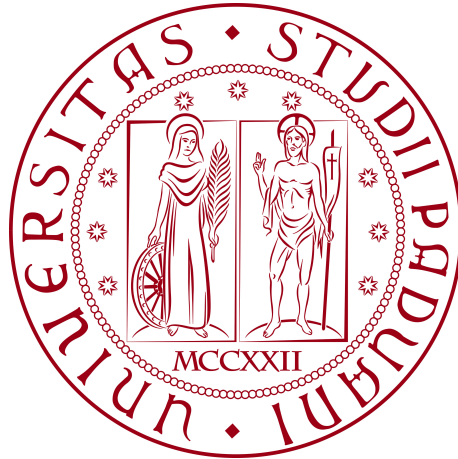


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA “TULLIO LEVI-CIVITA”

CORSO DI LAUREA IN INFORMATICA



**Come l'utilizzo dei Large Language Model può
influire sulla User Experience**

Tesi di laurea

Relatore

Prof. Zanella Marco

Laureando

Meneghini Fabio

Matricola 2034332

ANNO ACCADEMICO 2023-2024

“Non è mica necessario levarsi in volo fino al sole, basta strisciare fino a un posticino pulito sulla terra dove ogni tanto il sole faccia la sua comparsa e ci si possa riscaldare un po’.”

— Lettera al padre, Franz Kafka

Ringraziamenti

Innanzitutto ringrazio tutti i professori che mi hanno formato durante i corsi universitari; in particolare desidero ringraziare il professor Marco Zanella, il mio relatore, per la grandissima disponibilità e professionalità dimostrata e per il prezioso aiuto fornitomi durante la stesura della mia tesi.

Ringrazio inoltre tutti i miei amici, soprattutto i miei compagni di corso Samuele, Nicola e Leonardo, che fin dai primi giorni di università hanno alleggerito le mie giornate e le numerose sessioni di studio grazie alle infinite risate che abbiamo fatto assieme.

Infine desidero ringraziare di cuore mia mamma, la persona per me più importante: senza i tuoi enormi sacrifici e il tuo sostegno costante tutto questo non sarebbe stato possibile. Grazie davvero.

Padova, Dicembre 2024

Meneghini Fabio

Abstract

Il presente documento descrive il lavoro svolto durante il periodo di stage, dalla durata di trecentoventi ore, dal laureando Meneghini Fabio presso l'azienda Zucchetti S.p.A.

Il progetto di stage svolto riguarda tematiche inerenti all'*information retrieval* e ai *Large Language Model*: esso infatti consiste nello sviluppo di un *chatbot* che possa aiutare e guidare gli utenti meno esperti nell'utilizzo di *software* complessi e con molte funzionalità. Per fare ciò, il sistema dovrà segnalare i passaggi (per esempio, la pressione di un pulsante o una voce di un menù) che permettano di compiere l'operazione richiesta dall'utente al *chatbot*.

Prima di iniziare con lo sviluppo effettivo del progetto è stato svolto uno studio approfondito delle tecnologie e degli algoritmi coinvolti, i quali verranno analizzati nei successivi capitoli.

Indice

1	Introduzione	1
1.1	L'azienda	1
1.2	Strumenti utilizzati	2
1.3	Organizzazione del testo	3
1.3.1	Struttura del documento	3
1.3.2	Convenzioni tipografiche	4
2	Descrizione dello stage	5
2.1	Introduzione al progetto	5
2.1.1	User experience e usabilità	5
2.1.2	Funzionamento generale del chatbot	6
2.2	Analisi preventiva dei rischi	7
3	Analisi dei requisiti	10
3.1	Introduzione	10
3.2	Obiettivi dello stage	10
3.3	Attori	11
3.4	Casi d'uso	12
3.5	Tracciamento dei requisiti	27
3.5.1	Tabelle dei requisiti	28
4	Information retrieval	33
4.1	Introduzione	33
4.2	Panoramica ad alto livello del sistema	34
4.3	Data preparation	35

4.4	Chunking	36
4.4.1	Section-level chunking	37
4.4.2	Chunking a dimensione fissata con overlapping	37
4.5	Ricerca lessicale	38
4.5.1	Best Match 25	38
4.5.2	Stemming dei termini e rimozione delle stop words	39
4.6	Ricerca semantica	40
4.6.1	Sentence embedding	40
4.6.1.1	BERT e Sentence-BERT	40
4.6.2	Cosine similarity	42
4.7	Differenze tra i due tipi di ricerca	43
4.7.1	Ricerca per keywords presenti nel dataset	44
4.7.2	Ricerca per keywords non presenti nel dataset	45
4.7.3	Considerazioni	46
4.8	Ricerca ibrida	46
4.8.1	Algoritmi di rank fusion	46
4.8.1.1	Reciprocal Rank Fusion	47
4.8.1.2	Relative Score Fusion	47
4.8.1.3	Distribution-Based Score Fusion	52
4.8.1.3.1	Osservazioni	53
4.9	Knowledge graph	55
4.9.1	Generazione del grafo e query di attraversamento	57
5	Generazione della risposta	59
5.1	Introduzione	59
5.2	Question Answering	59
5.2.1	Extractive Question Answering	59
5.2.2	Generative Question Answering	60
5.3	Retrieval-Augmented Generation	63
5.4	Gestione dei casi eccezionali	64
5.4.1	Moderazione dei contenuti	64

5.4.2	Zero-shot classification	65
5.4.3	Altri casi eccezionali	66
6	Progettazione e implementazione	67
6.1	Introduzione	67
6.2	Tecnologie e strumenti utilizzati	67
6.3	Progettazione	72
6.3.1	Design pattern utilizzati	73
6.3.1.1	Model-View-Update	74
6.3.1.2	Adapter	75
6.3.1.3	Strategy	77
6.3.1.4	Template method	78
6.3.1.5	Dependency injection	79
6.4	Implementazione	81
6.4.1	MVU con Streamlit	82
6.4.2	Algoritmi di ricerca con PostgreSQL	83
6.4.2.1	Query di ricerca semantica e pgvector	83
6.4.2.2	Query di ricerca lessicale	84
7	Verifica e validazione	87
7.1	Introduzione	87
7.2	Analisi statica	87
7.3	Analisi dinamica	89
7.3.1	Test di unità	90
7.3.2	Test di integrazione	95
7.3.3	Test di sistema	96
7.3.4	Test di regressione	98
7.4	Validazione e accettazione	99
8	Conclusioni	101
8.1	Considerazioni finali	101
8.2	Raggiungimento degli obiettivi	102

INDICE

8.3	Conoscenze acquisite	102
8.4	Possibili miglioramenti futuri	103
8.5	Valutazione personale	104
Acronimi e abbreviazioni		ii
Glossario		iv
Bibliografia		xii

