soluzioni di intelligenza artificiale per svariati ambiti applicativi. Negli anni, *E4-Analytics* ha costruito una gamma di strumenti e piattaforme che supportano lo sviluppo di sistemi di intelligenza artificiale, integrando le migliori pratiche per modularità, scalabilità e gestione di modelli di linguaggio di grandi dimensioni (LLM).

Tutti i progetti aziendali seguono un insieme di principi architetturali comuni e condividono l'infrastruttura, garantendo consistenza e efficienza nello sviluppo. In questa sezione, verranno illustrati i principi architetturali e i principali prodotti di *E4 Analytics* che hanno supportato lo sviluppo di questo progetto, fornendo la base tecnologica e operativa per l'implementazione del sistema di Retrieval Augmented Generation.

4.1 Principi architetturali

Affrontando progetti di intelligenza artificiale, è necessario compiere scelte architetturali che garantiscano la buona riuscita del progetto e la scalabilità

del sistema. È altrettanto importante garantire la modularità dell'architettura, per consentire una facile sostituzione o aggiunta di componenti senza riprogettare l'intero sistema. *E4-Analytics* adotta principi architetturali che promuovono la modularità e la scalabilità dei sistemi, ma anche un'attenzione particolare alla gestione sicura dei dati e alla riservatezza degli stessi. Per garantire che i dati aziendali e i dati degli utenti siano trattati in modo sicuro e conforme alle normative sulla privacy, *E4-Analytics* evita di fare affidamento su modelli esterni esposti tramite API pubbliche, privilegiando soluzioni interne e sicure.

I principi architetturali alla base del progetto sono descritti in dettaglio di seguito.

4.1.1 Architettura a microservizi

Negli ultimi anni, il panorama dello sviluppo software ha subito un'evoluzione significativa, spostandosi da architetture monolitiche tradizionali a architetture a microservizi. Questo cambiamento è stato dettatato dalla necessità di rispondere a una domanda crescente di servizi digitali, che richiedono applicazioni scalabili, resilienti e flessibili.

Con il termine servizio si intende delle risorse software che eseguono un'attività specifica e che possono essere utilizzate da altre applicazioni, esse possono essere risorse di calcolo, di storage o di svariate altre tipologie. Dall'introduzione del concetto di servizio, si è passati a quello di microservizio, che ne rappresenta un'evoluzione. Un microservizio è un servizio software "piccolo" e autonomo, che esegue un'attività specifica e che comunica con altri servizi tramite API. I microservizi sono progettati per essere indipendenti e scalabili, vengono eseguiti come processi distinti e si prestano bene a essere distribuiti su più macchine o container. In un'architettura a microservizi, un'applicazione è composta da un insieme di microservizi, ognuno dei quali rappresenta un'unità funzionale indipendente e autonoma.

Vantaggi dell'architettura a microservizi

L'architettura a microservizi presenta diversi vantaggi rispetto a un'architettura monolitica, tra cui:

- **Modularità:** questa architettura facilita la suddivisione dell'applicazione in moduli distinti. Ogni modulo rappresenta un microservizio con limitate responsabilità e ciò semplifica gli aggiornamenti e le modifiche, consentendo di sostituire o aggiungere microservizi senza impatti sugli altri.
- **Scalabilità:** i microservizi possono essere scalati in modo indipendente, consentendo di allocare risorse solo ai servizi che ne hanno bisogno. Questo permette di ottimizzare l'utilizzo delle risorse e di garantire prestazioni elevate anche in situazioni di carico elevato.
- **Resilienza:** un'architettura a microservizi è più resiliente rispetto a un'architettura monolitica, poiché un errore in un microservizio non compromette l'intera applicazione.
- **Indipendenza tecnologica:** i microservizi possono essere sviluppati con tecnologie diverse, consentendo di utilizzare il linguaggio di programmazione o il framework più adatto per ogni servizio.

Questo tipo di architettura è particolarmente adatto per progetti di intelligenza artificiale, in quanto consente di sviluppare e gestire modelli, database e servizi vari in modo indipendente, modulare e scalabile. Per questo motivo, *E4-Analytics* adotta un'architettura a microservizi per tutti i progetti di intelligenza artificiale, garantendo flessibilità, scalabilità e sicurezza.

Containerizzazione

La containerizzazione è una tecnologia che consente di eseguire applicazioni in un ambiente isolato, chiamato container, che contiene tutti i componenti necessari per l'esecuzione dell'applicazione. Un container è un'unità software leggera e portatile, che include il codice dell'applicazione, le librerie,

le dipendenze e le configurazioni necessarie per l'esecuzione. I container sono indipendenti dal sistema operativo e possono essere eseguiti su qualsiasi piattaforma che supporti la containerizzazione.

La containerizzazione si presta bene all'architettura a microservizi, poiché consente di distribuire e gestire facilmente i microservizi in ambienti isolati e indipendenti. Inoltre, i container sono scalabili e possono essere facilmente replicati su più macchine, consentendo di gestire carichi di lavoro elevati in modo efficiente. In applicazioni con architettura a microservizi, la containerizzazione è una componente essenziale, poiché semplifica la distribuzione, la gestione e la scalabilità dei servizi.

Docker Docker è la piattaforma più diffusa per la containerizzazione, grazie alla sua grande flessibilità e alla sua facilità d'uso . Rappresenta uno standard de facto per la containerizzazione e offre un'ampia gamma di strumenti e servizi per la creazione, la distribuzione e la gestione di container. Consente agli sviluppatori di aggirare i problemi di compatibilità tra ambienti di sviluppo e produzione, garantendo che le applicazioni funzionino correttamente su qualsiasi piattaforma.

I componenti principali di Docker sono:

- **Docker Engine:** il cuore di Docker, responsabile della gestione dei container. Consiste di tre elementi principali:
  - *Docker Daemon:* processo in esecuzione sul sistema host, che ascolta le richieste API e gestisce la creazione, l'esecuzione e la distruzione dei container.
  - *Client Docker:* interfaccia da riga di comando (CLI) che consente agli utenti di interagire con il Docker Daemon. Questo client è progettato per essere leggero ed eseguibile su qualsiasi macchina che possa comunicare con il Docker Daemon.
  - *API RESTful:* API che consente al Docker Daemon di comunicare con il Client Docker e con altri servizi. Consente l'automatizzazione di molte operazioni di gestione dei container.

- **Docker Image:** un'immagine Docker è un file di sola lettura che contiene il codice dell'applicazione, le librerie, le dipendenze e le configurazioni necessarie per eseguire un container. Le immagini sono le componenti fondamentali per la creazione di container. Vengono create a partire da un file di configurazione chiamato Dockerfile, che definisce i passaggi necessari per la creazione dell'immagine. Le immagini Docker possono essere distribuite e condivise tramite Docker Hub, un registro pubblico di immagini Docker, o tramite registri privati. Hanno una struttura a layer, che consente di riutilizzare le parti comuni tra le immagini e di ridurre lo spazio di archiviazione necessario.
- **Docker Container:** un container Docker è un'istanza in esecuzione di un'immagine Docker. Un container è un ambiente isolato che ha ereditato le proprietà dell'immagine da cui è stato creato.

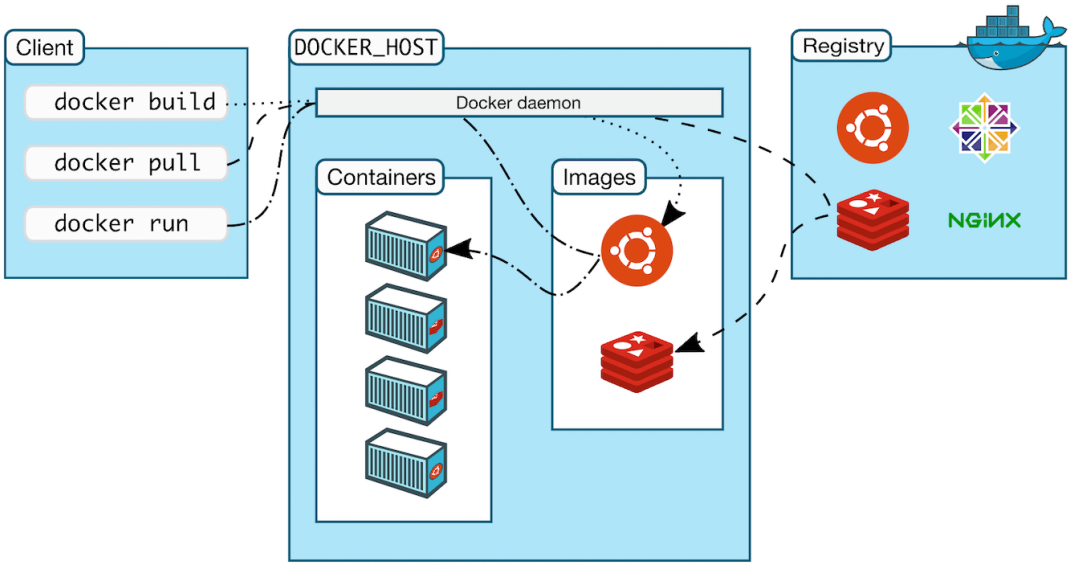

Figura 4.1: Architettura di Docker

Docker ha rivoluzionato il modo di gestire i microservizi poiché, grazie alla sua architettura basata su container, consente di mantenere un elevato grado di isolamento tra servizi indipendenti [48]. La containerizzazione con Docker è dunque uno strumento essenziale per l'architettura a microservizi e molte soluzioni di intelligenza artificiale si basano su Docker per la distribuzione e

la gestione dei servizi. La criticità fondamentale legata all'uso di Docker è la complessità della gestione dei container, che richiede competenze specifiche e tecnologie aggiuntive per gestione e automazione dei container in ambienti di produzione.

Orchestrazione

In ambienti complessi che adottano un'architettura a microservizi, con centinaia o migliaia di container in esecuzione, la gestione manuale dei container diventa impraticabile. L'orchestrazione è una tecnologia che cnsente di automatizzare la distribuzione, la gestione e la scalabilità dei container, semplificando la gestione di ambienti complessi e garantendo la disponibilità e l'affidabilità dei servizi. Questa tecnologia consente di definire regole e politiche per la distribuzione, il bilanciamento del carico, la scalabilità e la sicurezza dei container, consentendo di gestire facilmente un gran numero di container in ambienti di produzione. L'orchestrazione risolve anche sfide legate alla resilienza e alla disponibilità dei servizi, consentendo di ripristinare automaticamente i container in caso di guasti o errori, garantendo la continuità del servizio.

Kubernetes Kubernetes è una piattaforma open-source sviluppata inizialmente da Google e ora gestita dalla Cloud Native Computing Foundation (CNCF), per l'orchestrazione e gestione di container su larga scala. Viene considerato uno standard per l'orchestrazione, grazie alla sua capacità di gestire, scalare e monitorare container distribuiti su cluster di nodi. Il suo cuore è il modello di astrazione, che consente la definizione di risorse e servizi in modo dichiarativo, attraverso file di configurazione chiamati manifesti. Questi file definiscono lo stato desiderato del sistema, descrivendo come i container devono essere distribuiti. Kubernetes si occupa di garantire che lo stato attuale del sistema corrisponda allo stato desiderato, gestendo l'intero ciclo di vita del container.

Il funzionamento di Kubernetes è piuttosto complesso e richiede una conoscenza approfondita dei concetti e delle funzionalità offerte dalla piattaforma.