Elenco delle figure

1.1	Logo di Zucchetti S.p.A.	1
3.1	Attori individuati	12
3.2	UC1: Selezione del modello di question answering	13
3.3	UC2: Selezione algoritmo di rank fusion	14
3.4	UC3: Utilizzo del knowledge graph	16
3.5	UC4: Pulizia della cronologia della conversazione	17
3.6	UC5: Accesso alle funzionalità avanzate	18
3.7	UC5.1: Inserimento della password	18
3.8	UC6: Chat	19
3.9	UC6.1: Invio di una richiesta	20
3.10	UC6.2: Visualizzazione della risposta	21
3.11	UC7: Uscire dalla modalità amministratore	23
3.12	UC8: Ricalcolo degli IDF	24
3.13	UC9: Ricalcolo dei vettori di embedding	24
3.14	UC10: Rigenerazione del knowledge graph	25
3.15	UC11: Inserimento di un nuovo documento	25
3.16	UC12: Cancellazione del documento attualmente caricato	26
3.17	UC13: Visualizzazione del nome del documento caricato	27
4.1	Panoramica ad alto livello senza knowledge graph	35
4.2	Panoramica ad alto livello con knowledge graph	35
4.3	Esempio di applicazione del RSF con normalizzazione	49
4.4	Applicazione del RSF sui due ranking precedenti	51

4.5	Applicazione del DBSF sui due ranking precedenti	53
4.6	Esempio di knowledge graph	56
4.7	Rappresentazione della query di attraversamento di un knowledge graph	57
6.1	Logo di Python	68
6.2	Logo di Hugging Face	68
6.3	Logo di Txtai	69
6.4	Logo di Groq	69
6.5	Loghi di NetworkX e Matplotlib	70
6.6	Logo di PostgreSQL	70
6.7	Logo di Psycopg	71
6.8	Logo di PyMuPDF	71
6.9	Logo di Streamlit	72
6.10	Rappresentazione del flusso di dati nell'architettura MVU	75
6.11	Diagramma delle classi del design pattern Object adapter	76
6.12	Diagramma delle classi del design pattern Class adapter	76
6.13	Diagramma delle classi del design pattern Strategy	78
6.14	Diagramma delle classi del design pattern Template method	79
6.15	Tabelle docs e idf	83

Elenco delle tabelle

3.1	Tabella del tracciamento dei requisiti funzionali	31
3.2	Tabella del tracciamento dei requisiti qualitativi	32
3.3	Tabella del tracciamento dei requisiti di vincolo	32
4.1	Dataset di esempio	44
4.2	Risultati della ricerca lessicale per la richiesta “pianta ornamentale” .	44
4.3	Risultati della ricerca semantica per la richiesta “pianta ornamentale”	44
4.4	Risultati della ricerca lessicale per la richiesta “felino dal pelo medio”	45
4.5	Risultati della ricerca semantica per la richiesta “felino dal pelo medio”	45
4.6	Risultati della ricerca con BM25 per la richiesta “lavorare oltre il proprio orario di lavoro normale”	50
4.7	Risultati della ricerca con cosine similarity per la richiesta “lavorare oltre il proprio orario di lavoro normale”	51
7.1	Punteggi assegnati ai file del progetto da pylint	89
7.2	Tabella dei test di unità	94
7.3	Tabella dei test di integrazione	96
7.4	Tabella dei test di sistema	98
7.5	Tabella dei test di regressione	99

Elenco dei codici sorgenti

6.1	Esempio di client senza Dependency injection	80
6.2	Esempio di client con Constructor injection	80
6.3	Esempio di client con Setter injection	81
6.4	Query di ricerca semantica con cosine similarity	84
6.5	Esempio di query di ricerca lessicale	85

Capitolo 1

Introduzione

1.1 L'azienda

L'esperienza di tirocinio svolta dal laureando Meneghini Fabio ha avuto luogo negli uffici della sede padovana dell'azienda Zucchetti S.p.A.¹, ovvero la prima *software house* italiana.

Fondata nel 1978, Zucchetti S.p.A. risponde alle esigenze dei clienti fornendo numerosi servizi ad aziende, professionisti e associazioni di categoria per la gestione del personale, la contabilità, il fisco e la gestione dei processi aziendali (vendite, logistica, magazzino, produzione ecc.). In questo senso, si configura come un significativo punto di riferimento in grado di fornire un valido supporto per la gestione delle più svariate esigenze di *business*.

Tra le varie sedi sparse in tutto il mondo, quella di Padova si occupa di ricerca e sviluppo.



Figura 1.1: Logo di Zucchetti S.p.A.

¹Zucchetti, *il software per il successo di aziende e professionisti*. URL: <https://www.zucchetti.it/it/cms/home.html>.

1.2 Strumenti utilizzati

Gli strumenti utilizzati nello sviluppo del progetto e per la stesura della relativa documentazione sono stati i seguenti:

- **Visual Studio Code**²: conosciuto anche come *VSCode*, è un editor di codice sorgente sviluppato da Microsoft, progettato per essere leggero, veloce e altamente personalizzabile. È uno degli IDE più popolari tra gli sviluppatori;
- **Git**³: è un software per il controllo di versione distribuito utilizzabile da interfaccia a riga di comando, creato da Linus Torvalds nel 2005. *Git* nacque per essere un semplice strumento per facilitare lo sviluppo del *kernel*_G Linux ed è diventato uno degli strumenti di controllo di versione più diffusi;
- **GitHub**⁴: è una piattaforma online che permette agli sviluppatori di creare, archiviare, gestire e condividere il codice sorgente dei loro progetti *open source*_G. Utilizza *Git* per il versionamento del codice, inoltre implementa tutte le *feature* degli *Issue Tracking System (ITS)*_G e per la *Continuous Integration (CI)*_G;
- **LaTeX**⁵: è un linguaggio di marcatura per la preparazione di testi, basato sul programma di composizione tipografica TeX;
- **Overleaf**⁶: è un editor online per la redazione di documenti in linguaggio LaTeX;
- **Draw.io**⁷: è una piattaforma online che permette di disegnare schemi e diagrammi di ogni tipo.

² *Visual Studio Code - Code Editing. Redefined.* URL: <https://code.visualstudio.com>.

³ *Git.* URL: <https://git-scm.com>.

⁴ *GitHub: Let's build from here.* URL: <https://github.com>.

⁵ *LaTeX - A document preparation system.* URL: <https://www.latex-project.org>.

⁶ *Overleaf, Online LaTeX Editor.* URL: <https://it.overleaf.com>.

⁷ *draw.io.* URL: <https://app.diagrams.net>.

1.3 Organizzazione del testo

1.3.1 Struttura del documento

Il presente documento è suddiviso in otto capitoli il cui contenuto è brevemente riassunto in seguito:

il secondo capitolo descrive nel dettaglio in cosa consiste il progetto di stage, analizzando di conseguenza gli eventuali rischi che si possono incontrare durante lo sviluppo di tale progetto;

il terzo capitolo è dedicato all'analisi dei requisiti: vengono descritti dettagliatamente gli attori, tutti i casi d'uso, per poi elencare tutti i requisiti che il prodotto software finale dovrà soddisfare;

il quarto capitolo approfondisce il sistema di *information retrieval*, illustrando il funzionamento delle tecnologie e degli algoritmi impiegati per lo sviluppo di questa parte di progetto, oltre a fornire le ragioni che hanno guidato la scelta di tali soluzioni;

il quinto capitolo approfondisce il processo di generazione della risposta finale da parte del modello di *question answering* tramite la *Retrieval-Augmented Generation* (RAG). Esso inoltre descrive la gestione delle richieste che rappresentano casi particolari;

il sesto capitolo illustra le attività di progettazione e di codifica, descrivendo inoltre tutte le tecnologie e gli strumenti impiegati a tal fine;

il settimo capitolo descrive le attività di verifica e di validazione del prodotto software;

l'ottavo capitolo rappresenta una sintesi finale del lavoro svolto durante il periodo di tirocinio, descrivendo eventuali successi e difficoltà incontrate durante il percorso. Vengono inoltre analizzati i risultati ottenuti rispetto agli obiettivi iniziali così come l'insieme di competenze teoriche e pratiche acquisite nel corso

del progetto. Il documento si conclude con una riflessione critica sull'operato e sulla crescita personale e professionale del laureando Meneghini Fabio.

1.3.2 Convenzioni tipografiche

In merito alla redazione del presente documento, sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzato il seguente stile: *Application Program Interface***G**;
- i termini particolarmente rilevanti in una sezione e quelli in lingua straniera non di uso comune o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*, fatta eccezione per le occorrenze presenti nei titoli delle sezioni o nelle didascalie;
- i comandi di terminale, i frammenti di codice sorgente e i nomi di *file* o *directory* sono evidenziati con il carattere **monospaziato**.

Capitolo 2

Descrizione dello stage

2.1 Introduzione al progetto

Il progetto di stage consiste nello sviluppo di un *chatbot* il cui scopo è quello di guidare gli utenti nell'utilizzo di *software* complessi e che presentano molte funzionalità. In particolare, il prodotto *software* da sviluppare dovrà migliorare la *user experience* dei programmi che risultano molto dispersivi a causa del loro elevato numero di funzionalità esposte, e che quindi sono caratterizzati da una *usabilità* e una *user experience* carente.

2.1.1 User experience e usabilità

Con il termine *user experience*¹ (UX) si intende il modo in cui l'utente percepisce un prodotto, un servizio o un sistema e interagisce con esso. Secondo la definizione data dallo standard *ISO_G* 9241-210, ovvero uno standard che riguarda l'ergonomia e l'interazione uomo-macchina, la UX include tutte le emozioni, convinzioni, preferenze, percezioni, risposte fisiche e psicologiche, comportamenti e risultati degli utenti che si verificano prima, durante e dopo l'utilizzo del prodotto o servizio in questione. L'ISO elenca inoltre tre fattori che influenzano la UX: il sistema, l'utente e il contesto d'uso.

La UX è composta da aspetti *pragmatici* (come la capacità di svolgere un compito

¹*User experience*. URL: https://en.wikipedia.org/wiki/User_experience.

con efficienza, efficacia e soddisfazione) e da aspetti *edonistici* (come il benessere emotivo e l'appagamento sensoriale che deriva dall'utilizzo del prodotto o servizio in questione). L'usabilità corrisponde agli aspetti pragmatici della *user experience*. Essa si sviluppa su cinque assi:

- *comprensibilità*: rappresenta la facilità con cui i concetti del prodotto possono essere compresi, consentendo all'utente di valutare se il *software* sia adeguato alle proprie esigenze;
- *apprendibilità*: indica la capacità del *software* di ridurre l'impegno necessario agli utenti per apprendere il suo utilizzo;
- *operabilità*: è la capacità di mettere in condizione gli utenti di fare uso del prodotto *software* per i propri scopi e controllarne l'uso;
- *attrattività*: è la capacità del *software* di essere piacevole per l'utente che lo utilizza;
- *conformità*: è la capacità dell'applicativo di aderire a standard o convenzioni relativi all'usabilità.

Il *chatbot* sviluppato dovrà essere usato in combinazione al programma di riferimento per aumentarne la comprensibilità (sarà possibile capire più facilmente lo scopo del programma), l'apprendibilità (gli utenti potranno imparare più velocemente ad usare il programma) e l'operabilità (gli utenti saranno in grado di svolgere i compiti in modo più efficiente) grazie alla sua capacità di rispondere alle richieste dell'utente riguardo l'utilizzo di tale prodotto *software*. Ad esempio, il *chatbot* può rispondere ad una richiesta elencando tutti i passaggi da compiere che permettono di svolgere l'operazione richiesta dall'utente.

2.1.2 Funzionamento generale del chatbot

Per raggiungere questo scopo il sistema necessita di conoscere nel dettaglio il programma in questione: quindi, prima di poter formulare una richiesta, l'utente dovrà caricare nel *server* il *file* della documentazione di tale programma, che il

sistema utilizzerà per ricercare al suo interno tutte le informazioni utili per poter costruire una risposta alla domanda posta dall'utente. Tali informazioni verranno recuperate attraverso un sistema di *information retrieval* (descritto nel dettaglio al capitolo 4), e successivamente un modello di intelligenza artificiale (nello specifico un *Large Language Model (LLM)_G*) si occuperà di elaborare una risposta basata sulle informazioni ottenute dalla documentazione che sia quanto più completa e precisa possibile (maggiori dettagli sono disponibili al capitolo 5). Tale risposta sarà inoltre accompagnata dalle fonti consultate dal sistema: insieme, queste due parti formano il messaggio finale che il *chatbot* mostrerà in risposta alla domanda posta dall'utente.

Al fine di migliorare l'efficacia del sistema, questo progetto si propone inoltre di indagare quale sia il miglior modo di strutturare la documentazione per agevolare il recupero delle informazioni e permettere, quindi, la generazione di una risposta migliore.

2.2 Analisi preventiva dei rischi

Durante la fase di analisi iniziale sono stati individuati alcuni potenziali rischi a cui sarà possibile andare incontro. Si è quindi proceduto ad elaborare delle possibili soluzioni per far fronte a tali rischi.