- **Pod:** il pod è l'unità di base di Kubernetes, che rappresenta un gruppo di uno o più container che condividono risorse e spazio di rete. Ogni pod è isolato dagli altri e viene gestito da Kubernetes come un'unità atomica. I pod sono progettati per essere effimeri e possono essere creati, distrutti e replicati in modo dinamico. In caso di fallimento di un pod, Kubernetes si occupa di ripristinarlo automaticamente, garantendo la continuità del servizio e un elevato livello di disponibilità. Tipicamente un pod contiene un solo container, ma è possibile avere più container all'interno di un pod, se necessario.
- **Deployment:** il deployment è un controller di Kubernetes che gestisce la distribuzione e l'aggiornamento dei pod. Attraverso un deployment, è possibile dichiarare lo stato desiderato del sistema, specificando il numero di repliche dei pod, le risorse richieste, le immagini dei container, le politiche di aggiornamento e molti altri parametri. L'uso di un deployment semplifica la gestione dei pod, consentendo di definire regole e politiche per la distribuzione e l'aggiornamento dei container, che Kubernetes si occuperà di applicare. Alla creazione di un deployment, Kubernetes crea un set di repliche dei pod, garantendo che il numero di repliche corrisponda allo stato desiderato (ReplicaSet).
- **Service:** il service è un'astrazione di Kubernetes che consente di esporre un gruppo di pod come un servizio di rete. Con un service, è possibile definire un punto di accesso stabile per un insieme di pod, consentendo ad altri servizi di comunicare con i pod attraverso un nome di dominio o un indirizzo IP. Vengono generalmente utilizzati service per esporre servizi interni o in certi casi per esporre servizi esterni. I service possono essere di diversi tipi, tra cui ClusterIP, NodePort, LoadBalancer e ExternalName, a seconda delle esigenze di rete.

L'architettura di Kubernetes permette di scalare automaticamente i servizi in base al carico, migliorando l'efficienza delle risorse e garantendo elevata disponibilità. Kubernetes rappresenta dunque una soluzione completa per l'orchestrazione di container, che anche in ambito di intelligenza artificiale è ampiamente utilizzata.

4.1.2 LLM on premise

Come speigato nella sezione 2.2.6, quando si affrontano progetti di intelligenza artificiale, è possibile scegliere tra due approcci principali: LLM on cloud e LLM on premise. *E4-Analytics* ha scelto di adottare un approccio on premise per i progetti di intelligenza artificiale, per garantire la sicurezza e la riservatezza dei dati aziendali e dei dati degli utenti. L'approccio on premise consente di mantenere il controllo totale sui dati e di garantire che siano trattati in modo sicuro e conforme alle politiche aziendali e alle normative sulla privacy.

L'utilizzo di LLM on premise richiede l'implementazione di un'infrastruttura dedicata, che possa supportare l'esecuzione di modelli di linguaggio di grandi dimensioni in modo efficiente e scalabile. *E4-Analytics* espone i modelli su un'infrastruttura dedicata, garantendo tempi di risposta e performance ottimizzati, grazie a risorse hardware di calcolo elevate e configurate per l'elaborazione di modelli su larga scala.

Anche il progetto di tesi si basa su un'infrastruttura on premise, che consente di eseguire modelli di linguaggio ospitati internamente. Nel caso del presente lavoro, questa scelta architetturale è stata particolarmente importante dal punto di vista dei costi, permettendo di effettuare sperimentazioni e test su larga scala senza incorrere in costi eccessivi.

4.2 Prodotti di E4-Analytics

Per poter comprendere al meglio l'ambiente di sviluppo in cui è stato condotto il progetto, è necessario descrivere i principali prodotti di *E4-Analytics*. Verranno descritti nel dettaglio i principali prodotti, al fine di evidenziare quella che è la base tecnologica su cui si è sviluppato il progetto di tesi.

4.2.1 GAIA

Uno dei prodotti principali prodotti di *E4-Analytics* è GAIA (Generative Artificial Intelligence Appliance) [49], una piattaforma progettata per consentire

l'acccesso a modelli di intelligenza artificiale, specializzati per il dominio aziendale del cliente. Questo strumento è pensato per i professionisti in ambito di intelligenza artificiale, che necessitano di un ambiente di sviluppo avanzato e di risorse di calcolo elevate per eseguire modelli di linguaggio di grandi dimensioni. Questa soluzione combina le performance elevate di un server multi-GPU con un'infrastruttura software containerizzata. L'architettura di GAIA ha due moduli principali, come mostrato in Figura 4.2:

- **Infrastruttura Kaptain a singolo nodo:** basata su Kubernetes, che fornisce servizi di orchestrazione e gestione dei container ad alte prestazioni.
- **Private-LLM:** modelli di linguaggio di grandi dimensioni ospitati internamente e resi specifici per il dominio aziendale del cliente.

Figura 4.2: Architettura di GAIA

GAIA è progettata per supportare l'intero ciclo di vita dei progetti di intelligenza artificiale, offrendo un ambiente di calcolo multiutente interattivo, scalabile e sicuro. Si basa sulla tecnologia del Notebook Jupyter, supportando tutte le principlai librerie di machine learning e deep learning, tra cui TensorFlow, PyTorch, Scikit-learn e molti altri.

Il sistema di storage di GAIA è basato su MinIO, un sistema di storage ad alte prestazioni che consente di archiviare e gestire grandi quantità di dati in modo efficiente, ideale per gestire Data-Lake, e un batch scheduler per il lancio di job complessi.

GAIA è in grado di gestire carichi di lavoro complessi, garantendo prestazioni elevate in termini di velocità di calcolo, grazie all'uso di CPU multicore fino a 2TB di RAM e GPU NVIDIA.

Come tutti i prodotti di *E4-Analytics*, si basa sull'utilizzo di Private-LLM, cioè di modelli di linguaggio di grandi dimensioni ospitati internamente e resi specifici per il dominio aziendale del cliente. Questo tipo di prodotto garantisce l'accesso a un modello performante, in una infrastruttura ad alte prestazioni, garantendo inoltre i più alti standard di sicurezza e privacy. In Figura 4.3 è mostrata l'interfaccia di GAIA, con la spiegazione delle principali funzionalità offerte dalla piattaforma.

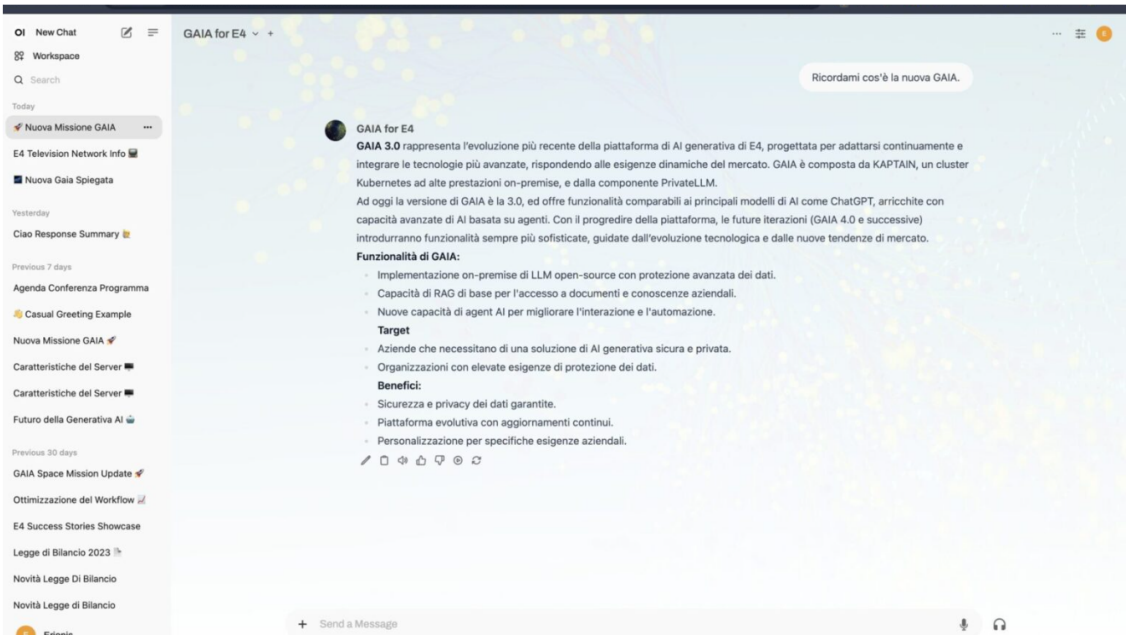

Figura 4.3: Interfaccia di GAIA e spiegazione delle funzionalità principali

4.2.2 URANIA

URANIA [50] rappresenta un'altra soluzione innovativa di *E4-Analytics*, pensata per la Data Analysis su larga scala e per il Machine Learning. Il focus principale di URANIA è la gestione di grandi quantità di dati e la realizzazione di vari tipi di attività con essi, mantenendo le performance elevate e garantendo la sicurezza e la privacy dei dati.

URANIA è una piattaforma end-to-end che utilizza Kubernetes per l'orchetsrazione dei container, offrendo una infrastruttura modulare e flessibile, che può adattarsi a diverse applicazioni e carichi di lavoro. Fornisce servizi di calcolo interattivi tramite Jupyter Notebook, includendo una serie di tool avanzati per l'analisi dati distribuita e supportando tutti i principali framework di machine learning e deep learning.

Figura 4.4: Struttura base di URANIA

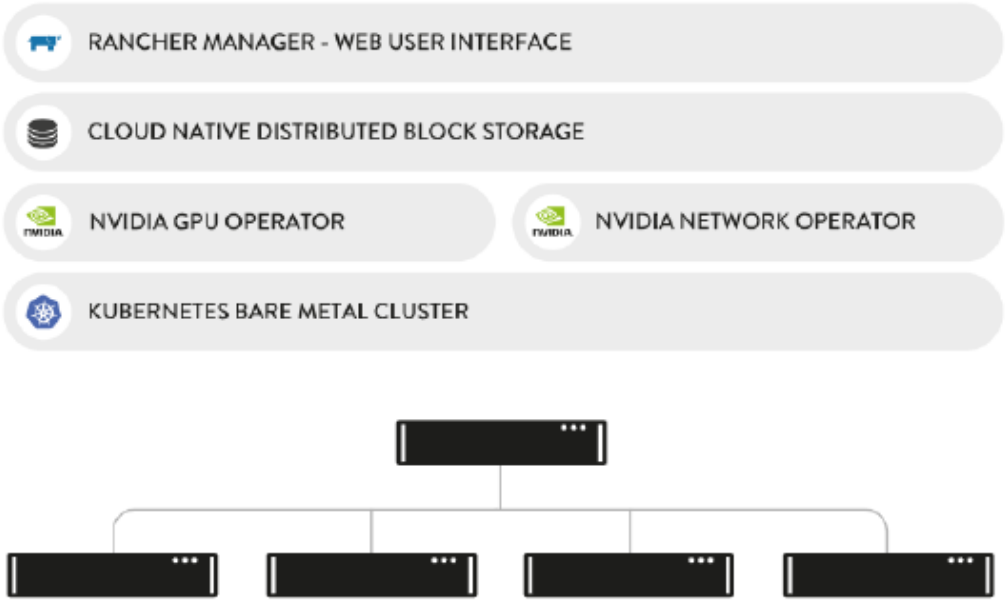

Figura 4.5: Infrastruttura Kaptain di URANIA

Al suo interno integra un batch scheduler per esecuzione di job distribuiti su più nodi, garantendo grande efficienza anche in scenari multiutente. La sua infrastruttura è denominata Kaptain multinodo, è mostrata in Figura 4.5 e offre servizi di orchestrazione di container, basandosi sulla piattaforma E4DS-PLATFORM, pensata per abilitare un set di funzionalità avanzate per la data

science. Nella Figura 4.6 sono mostrate le principali componenti, evidenziando gli strumenti principali messi a disposizione da E4DS-PLATFORM.