1. Cambiamento dei requisiti in corso di sviluppo

Descrizione: i requisiti del progetto potrebbero cambiare durante lo sviluppo dello stesso. Alcuni dei motivi potrebbero essere delle modifiche alle esigenze del progetto, un errore di analisi iniziale oppure motivi legati alle tecnologie scelte.

Soluzione: è utile adottare l'utilizzo di una *metodologia agile_G* per tollerare i cambiamenti ai requisiti. È inoltre di fondamentale importanza mantenere una comunicazione costante con il tutor aziendale, in modo da procedere con lo sviluppo del progetto con maggior sicurezza e in modo più organizzato.

2. Inesperienza tecnologica

Descrizione: il progetto prevede l'utilizzo di tecnologie e algoritmi di cui non si ha esperienza, rendendo così più impegnativa la fase di implementazione.

Soluzione: il periodo di tirocinio prevede un periodo di formazione iniziale per lo studio delle tecnologie e gli algoritmi da implementare, oltre l'aiuto e il supporto da parte del tutor aziendale.

3. Documentazione assente o insufficiente

Descrizione: le librerie impiegate nello sviluppo del progetto potrebbero essere accompagnate da una documentazione carente o poco esaustiva, portando così ad una maggiore difficoltà nella loro comprensione ed utilizzo.

Soluzione: prima di selezionare una libreria per il progetto, è importante valutare la qualità e la completezza della documentazione disponibile. Se una libreria manca di documentazione adeguata, potrebbe essere preferibile considerare alternative più documentate.

4. Ritardi nello sviluppo

Descrizione: durante lo sviluppo del progetto potrebbero verificarsi imprevisti come problemi tecnici o un'eccessiva complessità del tema affrontato, che potrebbero portare a ritardi nello sviluppo del prodotto.

Soluzione: come da piano di lavoro, sono state riservate, in accordo con il tutor aziendale, due settimane dedicate al *testing* del prodotto per verificarne il corretto funzionamento e per risolvere eventuali problemi o imprevisti riscontrati durante l'implementazione. È utile inoltre intensificare il dialogo con il tutor interno per discutere riguardo possibili soluzioni o rimodulando le attività e le tempistiche in base alle necessità.

5. Sistema di information retrieval poco efficace

Descrizione: è possibile che il sistema di *information retrieval* sviluppato non recuperi dei risultati del tutto coerenti con quanto richiesto dall'utente.

Soluzione: è importante verificare se il problema sorge da uno degli algoritmi di ricerca, dall'algoritmo di *rank fusion* scelto o dalla strategia di *chunking* della documentazione adottata, per poi valutare e testare delle alternative all'algoritmo che causa il problema.

6. Risposta del chatbot insoddisfacente

Descrizione: il *chatbot* potrebbe produrre delle risposte che sono totalmente o parzialmente sbagliate rispetto a quanto scritto nella documentazione caricata nell'applicativo.

Soluzione: è possibile valutare e testare delle alternative al modello di *question answering* adottato. È inoltre importante sottolineare che la causa di questo problema potrebbe essere che il sistema di *information retrieval* non lavora nel modo desiderato, la cui soluzione è stata discussa al punto precedente.

Capitolo 3

Analisi dei requisiti

3.1 Introduzione

L'analisi dei requisiti è un'attività preliminare allo sviluppo di un sistema software, il cui scopo è quello di definire le funzionalità che il nuovo prodotto deve offrire, ovvero i requisiti che devono essere soddisfatti dal software sviluppato. L'analisi dei requisiti è di fondamentale importanza, in quanto essa guida le successive fasi di sviluppo del prodotto¹.

I requisiti fondamentali che il prodotto finale doveva soddisfare (obiettivi dello *stage*) sono stati concordati assieme al tutor aziendale durante un incontro precedente all'inizio dello stage. Successivamente, tali requisiti sono stati ampliati e raffinati in rapporto alle possibilità emerse durante lo studio e l'approfondimento delle tecnologie e i possibili algoritmi da utilizzare per la realizzazione dell'applicativo.

3.2 Obiettivi dello stage

Gli obiettivi fondamentali da raggiungere durante il periodo di tirocinio, stilati in accordo con il tutor aziendale ed inseriti nel documento *Piano di Lavoro*, sono identificati dalla seguente notazione:

¹Ian Sommerville. *Software Engineering*. Pearson Education, 2016.

OO: requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;

OD: requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto.

Alle sigle precedentemente indicate seguirà un numero progressivo, identificando così tutti gli obiettivi.

Essi sono i seguenti:

- **OO1:** realizzazione dell'interfaccia del *chatbot*;
- **OO2:** realizzazione di un *box* utile a contenere le fonti delle informazioni fornite dal *chatbot*;
- **OO3:** realizzazione della struttura del messaggio di risposta del *chatbot*, ovvero un elenco numerato contenente la sequenza delle azioni da svolgere per compiere l'operazione richiesta dall'utente;
- **OO4:** definizione della struttura della documentazione in linguaggio Markdown per segnalare al LLM i possibili comandi;
- **OO5:** definizione di un metodo di test per stabilire la correttezza delle operazioni proposte dal *chatbot*;
- **OO6:** valutazione della correttezza delle operazioni proposte dal *chatbot*;
- **OD7:** integrazione con strumenti di Zucchetti S.p.A. per segnalare le azioni da svolgere direttamente nell'interfaccia grafica dell'applicazione in questione;
- **OD8:** *fine-tuning_G* significativo dei modelli selezionati.

3.3 Attori

L'individuazione degli attori è la prima fase dell'analisi dei requisiti. Gli attori sono gli utilizzatori finali del software prodotto: essi possono avere diversi ruoli e livelli di competenza, e le loro esigenze e aspettative devono essere comprese a fondo

per poter realizzare un'efficace analisi dei requisiti².

Gli attori individuati sono due:

- l'*utente ordinario*, ovvero un utente che ha accesso alle funzionalità di base dell'applicazione;
- l'*utente amministratore*, ovvero un utente che, oltre ad avere accesso alle funzionalità di base, può anche accedere alle funzionalità più avanzate.

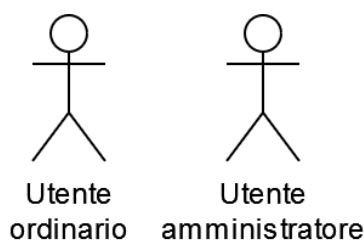


Figura 3.1: Attori individuati

3.4 Casi d'uso

Per la definizione dei requisiti sono stati realizzati dei diagrammi dei casi d'uso (in inglese *use case diagrams*), ovvero dei diagrammi *Unified Modeling Language (UML)*³ dedicati alla descrizione delle funzionalità e dei servizi offerti dal sistema, così come sono percepiti e utilizzati dagli attori che interagiscono con esso.

Ciascun caso d'uso riporta gli attori coinvolti, le sue precondizioni, la sua descrizione, le sue postcondizioni ed eventuali sottocasi d'uso, inclusioni, specializzazioni e scenari alternativi.

I casi d'uso che sono stati definiti sono i seguenti:

²Sommerville, *Software Engineering*.

³Grady Booch, James Rumbaugh e Ivar Jacobson. *Unified Modeling Language User Guide*. Addison-Wesley, 1999.

UC1: Selezione modello di question answering

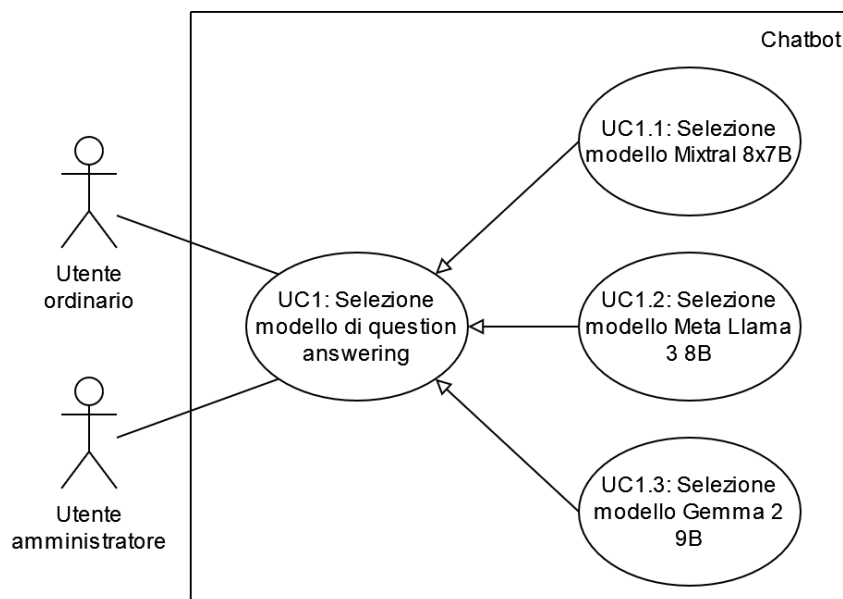


Figura 3.2: UC1: Selezione del modello di question answering

Attori Principali: Utente ordinario, Utente amministratore.

Precondizioni: L'utente ha avviato l'applicativo e sta visualizzando la pagina principale.

Descrizione: L'utente può selezionare il modello di *question answering* che verrà utilizzato nella generazione della risposta del *chatbot*.

Postcondizioni: Le modifiche vengono salvate nel sistema.

Specializzazioni: UC1.1, UC1.2, UC1.3.

UC1.1: Selezione modello Mixtral 8x7B

Attori Principali: Utente ordinario, Utente amministratore.

Precondizioni: L'utente non ha già selezionato il modello Mixtral 8x7B.

Descrizione: L'utente può selezionare Mixtral 8x7B come modello di *question answering*.

Postcondizioni: Le modifiche vengono salvate nel sistema.

UC1.2: Selezione modello Meta Llama 3 8B

Attori Principali: Utente ordinario, Utente amministratore.

Precondizioni: L'utente non ha già selezionato il modello Meta Llama 3 8B.

Descrizione: L'utente può selezionare Llama 3 8B come modello di *question answering*.

Postcondizioni: Le modifiche vengono salvate nel sistema.

UC1.3: Selezione modello Gemma 2 9B

Attori Principali: Utente ordinario, Utente amministratore.

Precondizioni: L'utente non ha già selezionato il modello Gemma 2 9B.

Descrizione: L'utente può selezionare Gemma 2 9B come modello di *question answering*.

Postcondizioni: Le modifiche vengono salvate nel sistema.

UC2: Selezione algoritmo di rank fusion

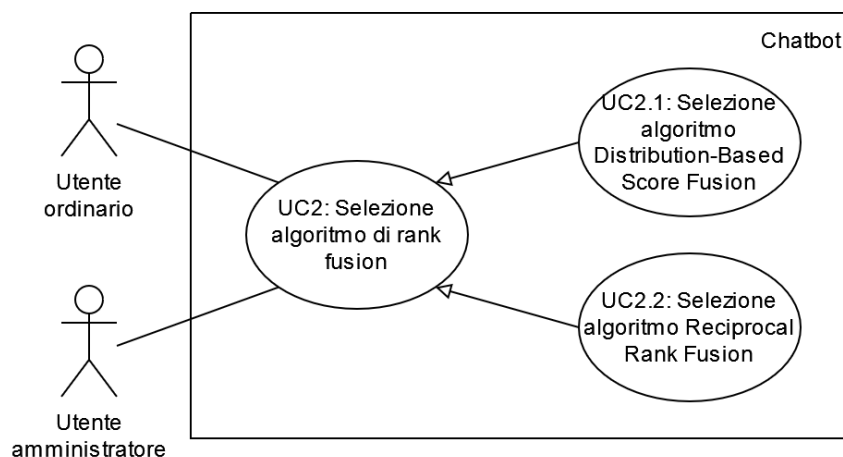


Figura 3.3: UC2: Selezione algoritmo di rank fusion

Attori Principali: Utente ordinario, Utente amministratore.

Precondizioni: L'utente ha avviato l'applicativo e sta visualizzando la pagina principale.

Descrizione: L'utente può selezionare l'algoritmo di *rank fusion* che verrà utilizzato dal sistema per la ricerca dei documenti più rilevanti con la richiesta dell'utente.

Postcondizioni: Le modifiche vengono salvate nel sistema.

Specializzazioni: UC2.1, UC2.2.

UC2.1: Selezione algoritmo Distribution-Based Score Fusion

Attori Principali: Utente ordinario, Utente amministratore.

Precondizioni: L'utente non ha già selezionato l'algoritmo *Distribution-Based Score Fusion*.

Descrizione: L'utente può selezionare *Distribution-Based Score Fusion* come algoritmo di *rank fusion*.

Postcondizioni: Le modifiche vengono salvate nel sistema.

UC2.2: Selezione algoritmo Reciprocal Rank Fusion

Attori Principali: Utente ordinario, Utente amministratore.

Precondizioni: L'utente non ha già selezionato l'algoritmo *Reciprocal Rank Fusion*.

Descrizione: L'utente può selezionare *Reciprocal Rank Fusion* come algoritmo di *rank fusion*.

Postcondizioni: Le modifiche vengono salvate nel sistema.