URANIA è la soluzione ideale per ospitare l'intero life cycle dei più recenti modelli open-source di Generative AI e può integrare in modo trasparente i servizi di Large Language Model (LLM) locali o remoti direttamente negli ambienti Jupyter notebook per generare nuovo codice AI o accelerare l'operatività del data scientist. Il grande vantaggio di URANIA è che il suo utilizzo permette ai data scientist e specialisti di analisi dati di concentrarsi solo sulla propria disciplina, senza doversi preoccupare della gestione dell'infrastruttura, della configurazione degli ambienti di sviluppo e dell'armonizzazione tra compinenti hardware e software.

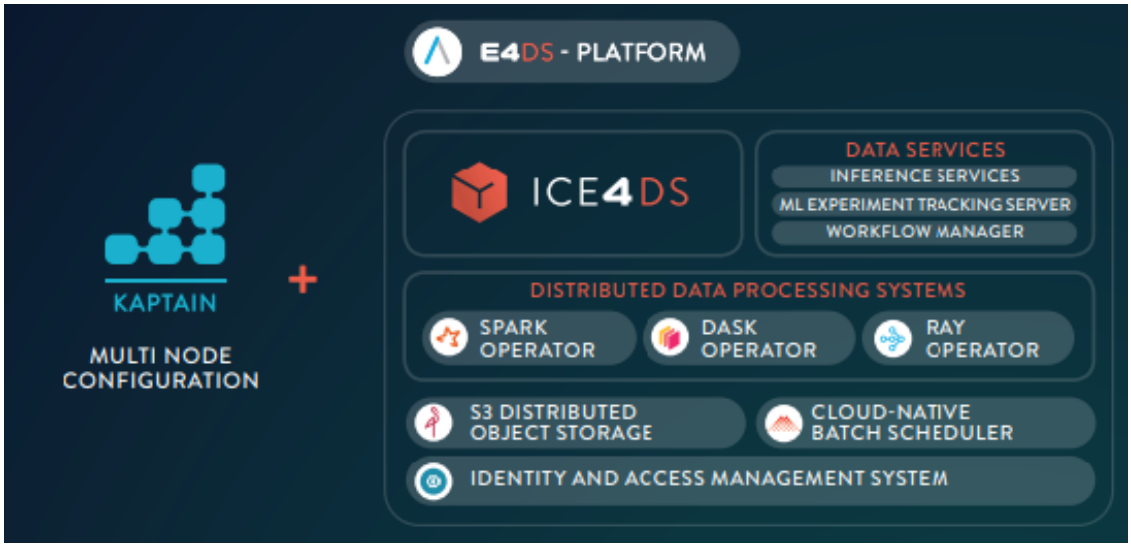

Figura 4.6: Architettura completa di URANIA

4.2.3 Ice4AI

Ice4AI non è come i precedenti un prodotto venduto da *E4-Analytics*, ma è un ambiente virtuale per lo sviluppo di progetti interni. Più nel dettaglio è una piattaforma avanzata pensata per progetti di intelligenza artificiale. Non è esposto esternamente, ma accessibile solo dall'interno del cluster aziendale.

E4-AI Platform è uno stack software basato su Kubernetes che incorpora i componenti necessari per un flusso di lavoro di un progetto di intelligenza

artificiale, come mostrato in Figura 4.7. Lo stack include al suo interno anche Ice4AI, che è basato su Jupyter Notebook e offre un ambiente di sviluppo versatile. Gli utenti di questa piattaforma hanno a disposizione molti ambienti virtuali preconfigurati, integrando al loro interno le principali librerie e framework di machine learning e deep learning, come TensorFlow, PyTorch e MXNet, per l'analisi dati e gestione di big data, come Dask e Ray, e molte altre tecnologie.

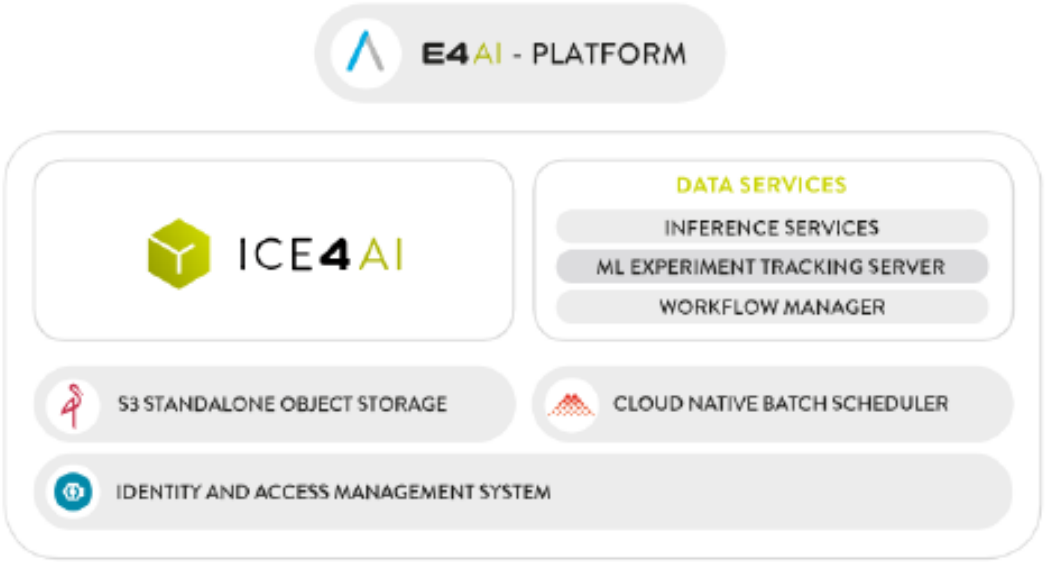

Figura 4.7: Architettura di E4AI-Platform

Il punto di forza principale di Ice4AI è la capacità di sfruttare il GPU Computing a prestazioni elevate, accelerando significativamente i processi di calcolo e l'addestramento di modelli, e rendendo l'analisi dati più efficiente e veloce. L'idea di base è quella di dedicare risorse ad ogni utente, in modo flessibile ed adattabile alle esigenze del progetto, per garantire sempre le migliori performance e un utilizzo efficiente in termini di risorse. Un'altra caratteristica molto importante è la possibilità di integrare container NVIDIA NGC, espandendo le capacità e offrento ulteriori strumenti.

4.3 Cluster aziendale per l'esposizione di modelli

E4-Analytics ha sviluppato un cluster aziendale dedicato per l'esposizione di modelli di intelligenza artificiale, seguendo i principi architetturali descritti in precedenza. Basato su Kubernetes e su prodotti interni come URANIA e VURANIA, il cluster permette una gestione efficiente e scalabile di una varietà di modelli, tra cui LLM, embedder e altri strumenti di AI. Il cluster è configurato per operare su due ambienti distinti:

- **URANIA:** utilizzato in produzione.
- **VURANIA:** versione di URANIA realizzata con macchine virtuali, è pensato per il development e il testing.

La gestione e orchestrazione dei container può essere effettuata tramite Rancher, semplificando il deployment e monitoraggio dei microservizi distribuiti. Il cluster è configurato per utilizzzare una GPU NVIDIA A100 80GB, un sistema di storage basato su NFS di 14 TB e una architettura composta da un master e due nodi worker.

Sono ospitati diversi endpoint per l'inferenza di modelli AI, con soluzioni per LLM, embedder e modelli specifici di vario tipo. Questi modelli sono esposti utilizzando due tecnologie principali:

- **vLLM:** vLLM è una soluzione specializzata per l'inferenza di LLM, in particolare modelli di grandi dimensioni che richiedono un'elevata capacità di calcolo e un'infrastruttura ottimizzata per la latenza ridotta. vLLM supporta un'interfaccia API compatibile con OpenAI, consentendo una facile integrazione con le applicazioni già esistenti che utilizzano lo standard API OpenAI, ma mantenendo l'inferenza on-premise. Questo approccio è ideale per applicazioni in produzione che richiedono risposte rapide e scalabili, minimizzando la latenza e migliorando la capacità di risposta sotto carichi elevati. L'architettura di vLLM gestisce in modo intelligente la suddivisione e l'allocazione della memoria tra i modelli caricati, ottimizzando l'uso delle risorse GPU per supportare molteplici

richieste simultanee senza compromettere la qualità dell'inferenza. Inoltre, vLLM consente la gestione di contesti di lunghezza variabile, permettendo di ottimizzare le risposte del modello in base alla lunghezza delle richieste dell'utente.

- **Ollama:** Ollama è una tecnologia di inferenza leggera, spesso utilizzata spesso utilizzata per la gestione di modelli di embedding o modelli leggeri che generano rappresentazioni vettoriali di testo e altri dati. Ollama è integrata con il cluster per gestire carichi di lavoro che non richiedono le risorse di un LLM completo, migliorando l'efficienza complessiva del sistema e riducendo il tempo di risposta per task meno intensivi. Questa tecnologia viene utilizzata soprattutto in fase di sviluppo e testing.

4.3.1 Modelli esposti

Il cluster aziendale espone una varietà di modelli AI adatti per task diversi, dal linguaggio naturale alla creazione di embedding multilingue, alcuni dei quali sono stati utilizzati per il progetto in questione. Di seguito sono descritti i modelli principali selezionati per il progetto.

Llama3.1-8B

Llama3.1-8B è un modello di linguaggio di grandi dimensioni, precisamente con 8 miliardi di parametri, sviluppato per gestire task di generazione di testo e di comprensione del linguaggio naturale. Basato sui principi di modellazione avanzati esposti da Meta AI, fa parte della famiglia dei modelli Llama ed è una versione che deriva dal modello Llama3, presentato in 'The Llama 3 Herd of Models' [51]. La raccolta Meta Llama 3.1 di modelli linguistici di grandi dimensioni (LLM) multilingue è una raccolta di modelli pre-addestrati e di tipo instruction fine-tuning di dimensioni differenti (8B in questo caso). Questi modelli sono di tipo only-decoder, ovvero sono modelli che generano testo a partire da un input testuale, senza la necessità di un contesto aggiuntivo. Questa versione del modello, esposto con vLLM, è stato utilizzato per la maggior parte degli step del progetto.

Llama3.1-70B

Llama3.1-70B, anch'esso parte della serie Llama di Meta AI, rappresenta una versione avanzata con 70 miliardi di parametri. Grazie alla sua maggiore complessità e capacità di apprendimento, è in grado di gestire task più complessi e di generare testo di alta qualità. Anche questo modello è esposto tramite vLLM, ma è stato utilizzato solo in alcune fasi del progetto, in particolare per task che richiedevano una maggiore complessità.

Multilingual-e5-large-instruct

Il modello Multilingual-e5-large-instruct è stato progettato per l'embedding multilingue, supportando applicazioni in ricerca e categorizzazione semantica di testi in diverse lingue. Ollama gestisce il modello in modo efficiente, riducendo il carico computazionale senza compromettere la qualità delle rappresentazioni vettoriali.

Nomic-embed-text

Nomic-embed-text è un modello di embedding di testo, progettato per generare rappresentazioni vettoriali di testo in modo efficiente e scalabile [52]. Questo modello è stato utilizzato tramite Ollama per task di embedding e di ricerca semantica, in particolare per la fase di retrieval.

Paraphrase-multilingual-MiniLM-L12-v2

Il modello Paraphrase-multilingual-MiniLM-L12-v2 è un modello di embedding multilingua, che è stato utilizzato in task di retrieval.

4.4 Setup del progetto

Analizzati nel dettaglio i principi architetturali e valutati attentamente gli strumenti e prodotti messi a disposizione dell'azienda, è stato possibile configurare un ambiente di sviluppo avanzato per il progetto di tesi. Il progetto

è stato sviluppato all'interno del cluster aziendale, con l'accesso a tutte le infrastrutture reso possibile dall'utilizzo di una VPN. La piattaforma utilizzata per lo sviluppo è stata Ice4AI, ospitata su VURANIA, che ha permesso di utilizzare risorse dedicate e di avere accesso ad ambienti virtuali preconfigurati con le librerie e i framework necessari per lo sviluppo di modelli di intelligenza artificiale. Per tutte le attività del progetto è stato utilizzato Python come linguaggio di programmazione. Il progetto è stato sviluppato principalmente utilizzando Jupyter Notebook, all'interno di un ambiente basato su Pytorch, esteso con l'installazione di numerose librerie di supporto, tra cui LangChain, HuggingFace Transformers, GraphRAG e molte altre. I modelli, precedentemente descritti, sono stati usati per svariate attività, senza la necessità di doverli configurare o installare, in quanto già esposti e pronti all'uso all'interno del cluster.

Capitolo 5

Risultati sperimentali

In questo capitolo sono riportati gli esperimenti effettuati per valutare le performance di GraphRAG rispetto a RAG tradizionale. Vengono descritti i dettagli della sperimentazione, i task assegnati ai due sistemi, i metodi di valutazione e i risultati ottenuti. Dopo una breve introduzione ai metodi impiegati per la sperimentazione, si descrivono le fasi di preprocessing dei dati, le prove preliminari di GraphRAG per la configurazione ottimale dei parametri di indicizzazione e i test comparativi tra GraphRAG e RAG tradizionale sui due task di valutazione. Infine, si presenta un sistema ibrido, che si propone di combinare i punti di forza dei due sistemi per ottenere risultati migliori di entrambi.

5.1 Introduzione alla sperimentazione

La sperimentazione, così come tutto il lavoro svolto, si pone l'obiettivo di analizzare il nuovo approccio di RAG basato su grafi. L'analisi di questa nuova tecnica è stata effettuata tramite confronto con il tradizionale RAG, per valutare le performance e notare eventuali differenze. Questi due metodi sono stati testati in condizioni identiche, utilizzando lo stesso LLM, lo stesso set di documenti, lo stesso set di domande e sono stati eseguiti sullo ambiente (Ice4AI) per garantire che entrambi abbiano a disposizione le risorse necessarie.

La sperimentazione comprende più fasi, partendo dal processamento dei dati, passando per l'esplorazione dei parametri di configurazione di GraphRAG, fino ad arrivare al confronto diretto delle performance dei due sistemi. Il cuore della sperimentazione è costituito dai due task di valutazione, uno globale e uno locale, che sono stati assegnati a GraphRAG e RAG tradizionale. Questi due task si pongono obiettivi diversi, il primo mira a ottenere una panoramica d'insieme dei contenuti dei documenti, mentre il secondo si concentra su risposte precise e mirate per quesiti specifici. Le due pipeline saranno dunque valutate in base alla qualità delle risposte fornite per una query-focused summarization e per una question answering classica.

5.2 Preprocessing dei dati

La fase di preprocessing dei dati è fondamentale per rendere i documenti PDF compatibili con GraphRAG, che supporta come input esclusivamente file in formato `.txt`. Per questa operazione è stato utilizzato **LlamaParse**, uno strumento specializzato per la conversione di documenti PDF in formato testo (`.txt`), con una gestione avanzata degli elementi complessi presenti nei documenti scientifici. LlamaParse estrae accuratamente il contenuto dai PDF, mantenendo la formattazione del testo e convertendo elementi complessi come tabelle e immagini in un formato leggibile da un modello di linguaggio. Questo permette a GraphRAG di trattare informazioni strutturate (tabelle) e visive (immagini) come contenuti testuali, utilizzando una rappresentazione descrittiva che ne preserva il contesto e la comprensibilità.

La procedura di conversione con LlamaParse prevede i seguenti passaggi:

- **Estrazione del testo**: il testo presente nel documento PDF viene estratto e segmentato per mantenere l'integrità logica dei contenuti, inclusi titoli, sezioni e paragrafi.
- **Conversione delle tabelle**: le tabelle vengono identificate e convertite in un formato testuale strutturato che rappresenta chiaramente le righe, le colonne e le intestazioni, rendendo le informazioni tabulari accessibili per il modello di linguaggio.

- **Gestione delle immagini**: le immagini rilevanti vengono descritte attraverso didascalie estratte o create appositamente da LlamaParse, permettendo al modello di linguaggio di acquisire una comprensione del contenuto visivo presente nel documento, laddove appropriato.
- **Rimozione della bibliografia**: le citazioni bibliografiche e le referenze, di default verrebbero mantenute, ma si è scelto di rimuoverle per evitare che il modello di linguaggio le consideri come contenuto informativo.

L'utilizzo di LlamaParse ha permesso di preservare la complessità informativa dei documenti PDF originari, assicurando che le tabelle e le immagini fossero comprensibili anche in formato `.txt`. Questo ha reso possibile una conversione ottimale per GraphRAG, mantenendo intatta la ricchezza di contenuto dei documenti scientifici, che è essenziale per il task di query-focused summarization e question answering sui paper tecnici.

5.3 Prove preliminari su GraphRAG

Prima di procedere con la sperimentazione vera e propria, sono state effettuate delle prove preliminari per testare le performance di GraphRAG in condizioni controllate e per esplorare i parametri di configurazione ottimali per il modello. I test eseguiti in questa fase hanno il compito di valutare l'impatto dei principali parametri sulle performance di indicizzazione. Per valutare l'efficacia della fase di indicizzazione, ci si può basare sul numero di entità e relazioni estratte, sul numero di cluster identificati, con relativi report collegati e sui tempi impegati.

5.3.1 Prompt tuning

Uno dei primi esperimenti effettuati ha riguardato l'utilizzo del prompt tuning, confrontando i risultati ottenuti con e senza ottimizzazione dei prompt. Nella configurazione base, senza il prompt tuning, il sistema ha estratto 614

entità, 838 relazioni e generato 111 report. Applicando il prompt tuning, invece, il sistema ha estratto 2318 entità, 2995 relazioni e generato 349 report. I risultati sono riportati nella tabella seguente:

Configurazione	**Entità**	**Relazioni**	**Report**
Senza Prompt Tuning	614	838	111
Con Prompt Tuning	2318	2995	349

Tabella 5.1: Confronto delle performance di GraphRAG con e senza prompt tuning

I risultati mostrano chiaramente come il prompt tuning sia fondamentale per migliorare la qualità dell'indicizzazione. Grazie al riconoscimento automatico del dominio dei dati, GraphRAG è in grado di creare prompt personalizzati. In questo caso, ha identificato come dominio "Algorithmic analysis" e, più nello specifico, "Machine learning" o "Time series forecasting" come dominio principale. Il sistema ha quindi generato prompt specifici, che hanno migliorato l'estrazione di entità e relazioni, aumentando la densità di informazioni nei report prodotti.

5.3.2 Chunk size tuning

Un secondo test ha esaminato l'effetto della dimensione del *chunk-size* sull'indicizzazione. Il parametro predefinito di *chunk-size* è stato inizialmente impostato a 600 token, ma sono stati testati anche i valori di 800 e 1000 token. Come previsto, all'aumentare della dimensione del *chunk*, i tempi di indicizzazione sono diminuiti, ma si è osservato anche un calo nelle entità, relazioni e report estratti (Tabella **??** e Figura 5.1).

Chunk Size	**Entità**	**Relazioni**	**Report**	**Tempo (s)**
600	2318	2995	349	2340
800	1940	2299	267	2091
1000	1567	1916	209	1743

Tabella 5.2: Confronto delle performance di GraphRAG al variare della dimensione del chunk

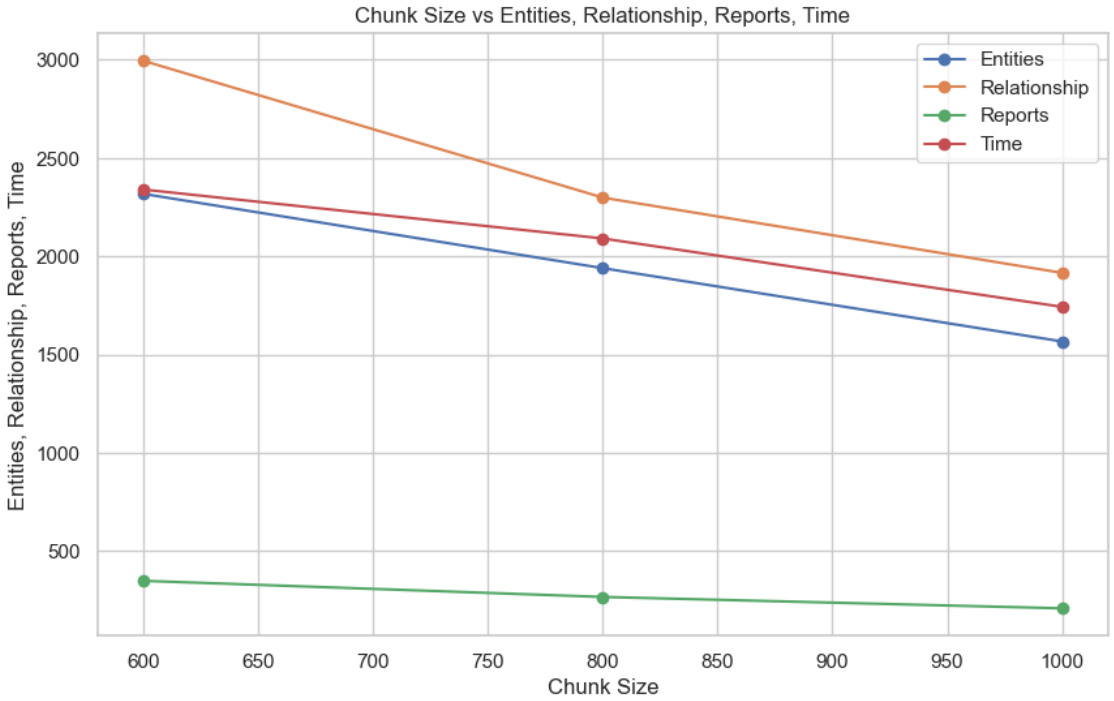

Figura 5.1: Confronto delle performance di GraphRAG al variare della dimensione del chunk

L'osservazione chiave di questo test è la presenza di un fenomeno noto come *lost in the middle* [53], che si verifica quando si usano *chunk* di dimensioni elevate. Tale fenomeno comporta che i modelli di linguaggio possano perdere di vista alcune informazioni nel mezzo di blocchi di testo estesi, con il rischio di non riconoscere o trascurare dettagli rilevanti. Sebbene l'aumento della dimensione del *chunk* possa ridurre i tempi di indicizzazione, è importante bilanciare questo aspetto con la qualità delle informazioni estratte. La fase di indicizzazione, nonostante sia molto onerosa sia in termini di tempo sia di chiamate al modello di linguaggio, è effettuata una sola volta; è stato quindi preferibile mantenere la configurazione di 600 token per preservare la massima quantità di informazioni.