UC3: Utilizzo del knowledge graph

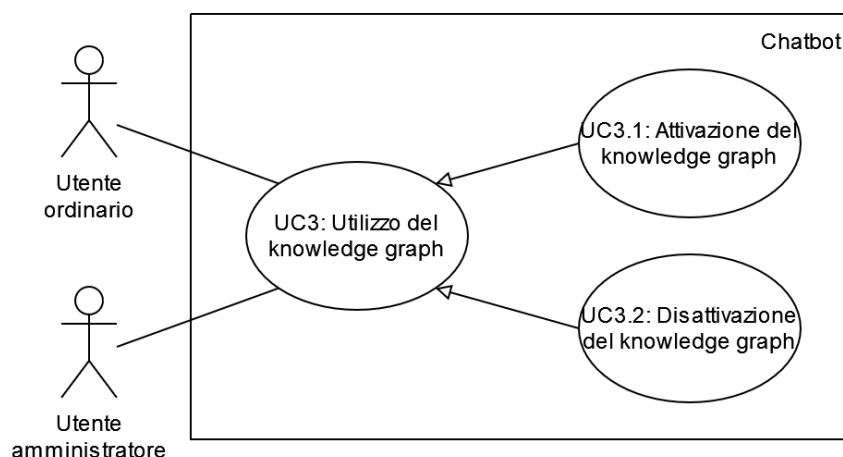


Figura 3.4: UC3: Utilizzo del knowledge graph

Attori Principali: Utente ordinario, Utente amministratore.

Precondizioni: L'utente ha avviato l'applicativo e sta visualizzando la pagina principale.

Descrizione: L'utente può scegliere se utilizzare o meno il *knowledge graph* per permettere al sistema una migliore selezione dei documenti più rilevanti con la richiesta dell'utente.

Postcondizioni: Le modifiche vengono salvate nel sistema.

Specializzazioni: UC3.1, UC3.2.

UC3.1: Attivazione del knowledge graph

Attori Principali: Utente ordinario, Utente amministratore.

Precondizioni: L'utente non ha già attivato la ricerca con *knowledge graph*.

Descrizione: L'utente può attivare l'utilizzo del *knowledge graph* per la ricerca.

Postcondizioni: Le modifiche vengono salvate nel sistema.

UC3.2: Disattivazione del knowledge graph

Attori Principali: Utente ordinario, Utente amministratore.

Precondizioni: L'utente non ha già disattivato la ricerca con *knowledge graph*.

Descrizione: L'utente può disattivare l'utilizzo del *knowledge graph* per la ricerca.

Postcondizioni: Le modifiche vengono salvate nel sistema.

UC4: Pulizia della cronologia della conversazione

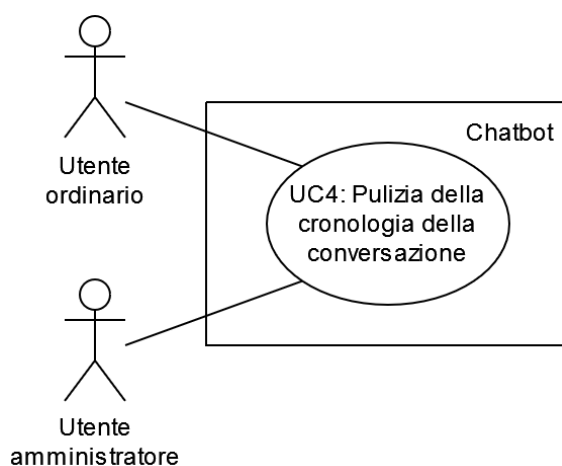


Figura 3.5: UC4: Pulizia della cronologia della conversazione

Attori Principali: Utente ordinario, Utente amministratore.

Precondizioni: L'utente ha inviato una o più richieste al *chatbot*.

Descrizione: L'utente può pulire la cronologia della conversazione, ovvero la lista delle sue richieste al *chatbot* e le relative risposte ricevute.

Postcondizioni: La cronologia della *chat* viene eliminata, quindi i messaggi precedentemente inviati e ricevuti non vengono più visualizzati nell'interfaccia grafica.

UC5: Accesso alle funzionalità avanzate

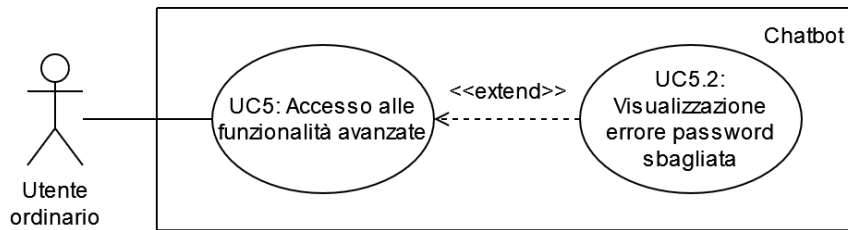


Figura 3.6: UC5: Accesso alle funzionalità avanzate

Attori Principali: Utente ordinario.

Precondizioni: L'utente non ha già effettuato l'accesso per le funzionalità avanzate.

Descrizione: L'utente può accedere alle funzionalità avanzate dopo l'inserimento di una *password*.

Postcondizioni: L'utente ottiene l'accesso alle funzionalità avanzate.

Sottocasi d'uso: UC5.1.

Scenari Alternativi: UC5.2.

UC5.1: Inserimento della password

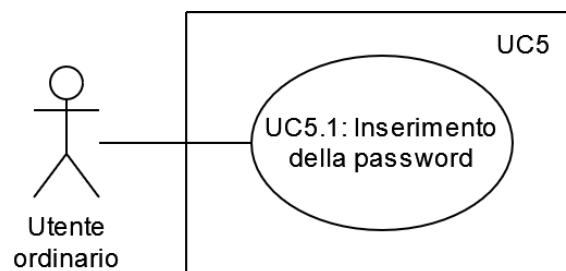


Figura 3.7: UC5.1: Inserimento della password

Attori Principali: Utente ordinario.

Precondizioni: L'utente non ha già effettuato l'accesso per le funzionalità avanzate.

Descrizione: L'utente può inserire la *password* per poter accedere alla sezione dedicata alle funzionalità avanzate.

Postcondizioni: L'utente ha inserito la *password* per l'accesso alle funzionalità avanzate.

UC5.2: Visualizzazione errore password sbagliata

Attori Principali: Utente ordinario.

Precondizioni: L'utente ha tentato di effettuare l'accesso alle funzionalità avanzate.

Descrizione: Se l'utente inserisce una *password* sbagliata, il sistema visualizza un opportuno messaggio di errore.

Postcondizioni: Viene visualizzato un messaggio di errore per informare l'utente che la *password* inserita è sbagliata.