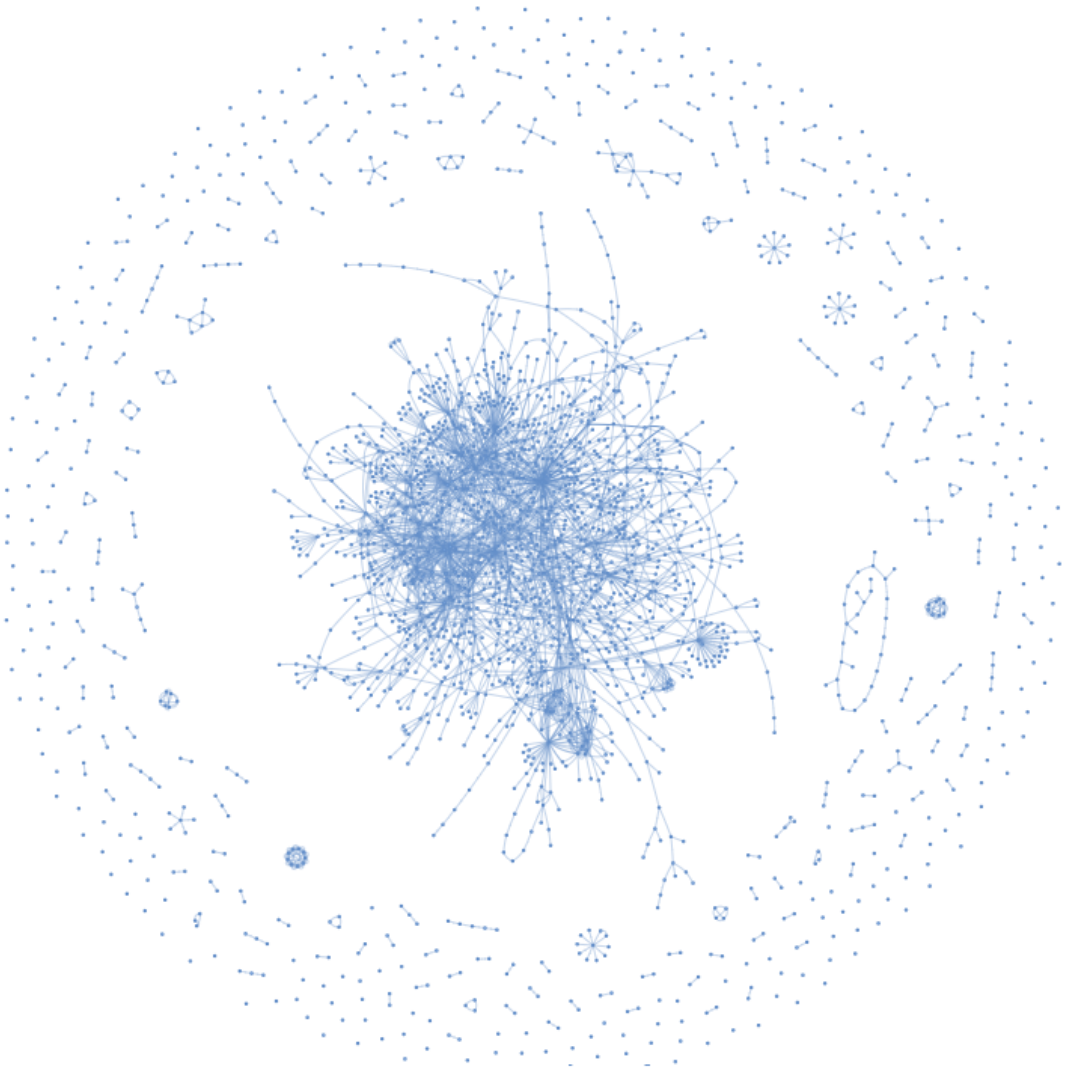

Figura 5.2: Grafo generato da GraphRAG

5.3.3 Gleaning tuning

Nella configurazione di GraphRAG, è presente un ulteriore parametro avanzato, denominato *gleanings*, che controlla il numero di passaggi di estrazione e arricchimento semantico da effettuare su ciascun *chunk* durante la fase di indicizzazione. Di default, questo parametro è impostato a 0, il che significa che viene effettuato un solo passaggio di estrazione delle informazioni. Per questo studio, si è deciso di mantenere *gleanings* a 0 per semplificare la fase di

indicizzazione e ottenere tempi di elaborazione accettabili. Tuttavia, un aumento di *gleanings* potrebbe portare a una maggiore densità di informazioni e una rappresentazione ancora più dettagliata, poiché ogni passata successiva analizza i dati estratti in precedenza e tenta di arricchire ulteriormente le entità e le relazioni identificate.

In Figura 5.2 è possibile osservare il grafo generato da GraphRAG, utilizzando i parametri di configurazione scelti. Essendo il numero di entità e relazioni estratte molto elevato, il grafo risulta molto complesso e denso, con molteplici connessioni.

5.4 Confronto tra GraphRAG e RAG tradizionale

La fase dei test preliminari ha permesso di configurare GraphRAG per ottenere le migliori performance possibili. A questo punto, è stato possibile confrontare direttamente GraphRAG con RAG tradizionale, utilizzando le stesse condizioni e gli stessi dati per entrambi i sistemi.

5.4.1 Task di valutazione

Per confrontare efficacemente GraphRAG con il RAG tradizionale, sono stati definiti due task di valutazione distinti, ciascuno con finalità specifiche. Entrambi i task si basano sullo stesso set di documenti e richiedono ai sistemi di rispondere a domande, ma differiscono per il livello di comprensione e il tipo di risposta attesa.

- **Task 1 - Query-Focused Summarization:** questo task si concentra sull'abilità dei sistemi di generare una panoramica globale e contestualizzata dei contenuti del documento. Le domande di questo task richiedono risposte estese e articolate, che sintetizzino i principali concetti e relazioni tra gli argomenti trattati nei documenti. L'obiettivo è testare la capacità dei modelli di comprendere il contesto generale e di fornire risposte complete, utilizzando le informazioni in modo coerente e strutturato.

- **Task 2 - Question Answering Specifico:** questo task valuta la capacità dei sistemi di rispondere a domande puntuali su dettagli specifici del documento. Le domande poste in questo task richiedono risposte concise, mirate e accurate, focalizzate su aspetti o concetti precisi. Il task verifica quanto i sistemi riescano a isolare informazioni rilevanti e a rispondere con precisione a quesiti di natura più dettagliata, piuttosto che riassuntiva.

I due task permettono di testare i due approcci in condizioni diverse e di valutare se le performance differiscono a seconda del tipo di domanda posta.

5.4.2 Creazione dei dataset di test

Per eseguire il confronto tra GraphRAG e RAG tradizionale, è stato necessario sviluppare un dataset di domande rappresentativo e costruire un *golden dataset* per valutare la qualità delle risposte di entrambi i sistemi.

Creazione delle Domande

Il set di domande è stato generato utilizzando un modello di linguaggio di grandi dimensioni (*Llama3.1-70B*), selezionato per la sua capacità di produrre output di alta qualità e coerenza. Il modello è stato configurato per generare domande che coprissero i contenuti dei documenti scientifici su vari livelli di dettaglio e complessità. In questo modo, si è creato un set di domande adeguato per entrambi i task di valutazione:

- **Domande globali:** pensate per il task di query-focused summarization, sono formulate per richiedere una panoramica o sintesi dei contenuti principali dei documenti.
- **Domande locali:** progettate per il task di question answering, mirano a valutare la capacità dei sistemi di rispondere a richieste puntuali su dettagli contenuti nei documenti.

Il modello di linguaggio è stato utilizzato per generare i due set di domande, che sono stati poi verificati manualmente per garantire la qualità e la coerenza delle richieste. Sono state generate 35 domande globali che coprono una vasta gamma di argomenti comuni ai vari documenti o che richiedono la comprensione e sintesi di tutto un documento. Alcuni esempi di domande globali sono:

- *What are the main topics covered by the data in the set of time-series papers?*
- *How are transformer-based models improving the accuracy of anomaly detection in real-time applications?*
- *How do the different models balance the trade-off between model complexity and interpretability when applied to real-world time series tasks such as anomaly detection and forecasting?*

Per quanto riguarda le domande locali, sono state invece generate 45 domande, 5 per ciascun documento, che richiedono risposte specifiche su dettagli contenuti nei documenti. Alcuni esempi di domande locali sono:

- *What is the size of the training dataset used for TimeGPT, as mentioned in Section 5.2 of the paper?*
- *What is the evaluation metric used to compare the performance of AnomalyBERT with previous works, as described in Section 4.2?*
- *What are some potential limitations and future directions for the TimesFM model, as discussed in the paper?*

Creazione del Golden Dataset

Generati i due set di domande, è stato necessario creare un *golden dataset* per valutare le risposte fornite da GraphRAG e RAG tradizionale. Anche questa fase ha sfruttato il modello *Llama3.1-70B*, utilizzato per generare risposte complete e di alta qualità, che fungano da standard di riferimento. Il modello è stato configurato con prompt specifici per ogni domanda, fornendo risposte che includessero i dettagli più rilevanti e contestualizzati.

Anche in questo caso, le risposte generate sono state verificate manualmente per garantire l'assenza di allucinazioni o informazioni errate. Nonostante la fase di verifica, il *golden dataset* generato è da considerarsi come un riferimento approssimativo, in quanto le risposte fornite dal modello di linguaggio potrebbero non essere perfettamente accurate o complete.

5.4.3 Esecuzione delle due pipeline

Una volta creati i dataset di test, è stato possibile eseguire le due pipeline, GraphRAG e RAG tradizionale. Le configurazioni dei due sistemi sono state descritte in precedenza, insieme anche alle fasi che compongono le due pipeline. Entrambi i sistemi sono stati eseguiti sul set di documenti e testati con le domande generate, salvando le risposte prodotte per ciascuna domanda. Oltre alle risposte, sono stati registrati anche i tempi di esecuzione delle due pipeline, per valutare le performance in termini di velocità.

5.4.4 Tempi di esecuzione e chiamate al modello

Per confrontare le performance di GraphRAG e RAG tradizionale, l'analisi si è concentrata come primo aspetto sui tempi di risposta alle query delle due pipeline e sul numero di chiamate al LLM per ciascuna domanda, facendo una distinzione tra domande globali e locali. Non sono stati inclusi nell'analisi i tempi di indicizzazione, poichè si tratta di un'operazione eseguita una sola volta e non influisce direttamente sul funzionamento del sistema. Tuttavia, è importante notare che GraphRAG richiede un tempo di indicizzazione molto più lungo rispetto a RAG tradizionale, a causa del numero elevato di chiamate al modello di linguaggio necessarie per estrarre le entità e le relazioni dai documenti e successivamente per generare i report.

Nel contesto aziendale specifico, l'utilizzo di un LLM privato mitiga l'impatto delle chiamate al modello, poiché non comporta costi aggiuntivi per l'azienda. Tuttavia, è importante analizzare comunque questo aspetto, che potrebbe essere rilevante con altre configurazioni o in contesti diversi.

Domande globali Per le domande globali, GraphRAG adotta una pipeline articolata, basata su un meccanismo di map-reduce, che coinvolge un numero elevato di chiamate al modello di linguaggio per generare una risposta. Nel caso di questo progetto, con la configurazione attuale, GraphRAG effettua 18 chiamate al modello per la fase di map e 1 chiamata per la fase di reduce, per un totale di 19 chiamate per ogni domanda globale. Il tempo medio di risposta di GraphRAG per una domanda globale è di **22.46 secondi**, mentre il RAG tradizionale ha completato le risposte nello stesso scenario in **7.16 secondi** di media, effettuando una singola chiamata al LLM.

Domande locali Per le domande locali, entrambe le pipeline effettuano una singola chiamata al LLM per rispondere alla query. Tuttavia, la configurazione interna delle due pipeline influisce significativamente sui tempi di risposta. In questo scenario, il RAG tradizionale ha risposto in media in **4,84 secondi** per ogni domanda locale, mentre GraphRAG ha impiegato un tempo medio di **9,91 secondi** per rispondere, circa il doppio rispetto al RAG tradizionale.

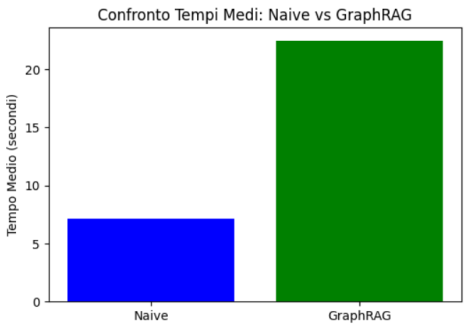

Figura 5.3: Tempi medi di risposta per domande globali

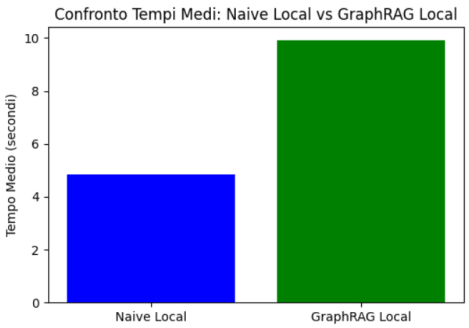

Figura 5.4: Tempi medi di risposta per domande locali

5.4.5 Valutazione umana delle risposte

Per comprendere meglio la qualità delle risposte fornite da GraphRAG e RAG tradizionale, è stata effettuata una valutazione umana preliminare delle risposte generate. Questa fase mira ad esaminare principalmente la tipologia di

risposte prodotte dai due sistemi, valutando se le risposte vengano percepite come corrette, se il linguaggio utilizzato è chiaro e comprensibile e se è possibile notare differenze qualitative tra le risposte di GraphRAG e RAG tradizionale. Non ci si aspetta una valutazione dettagliata o quantitativa da questa fase, poichè spetterà alla valutazione con LLM, ma è importante avere un'idea generale della qualità delle risposte prodotte. La percezione umana delle risposte è un aspetto importante da considerare, spesso trascurato nelle valutazioni di sistemi di NLP e non facile da quantificare e riassumere in metriche standard, ma che può fornire informazioni preziose sulla qualità e l'efficacia dei sistemi.

La valutazione umana è stata effettuata con conoscenza del dominio, tramite confronto diretto delle risposte di GraphRAG e RAG tradizionale per ciascuna domanda. Ciò che è potuto emergere da questa fase è che per quanto riguarda le domande globali, GraphRAG ha dimostrato una maggiore capacità di fornire risposte complete, mentre RAG tradizionale ha mostrato una tendenza a fornire risposte più limitate. Questa osservazione è in linea con le aspettative, poiché GraphRAG è progettato per fornire una panoramica globale dei contenuti, garantendo una visione d'insieme dei documenti, simile a quello che potrebbe essere un ragionamento umano. Per quanto riguarda le domande locali, entrambi i sistemi hanno dimostrato di essere in grado di fornire risposte complesse e ricche di dettagli, che vanno necessariamente valutate con un sistema più rigoroso e dettagliato. Tramite l'analisi umana si è potuto notare inoltre come GraphRAG sia migliore dal punto di vista della spiegabilità, fornendo i riferimenti ai report e alle entità utilizzate per generare la risposta, mentre RAG tradizionale non fornisce informazioni aggiuntive sulle risposte prodotte.

Questa metodologia di valutazione umana è stata utile per ottenere un feedback qualitativo sulle risposte prodotte dai due sistemi e per comprendere meglio alcune differenze evidenti tra i due approcci. Tuttavia, è importante sottolineare che questa fase non è esaustiva e non fornisce una valutazione completa delle performance dei sistemi, che richiedono una confronto più dettagliato e rigoroso.

5.4.6 LLM as a Judge - Valutazione senza golden dataset

La prima delle due fasi di valutazione, condotta con il modello di linguaggio come giudice, è stata effettuata senza l'utilizzo del *golden dataset*. Questa analisi rappresenta un proseguimento naturale della valutazione umana, andando ad approfondire alcuni aspetti qualitativi delle risposte dei due sistemi, concentrandosi su metriche orientate all'esperienza utente. Le metriche selezionate per questa fase sono **Comprehensiveness**, **Diversity**, e **Empowerment**, tutte focalizzate a misurare quanto le risposte siano complete, ricche di dettagli e utili per l'utente. Questa valutazione, sebbene non basata su un golden dataset di risposte perfettamente standardizzate, offre una prospettiva sulle risposte generate dai due modelli, consentendo di evidenziare differenze qualitative significative. Il modello di linguaggio è stato utilizzato per scegliere quale risposta fosse migliore tra GraphRAG e RAG tradizionale, fornendo un giudizio imparziale secondo un prompt specifico per ciascuna metrica. Di seguito è riportato un esempio del prompt usato per valutare la metrica di **Comprehensiveness**:

```
### Task Description:
You are an expert evaluator tasked with assessing the
    **Comprehensiveness** of two generated answers in
    response to the same query. **Comprehensiveness**
    refers to how thoroughly each answer covers all
   aspects and details of the question, providing
   complete and relevant information.

### Instructions:
1. Compare the two generated answers and determine
   which one is better in terms of **
   Comprehensiveness**.
2. Indicate your choice without providing scores or
   detailed feedback.
3. The output format should be:

        "better_answer": "Model␣A" // oppure "Model␣B
           " o "Tie"
```

```
4. Do not include any additional text beyond the
   specified output format.
5. Ensure that your evaluation is objective and
   strictly follows the definition of **
   Comprehensiveness**.

### Query:
{question}

### Model A Answer:
{model_a_answer}

### Model B Answer:
{model_b_answer}

### Evaluation:
```

Listing 5.1: Prompt per la valutazione di Comprehensiveness

Risultati

I risultati della valutazione con il modello di linguaggio come giudice, senza l'utilizzo del *golden dataset*, sono riportati nelle tabelle seguenti. Ogni metrica è stata valutata separatamente per entrambi i set di domande.

Metriche	**RAG Tradizionale**	**GraphRAG**	**Totale Domande**
Comprehensiveness	1	34	35
Diversity	0	35	35
Empowerment	5	30	35

Tabella 5.3: Risultati della valutazione su domande globali

Metriche	**RAG Tradizionale**	**GraphRAG**	**Totale Domande**
Comprehensiveness	17	28	45
Diversity	21	24	45
Empowerment	26	19	45

Tabella 5.4: Risultati della valutazione su domande locali

Per quanto riguarda le domande globali, GraphRAG ha mostrato una superiorità notevole, come notato anche in fase di valutazione umana e confermato da tutte e tre le metriche. Per le domande locali, invece, GraphRAG si è dimostrato superiore in termini di **Comprehensiveness** e **Diversity**, mentre RAG tradizionale ha ottenuto risultati migliori per la metrica di **Empowerment**. Questi risultati confermano la tendenza osservata in fase di valutazione umana, con GraphRAG che si dimostra più adatto per rispondere a domande globali, mentre per domande locali entrambi i sistemi producono risposte apparentemente valide. A questo punto, è necessario effettuare una valutazione definitiva per poter conclusivamente determinare quale sistema sia migliore per i due task di valutazione.

5.4.7 LLM as a Judge - Valutazione con golden dataset

La seconda fase di valutazione, condotta con il modello di linguaggio come giudice, è stata effettuata utilizzando i *golden dataset* creati in precedenza. Questa fase rappresenta il passo finale della valutazione, in cui il modello di linguaggio è utilizzato per confrontare le risposte di GraphRAG e RAG tradizionale con le risposte di riferimento, per determinare quale sistema abbia prodotto risposte migliori e più accurate. L'obiettivo di questa fase è ottenere una misurazione oggettiva e quantitativa delle performance dei due sistemi, basata su criteri standardizzati e confrontabili. Per questa analisi sono state scelte tre metriche fondamentali: **Correctness**, **Completeness** e **Relevance**, ciascuna delle quali valuta aspetti specifici delle risposte prodotte.

La valutazione è stata effettuata tramite un modello di linguaggio che, utilizzando i golden dataset, confronta ciascuna risposta generata con la risposta di riferimento e assegna un punteggio su una scala da 1 a 5 per ciascuna metrica. Di seguito è riportato un esempio del prompt utilizzato per la valutazione della **Correctness**:

```
Task Description:
```

```
You are given a query, a generated answer, a
   reference answer (which receives a score of 5),
   and a scoring rubric representing the evaluation
   criteria for correctness.

Instructions:

Provide detailed feedback that assesses the
   correctness of the generated answer strictly based
    on the scoring rubric.
After writing the feedback, assign a score from 1 to
   5 according to the rubric.
The output format should be:
Feedback: (your detailed feedback)
[RESULT] (1-5) or Score: (1-5)
Do not include any additional text beyond the
   feedback and the score.
Focus your evaluation only on the content present in
   both the generated answer and the reference answer
   . Do not penalize for missing information not
   present in the generated answer.
Query:
{query}

Generated Answer:
{generated_answer}

Reference Answer (Score 5):
{reference_answer}

Scoring Rubric for Correctness:
Score 1: The generated answer is completely incorrect
    and does not relate to the query or the reference
    answer.
```

```
    Score 2: The generated answer has significant
       inaccuracies and fails to correctly address the
       main points of the query or the reference answer.
    Score 3: The generated answer is partially correct
       but contains notable errors or misconceptions.
    Score 4: The generated answer is mostly correct with
       minor inaccuracies.
    Score 5: The generated answer is entirely correct and
        aligns perfectly with the reference answer in
       terms of factual accuracy.
    Feedback:
```

Listing 5.2: Prompt per la valutazione di Correctness

Risultati su domande globali

La valutazione delle risposte alle domande globali ha evidenziato una chiara superiorità di GraphRAG, come si può osservare dai punteggi medi riportati nella Tabella 5.5. GraphRAG ha ottenuto risultati significativamente migliori rispetto a RAG tradizionale in tutte e tre le metriche.

Metrica	**GraphRAG**	**RAG Tradizionale**
Correctness	3.41	2.54
Completeness	3.62	2.38
Relevance	4.95	3.59

Tabella 5.5: Punteggi medi su domande globali

I risultati confermano la tendenza osservata in fase di valutazione umana e nella valutazione preliminare, con GraphRAG che si dimostra superiore in termini di completezza, correttezza e rilevanza delle risposte. L'approccio con grafi di conoscenza è progettato per task di query focused summarization, che richiedono una visione d'insieme dei contenuti e una comprensione approfondita del contesto, e i risultati ottenuti confermano la validità di questo approccio per questo tipo di task. I risultati confermano anche l'inadeguatezza degli approcci tradizionali a rispondere a domande globali, complesse

e articolate, che richiedono una comprensione profonda dei contenuti e una capacità di sintesi avanzata.

Risultati su domande locali

Per quanto riguarda le domande locali, i risultati mostrano un'inversione di tendenza, con RAG tradizionale che ottiene punteggi medi più elevati rispetto a GraphRAG in tutte le metriche, come riportato nella Tabella 5.6.

Metrica	GraphRAG	RAG Tradizionale
Correctness	2.15	3.00
Completeness	2.26	3.18
Relevance	2.58	3.53

Tabella 5.6: Punteggi medi su domande locali

Questi risultati confermano le aspettative riguardo alle performance del RAG tradizionale, che è performante in task di question answering specifico, come le domande locali, fortemente focalizzate su dettagli specifici e risposte puntuali. GraphRAG, d'altra parte, si è dimostrato meno adatto a rispondere a domande locali, producendo risposte meno accurate e complete rispetto a RAG tradizionale. Questa fase di valutazione mette in luce i limiti di GraphRAG in task di question answering specifico, che richiedono una precisione e una completezza delle risposte che il sistema non è in grado di garantire.

5.4.8 Confronto finale e conclusioni

Il confronto tra GraphRAG e RAG tradizionale ha permesso di valutare le performance dei due sistemi in condizioni controllate e di confrontare i due approcci in due task di valutazione distinti. I risultati preliminari e la valutazione con il modello di linguaggio come giudice hanno evidenziato differenze significative tra i due sistemi, con GraphRAG che si è dimostrato superiore in task di query-focused summarization, mentre RAG tradizionale ha ottenuto risultati migliori in task di question answering specifico. I risultati ottenuti

confermano in parte le aspettative, infatti i due approcci performano meglio in task per i quali sono stati progettati, ma mettono in luce anche alcune limitazioni e punti deboli dei due sistemi, quando utilizzati in contesti diversi.