UC6: Chat

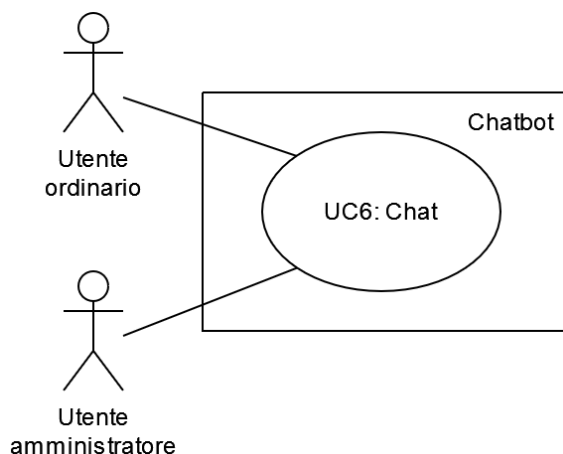


Figura 3.8: UC6: Chat

Attori Principali: Utente ordinario, utente amministratore.

Precondizioni: L'utente ha avviato l'applicativo e sta visualizzando la pagina principale.

Descrizione: L'utente può inviare richieste al *chatbot* ed ottenere le rispettive risposte, ciascuna accompagnata dalle fonti consultate dal sistema per generare tale risposta.

Postcondizioni: L'utente ha inviato una richiesta ed ha ottenuto una risposta dal *chatbot*.

Sottocasi d'uso: UC6.1, UC6.2.

UC6.1: Invio di una richiesta

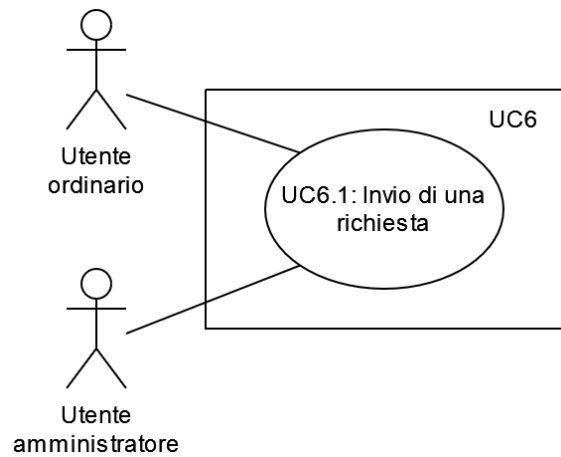


Figura 3.9: UC6.1: Invio di una richiesta

Attori Principali: Utente ordinario, utente amministratore.

Precondizioni: L'utente ha avviato l'applicativo e sta visualizzando la pagina principale.

Descrizione: L'utente può inviare una richiesta al *chatbot*.

Postcondizioni: La richiesta inviata dall'utente è stata correttamente presa in carico dal sistema.

UC6.2: Visualizzazione della risposta

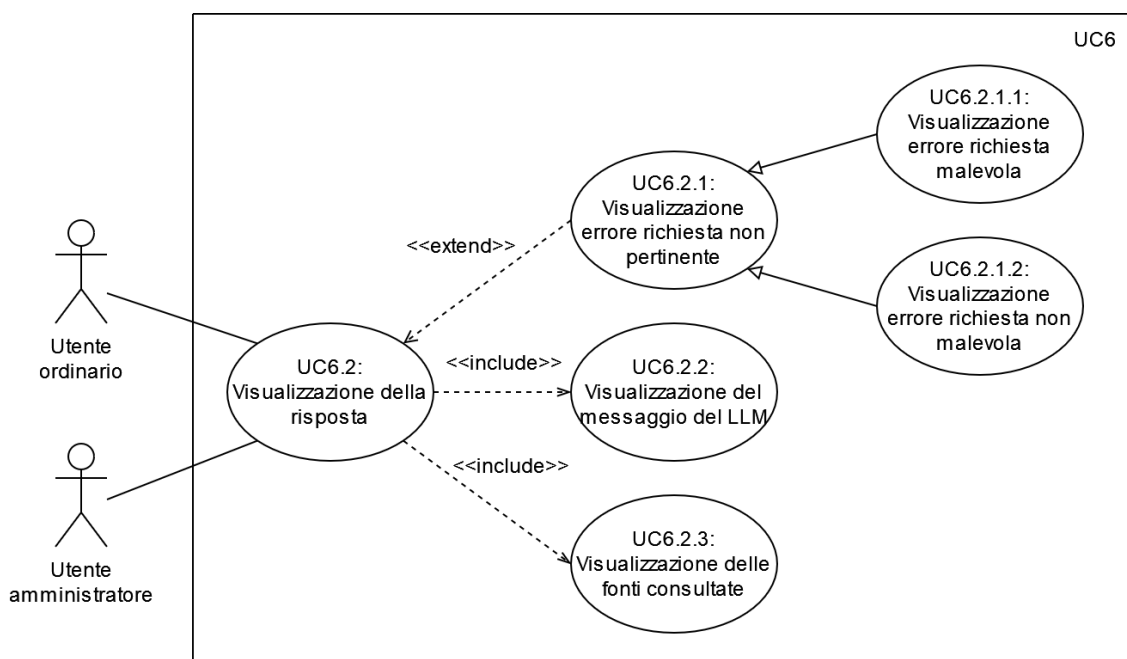


Figura 3.10: UC6.2: Visualizzazione della risposta

Attori Principali: Utente ordinario, utente amministratore.

Precondizioni: L'utente ha inviato una richiesta al *chatbot*.

Descrizione: Dopo aver inviato una richiesta al *chatbot*, l'utente può visualizzare il relativo messaggio di risposta, che viene generato dal modello di *question answering* in base al contesto fornito dalla ricerca precedentemente effettuata dal sistema.

Postcondizioni: Viene visualizzato un messaggio di risposta.

Inclusioni: UC6.2.2, UC6.2.3.

Scenari Alternativi: UC6.2.1.

UC6.2.1: Visualizzazione errore richiesta non pertinente

Attori Principali: Utente ordinario, utente amministratore.

Precondizioni: L'utente ha inviato al *chatbot* una richiesta non pertinente con il documento caricato.

Descrizione: Nel caso il sistema ritenga che la richiesta inviata dall'utente non sia

pertinente con il documento caricato, il *chatbot* risponde con un messaggio di errore, informando l'utente dell'accaduto.

Postcondizioni: Viene visualizzato un opportuno messaggio di errore come risposta dal *chatbot*.

Specializzazioni: UC6.2.1.1, UC6.2.1.2.

UC6.2.1.1: Visualizzazione errore richiesta malevola

Attori Principali: Utente ordinario, utente amministratore.

Precondizioni: L'utente ha inviato al *chatbot* una richiesta malevola.

Descrizione: Nel caso il sistema ritenga che la richiesta inviata dall'utente contenga del contenuto malevolo, il *chatbot* risponde con un messaggio di errore, elencando tutte le motivazioni per cui non è stato possibile generare una risposta.