La valutazione ha mostrato che tra i due approcci non esiste un sistema nettamente superiore, ma che entrambi hanno punti di forza e debolezza, che li rendono più adatti a task specifici. Sono stati comunque calcolati i punteggi globali per le tre metriche, ottenendo i seguenti risultati:

Metrica	**GraphRAG**	**RAG Tradizionale**
Correctness	2.77	2.77
Completeness	2.96	2.78
Relevance	3.76	3.56

Tabella 5.7: Punteggi medi globali

I risultati finali mostrano che i punteggi medi dei due approcci sono molto simili, con GraphRAG che ottiene punteggi leggermente superiori in tutte e tre le metriche.

Dalla sperimentazione non è comunque emerso un sistema nettamente superiore, in grado di affrontare task differenti con prestazioni sempre elevate. Da questa limitazione nasce l'idea di creare un sistema ibrido, che possa sfruttare i punti di forza di entrambi i sistemi e combinare le due pipeline per ottenere risultati migliori e più completi.

5.5 Sistema Ibrido: Integrazione di GraphRAG e RAG Tradizionale

A seguito delle limitazioni riscontrate nei due sistemi, GraphRAG e RAG tradizionale, è stato ideato un sistema ibrido che sfrutti i punti di forza di entrambi i sistemi e combini le due pipeline per ottenere risultati migliori e più completi. Quello che si vuole ottenere è un sistema più flessibile e adattabile, in grado di affrontare task differenti con prestazioni elevate e di sfruttare al meglio le potenzialità dei due approcci. L'idea è quella di identificare la tipologia di domanda posta e di indirizzarla alla pipeline più adatta, sfruttando

le capacità di GraphRAG per task di query-focused summarization e quelle di RAG tradizionale per task di question answering specifico.

5.5.1 Architettura del sistema ibrido

Il sistema si basa su un meccanismo di classificazione per distinguere automaticamente tra domande globali e locali e, di conseguenza, attivare la pipeline più appropriata. Questa operazione di classificazione è effettuata utilizzando non un modello di classificazione, ma ancora una volta un modello di linguaggio, che riceve in input la domanda tramite un prompt specifico e restituisce una classificazione binaria tra domande globali e locali.

La struttura operativa del sistema ibrido si articola nei seguenti passaggi:

(a) **Creazione del prompt di classificazione:** il primo step consiste nel generare un prompt specifico perche contenga la domanda dell'utente e un insieme di istruzioni per il modello di linguaggio. Questo prompt ha l'obiettivo di guidare il modello a classificare correttamente la domanda in base alla tipologia.

(b) **Classificazione della domanda:** il prompt viene passato al modello di linguaggio esposto sul cluster aziendale. Il modello analizza il contenuto della domanda e, seguendo le istruzioni del prompt, decide se l'input rappresenta una richiesta di sintesi globale o una domanda focalizzata su dettagli specifici.

(c) **Attivazione della pipeline appropriata:** in base alla classificazione ottenuta, il sistema attiva la pipeline più adatta per rispondere alla domanda.

- Se la domanda è classificata come globale, il sistema attiva GraphRAG per generare una risposta per un task di query-focused summarization.
- Se la domanda è classificata come locale, il sistema attiva RAG tradizionale per rispondere a una richiesta di question answering specifico.

(d) **Generazione della risposta:** la pipeline attivata elabora la domanda e genera una risposta, che viene restituita all'utente.

5.5.2 Risultati del sistema ibrido

Il sistema ibrido è stato testato utilizzando lo stesso set di domande e documenti utilizzati per valutare GraphRAG e RAG tradizionale. Anche l'approccio di valutazione è stato lo stesso, con l'utilizzo del LLM come giudice per confrontare le risposte generate dal sistema ibrido con le risposte di riferimento.

Per prima cosa sono state testate le performance del modello di classificazione, valutando la capacità del modello di distinguere tra domande globali e locali. Durante i test, il classificatore basato sul modello di linguaggio ha correttamente identificato tutte le domande globali, attivando la pipeline di GraphRAG senza errori. Tuttavia, su un totale di 45 domande locali, 5 sono state erroneamente classificate come domande globali, innescando la pipeline di GraphRAG. Sebbene la classificazione errata di alcune domande locali come domande globali possa apparire come un errore, in alcuni casi questa sovrapposizione può essere considerata accettabile, poiché alcune domande locali richiedono comunque una comprensione complessiva del contesto. Infatti, il confine tra una domanda globale e una domanda locale non è sempre netto e definito, poiché alcune domande locali potrebbero beneficiare di una risposta più articolata o viceversa.

Note le performance del classificatore, è stato possibile valutare le performance del sistema ibrido nel suo complesso, confrontando le risposte generate con le risposte di riferimento. La valutazione delle performance non è stata effettuata facendo distinzione tra domande globali e locali, ma considerando l'insieme delle domande e generando solo un punteggio finale.

I punteggi delle tre metriche sono riportati in Tabella 5.8, confrontati a quelli che erano i punteggi finali delle due pipeline singole.

Metrica	GraphRAG	RAG Tradizionale	Sistema Ibrido
Correctness	2.77	2.77	3.22
Completeness	2.96	2.78	3.33
Relevance	3.76	3.56	4.54

Tabella 5.8: Confronto dei punteggi medi tra GraphRAG, RAG Tradizionale e Sistema Ibrido

I risultati, visibili anche graficamente in Figura 5.5 e Figura 5.6, mostrano che il sistema ibrido ottiene punteggi medi superiori rispetto a GraphRAG e RAG tradizionale in tutte e tre le metriche. Viene evidenziato come il sistema ibrido sia in grado di combinare i punti di forza dei due approcci e di migliorare di molto i punteggi finali. Questi risultati confermano l'efficacia del sistema ibrido nel fornire risposte più accurate, complete e rilevanti, sfruttando i punti di forza dei due approcci e mitigandone le debolezze nei rispettivi ambiti di specializzazione.

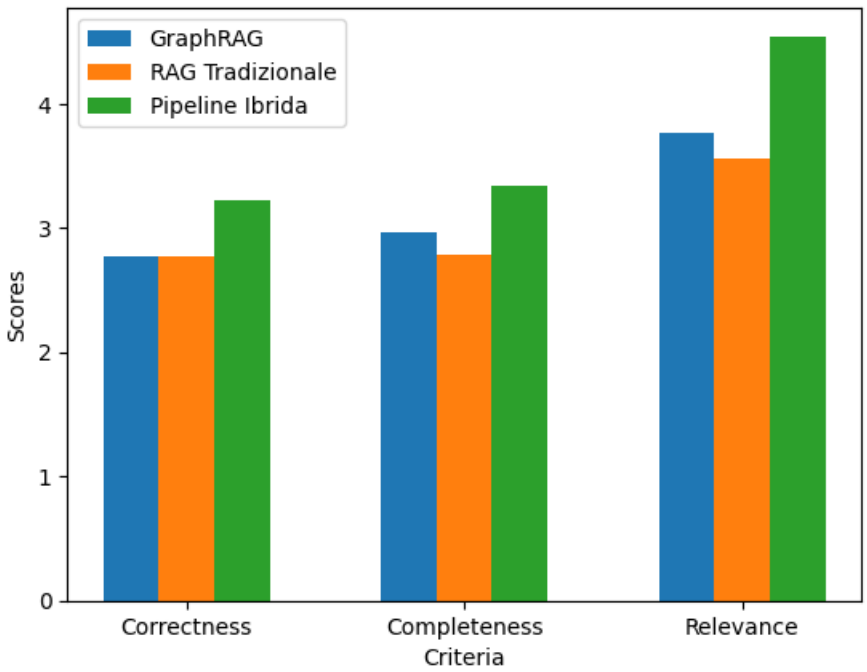

Figura 5.5: Metriche delle tre pipeline a confronto

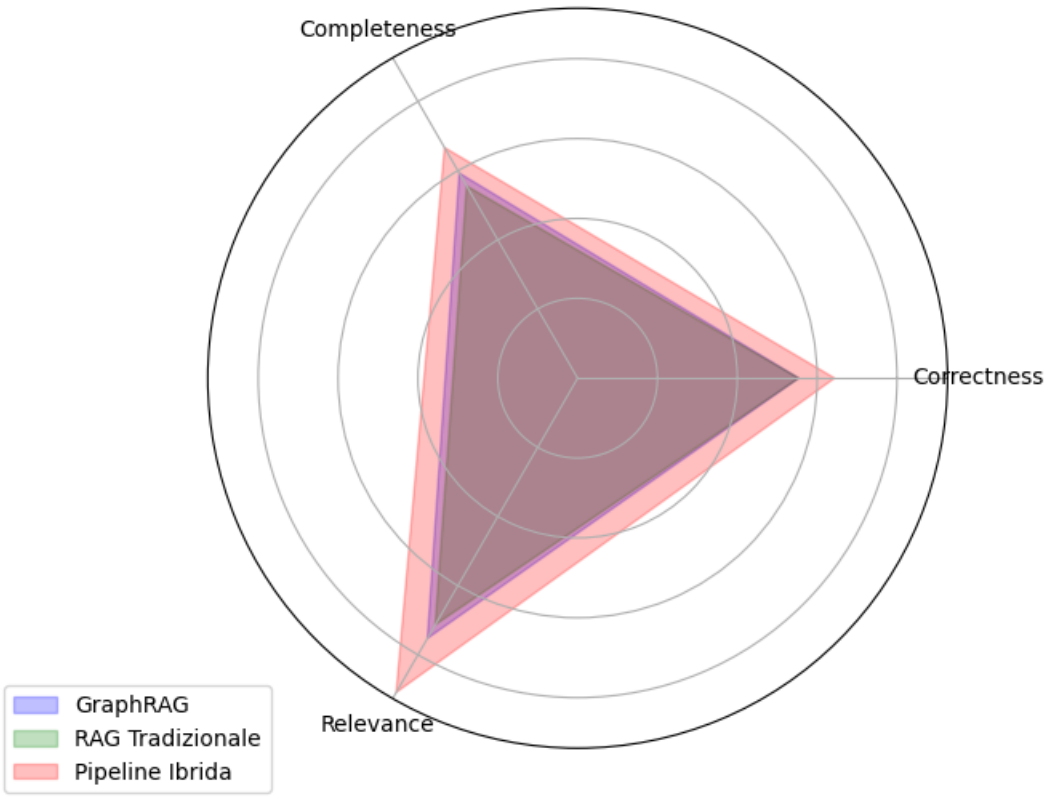

Figura 5.6: Metriche delle tre pipeline a confronto

Capitolo 6

Conclusioni e Sviluppi Futuri

6.1 Riepilogo e considerazioni finali

Il lavoro presentato in questa tesi ha esaminato e confrontato due approcci per task di NLP, in particolare il question answering e la query-focused summarization, applicati su documenti reali complessi. Dopo una fase di studio e progettazione approfondita e la sperimentazione di configurazioni diverse, caratterizzate da modifiche nei parametri chiave, per ottimizzare le performance nei due approcci, è stato sviluppato un sistema che permette di eseguire entrambi i task in modo efficiente e accurato.

Ciò che il lavoro ha dimostrato è che le tecnologie per affrontare uno dei due task possono essere riutilizzate per affrontare l'altro, ma i risultati non sono sempre garantiti. GraphRAG ha mostrato alcuni limiti in scenari di local search, infatti questa funzionalità è ancora in fase di ottimizzazione da parte degli sviluppatori di Microsoft. La pipeline di RAG tradizionale, come noto, è poco adatta a task di comprensione profonda e sintesi accurata e i test hanno confermato questa ipotesi.

Per questi motivi è stato necessario lo sviluppo di un sistema ibrido, che ha dimostrato di poter ottenere risultati migliori rispetto a quelli ottenuti con i due sistemi singoli. Il sistema ibrido sviluppato, si basa su un meccanismo molto semplice, ma nonostante ciò ha ottenuto risultati molto buoni. Questo

sistema può però essere ulteriormente migliorato e ottimizzato, partendo dall'implementazione sviluppata in questa tesi e aggiungendo nuove funzionalità e miglioramenti.

Questo lavoro di tesi ha esplorato le differenze tra i due approcci e proposto una soluzione di base, ma ci sono molte altre strade da esplorare e molti altri miglioramenti che possono essere apportati al sistema, di seguito discussi.

6.2 Limitazioni dell'approccio

La sperimentazione svolta nel progetto di tesi ha evidenziato delle differenze tra i due approcci, che hanno portato alla necessità di sviluppare un sistema ibrido. Tutti i risultati ottenuti sono da considerarsi solo come un punto di partenza per ulteriori sviluppi e miglioramenti, per una serie di motivi.

- **Dataset non ottimizzato per il task specifico**: il dataset utilizzato è stato creato appositamente per la sperimentazione, senza un riferimento standard specifico per questo task. Questo potrebbe aver influenzato i risultati e limitato la generalizzabilità delle conclusioni. In futuro, l'utilizzo di dataset standardizzati o la creazione di un dataset più completo per task specifici potrebbe migliorare la valutazione.
- **Limitato numero di test**: un numero maggiore di test, possibilmente su più dataset con caratteristiche diverse, potrebbe fornire un quadro più accurato delle reali performance di GraphRAG e del sistema ibrido. In questo lavoro, la variabilità dei documenti e delle domande è stata limitata; un set di test più ampio e diversificato potrebbe far emergere ulteriori punti di forza e debolezze del sistema.
- **LLM privato**: l'utilizzo di un LLM privato ha facilitato la gestione delle chiamate al modello, riducendo i vincoli di costo e latenza legati alle risorse. Tuttavia, per applicazioni su larga scala, il numero di chiamate al LLM diventa critico in termini di costi computazionali e di scalabilità. Questo aspetto richiederebbe ulteriori ottimizzazioni in contesti dove l'accesso ai modelli non sia completamente gestibile in-house.

6.3 Sviluppi Futuri

I risultati ottenuti suggeriscono interessanti spunti per possibili sviluppi e miglioramenti:

- **Rilascio del sistema ibrido con infrastruttura a microservizi:** il sistema ibrido è stato sviluppato solo in ambiente locale e di test, ma potrebbe essere rilasciato tramite accurata progettazione di un'infrastruttura a microservizi. Lo sviluppo naturale di questo progetto potrebbe essere l'ottimizzazione e il consecutivo rilascio del sistema ibrido in un ambiente di produzione. Il possibile schema architetturale dovrebbe sicuramente basarsi su una struttura a microservizi, per poter comprendere tutte le componenti (modello LLM, embedder, vector database, grafo di conoscenza, etc.) e poterle trattare come servizi indipendenti.
- **Sperimentazione con modelli di linguaggio differenti**: l'adozione di nuovi modelli o varianti avanzate di LLM, inclusi modelli ottimizzati per compiti di question answering o query-focused summarization, potrebbe incrementare la qualità delle risposte. La sperimentazione su modelli con caratteristiche diverse potrebbe portare a dei miglioramenti significativi.
- **Ottimizzazione della pipeline di RAG tradizionale**: esistono varianti del RAG tradizionale che includono tecniche avanzate come il reranking dei passaggi o l'uso di modelli più accurati per la selezione delle risposte. Provare a integrare tali miglioramenti nella pipeline potrebbe aumentare l'accuratezza nelle risposte, in particolare per i task di local search.
- **Realizzazione di piepline di GraphRAG differenti:** l'implementazione di pipeline di GraphRAG differenti, con configurazioni diverse, potrebbe portare a risultati migliori. Potrebbero essere sviluppate delle pipeline con passaggi aggiuntivi e meccanismi più complessi oppure con configurazioni dei parametri differenti. Altri test possibili potrebbero includere l'uso di tecnologie alternative per la pipeline di GraphRAG.

- **Realizzazione di un sistema ibrido più complesso:** il sistema ibrido sviluppato in questa tesi è molto semplice e basato su un classificatore LLM. Potrebbe essere implementata una versione più complessa, che includa più passaggi e meccanismi di fusione delle risposte più avanzati. Questo potrebbe portare a risultati migliori e più accurati.

In conclusione, il lavoro svolto in questa tesi ha dimostrato che l'approccio ibrido proposto può portare a risultati migliori rispetto ai due approcci singoli, ma tutte le considerazioni fatte e i risultati ottenuti vanno attentantamente valutati e considerati come punto di partenza per ulteriori sviluppi e miglioramenti. Questa tesi si propone come punto di partenza per la ricerca in ambito di pipeline ibride per task di NLP, e si auspica che possa essere di ispirazione per futuri lavori in questo campo.

Bibliografia

[1] David Fernández-Llorca, Emilia Gómez, Ignacio Sanchez, and Gabriele Mazzini. An interdisciplinary account of the terminological choices by eu policymakers ahead of the final agreement on the ai act: Ai system, general purpose ai system, foundation model, and generative ai. *Artificial Intelligence and Law*, pages 1–14, 08 2024.

[2] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech,

Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models, 2022.

[3] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, March 2023.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[5] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. volume 2, pages 1045–1048, 09 2010.

[6] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 38, 03 2013.

[7] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

[8] Deepak Kumar and Shoumya Singh. Advancements in transformer architectures for large language model: From bert to gpt-3 and beyond. *International Research Journal of Modernization in Engineering Technology and Science*, 06:2582–5208, 05 2024.

[9] Zhang. Attention mechanism output, 2023.

[10] Aston Zhang, Zack C. Lipton, Mu Li, and Alexander J. Smola. Multihead attention mechanism.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier

neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[12] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification, 2016.

[13] Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers, 2023.

[14] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[16] Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks, 2016.

[17] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[20] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[21] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

[22] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad

Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak,

Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024.

[23] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[24] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.

[25] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. Language models as knowledge bases?, 2019.

[26] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[27] Himank Jain. Rag & graphrag, 2024.

[28] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering, 2020.

[29] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models, 2023.

[30] Ontotext. What is graph rag? `https://www.ontotext.com/knowledgehub/fundamentals/what-is-graph-rag/`, 2023. Accessed: 2024-10-23.

[31] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization, 2024.

[32] Bhaskarjit Sarmah, Benika Hall, Rohan Rao, Sunil Patel, Stefano Pasquali, and Dhagash Mehta. Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction, 2024.

[33] Philip Rathle. The graphrag manifesto: Adding knowledge to genai, 2024.

[34] NebulaGraph. Nebulagraph launches industry-first graph rag: Retrieval-augmented generation with llm based on knowledge graphs, 2023.

[35] Memgraph. Graphrag with memgraph, 2024.

[36] Darren Edge, Ha Trinh, Steven Truitt, and Jonathan Larson. Graphrag: New tool for complex data discovery now on github. July 2024.

[37] Jonathan Larson and Steven Truitt. Graphrag: Unlocking llm discovery on narrative private data. February 2024.

[38] Alonso Guevara Fernández, Katy Smith, Joshua Bradley, Darren Edge, Ha Trinh, Sarah Smith, Ben Cutler, Steven Truitt, and Jonathan Larson. Graphrag auto-tuning provides rapid adaptation to new domains. *Microsoft Research Blog*, September 2024.

[39] Yungi Jeong, Eunseok Yang, Jung Hyun Ryu, Imseong Park, and Myungjoo Kang. Anomalybert: Self-supervised transformer for time series anomaly detection using data degradation scheme, 2023.

[40] Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. Chronos: Learning the language of time series, 2024.

[41] Yuxuan Liang, Haomin Wen, Yuqi Nie, Yushan Jiang, Ming Jin, Dongjin Song, Shirui Pan, and Qingsong Wen. Foundation models for time series analysis: A tutorial and survey. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 6555–6565. ACM, August 2024.

[42] Kashif Rasul, Arjun Ashok, Andrew Robert Williams, Hena Ghonia, Rishika Bhagwatkar, Arian Khorasani, Mohammad Javad Darvishi Bayazi, George Adamopoulos, Roland Riachi, Nadhir Hassen, Marin Biloš, Sahil Garg, Anderson Schneider, Nicolas Chapados, Alexandre Drouin, Valentina Zantedeschi, Yuriy Nevmyvaka, and Irina Rish. Lag-llama: Towards foundation models for probabilistic time series forecasting, 2024.

[43] Ramin Ghorbani, Marcel J. T. Reinders, and David M. J. Tax. Restad: Reconstruction and similarity based transformer for time series anomaly detection, 2024.

[44] Azul Garza, Cristian Challu, and Max Mergenthaler-Canseco. Timegpt-1, 2024.

[45] Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, and

Qingsong Wen. Time-llm: Time series forecasting by reprogramming large language models, 2024.

[46] Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting, 2024.

[47] Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. Tranad: Deep transformer networks for anomaly detection in multivariate time series data, 2022.

[48] Charles Anderson. Docker [software engineering]. *IEEE Software*, 32, 2015.

[49] E4 Analytics. GAIA 3.0: Generative Artificial Intelligence Appliance, 2023.

[50] E4 Analytics. Urania: La soluzione cloud native per l'Artificial Intelligence, 2023.

[51] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen

Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay

Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron

Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick,

Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The llama 3 herd of models, 2024.

[52] Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. Nomic embed: Training a reproducible long context text embedder, 2024.

[53] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

Ringraziamenti

Scrivere questa tesi non rappresenta solo il compimento di un percorso di studi, ma anche il coronamento di un percorso di crescita personale e la realizzazione di un sogno. Durante questi anni, ho affrontato sfide e momenti difficili che mi hanno messo alla prova, ma che allo stesso tempo hanno contribuito a formarmi, arricchendo non solo il mio sapere accademico, ma anche la mia consapevolezza personale. In questo cammino sono stato fortunato ad avere accanto a me persone che, con il loro sostegno, la loro pazienza e la loro fiducia, mi hanno permesso di superare ogni ostacolo e di raggiungere questo traguardo. Desidero dunque ringraziare queste persone, a partire dalla mia famiglia che è stata il mio pilastro più solido in questo percorso. Li ringrazio per avermi sostenuto ogni giorno, per la pazienza nei momenti difficili e per aver creduto nei miei sogni prima ancora che io stesso lo facessi. Un ringraziamento speciale va alla mia ragazza, Lucia, che ha condiviso con me ogni momento di questi anni, dandomi forza e serenità anche nei momenti più difficili. La sua presenza costante, il suo incoraggiamento e la sua comprensione hanno reso tutto questo più leggero. Grazie per essere stata il mio sostegno e la mia compagna di viaggio, questo traguardo è anche tuo. Desidero anche ringraziare i miei amici, che sono stati una presenza fondamentale lungo questo percorso. Grazie per avermi sostenuto e per avermi regalato dei momenti di spensieratezza e di allegria, che mi hanno aiutato a ricaricare le energie e ad affrontare le sfide con rinnovata determinazione. Un grazie anche ai miei compagni di corso, con i quali ho condiviso le fatiche e le gioie di questi anni. Ci siamo sostenuti a vicenda fino ad arrivare a questo traguardo, e sono sicuro che il legame che ci unisce continuerà a crescere anche in futuro. Ci tengo a ringraziare anche il mio relatore, il Prof. Fabrizio Montecchiani, per

avermi guidato nella realizzazione di questo progetto. Infine, desidero ringraziare tutti le altre persone che fanno parte della mia vita, che in un modo o nell'altro mi hanno aiutato ad arrivare fino a qui. Grazie a tutti voi!