Postcondizioni: Viene visualizzato un messaggio di errore che contiene l'elenco di tutte le motivazioni per cui non è stato possibile generare una risposta.

UC6.2.1.2: Visualizzazione errore richiesta non malevola

Attori Principali: Utente ordinario, utente amministratore.

Precondizioni: L'utente ha inviato al *chatbot* una richiesta non pertinente con il documento caricato ma non malevola.

Descrizione: Nel caso il sistema ritenga che la richiesta inviata dall'utente non sia pertinente con il documento caricato e non contenga del contenuto malevolo, il *chatbot* risponde con un messaggio di errore, informando l'utente dell'accaduto.

Postcondizioni: Viene visualizzato un opportuno messaggio di errore come risposta dal *chatbot*.

UC6.2.2: Visualizzazione del messaggio del LLM

Attori Principali: Utente ordinario, utente amministratore.

Precondizioni: L'utente ha inviato una richiesta al *chatbot*.

Descrizione: Dopo aver ricevuto la richiesta, il *chatbot* visualizza la risposta gene-

rata dal modello di *question answering* basata sul contesto.

Postcondizioni: Viene visualizzata la risposta generata dal modello.

UC6.2.3: Visualizzazione delle fonti consultate

Attori Principali: Utente ordinario, utente amministratore.

Precondizioni: L'utente ha inviato una richiesta al *chatbot*.

Descrizione: Dopo aver ricevuto la richiesta, il *chatbot* visualizza le fonti consultate (cioè i documenti che compongono il contesto) per costruire il messaggio di risposta del modello di *question answering*.

Postcondizioni: Viene visualizzato un messaggio contenente le fonti consultate per la generazione della risposta.

UC7: Uscire dalla modalità amministratore

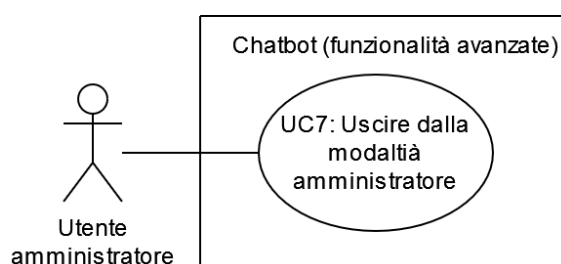


Figura 3.11: UC7: Uscire dalla modalità amministratore

Attori Principali: Utente amministratore.

Precondizioni: L'utente ha effettuato l'accesso alle funzionalità avanzate.

Descrizione: L'utente, dopo aver fatto l'accesso alle funzionalità avanzate, ha la possibilità di uscire da tale sezione.

Postcondizioni: L'utente smette di visualizzare le funzionalità avanzate, e al loro posto viene mostrato nuovamente il *form* per l'accesso.

UC8: Ricalcolo degli IDF

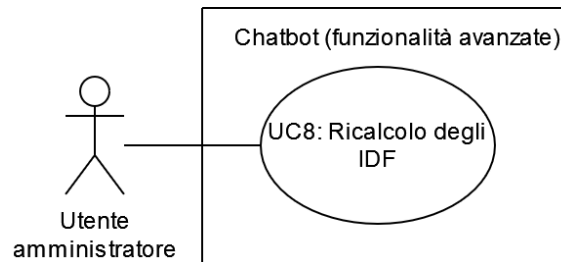


Figura 3.12: UC8: Ricalcolo degli IDF

Attori Principali: Utente amministratore.

Precondizioni: L'utente ha effettuato l'accesso alle funzionalità avanzate.

Descrizione: L'utente amministratore ha la possibilità di ricalcolare gli IDF delle parole che compaiono nel *file* caricato.

Postcondizioni: Gli IDF vengono ricalcolati e salvati nel *database* sovrascrivendo quelli preesistenti.

UC9: Ricalcolo dei vettori di embedding

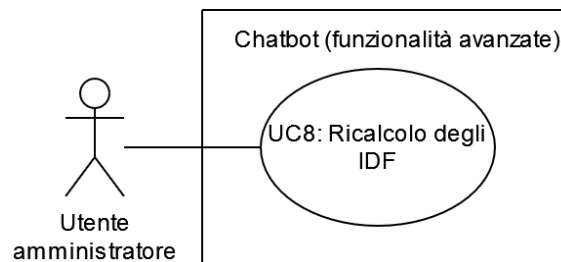


Figura 3.13: UC9: Ricalcolo dei vettori di embedding

Attori Principali: Utente amministratore.

Precondizioni: L'utente ha effettuato l'accesso alle funzionalità avanzate.

Descrizione: L'utente amministratore ha la possibilità di ricalcolare i vettori di *embedding* delle sezioni del *file* caricato.

Postcondizioni: I vettori di *embedding* vengono ricalcolati e salvati nel *database* sovrascrivendo quelli preesistenti.

UC10: Rigenerazione del knowledge graph

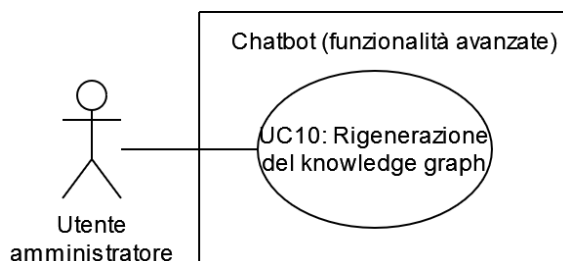


Figura 3.14: UC10: Rigenerazione del knowledge graph

Attori Principali: Utente amministratore.

Precondizioni: L'utente ha effettuato l'accesso alle funzionalità avanzate.

Descrizione: L'utente amministratore ha la possibilità di rigenerare il *knowledge graph* relativo al *file* caricato.

Postcondizioni: Il *knowledge graph* viene rigenerato e salvato sul disco sovrascrivendo quello preesistente.

UC11: Inserimento di un nuovo documento

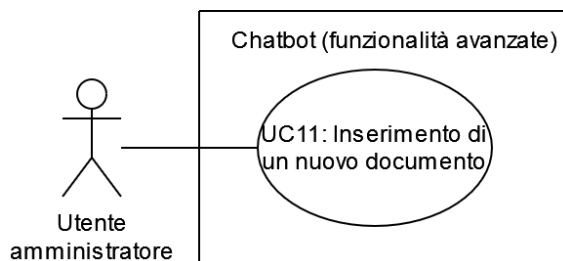


Figura 3.15: UC11: Inserimento di un nuovo documento

Attori Principali: Utente amministratore.

Precondizioni: L'utente ha effettuato l'accesso alle funzionalità avanzate.

Descrizione: L'utente amministratore ha la possibilità di inserire un nuovo documento, al quale il *chatbot* farà riferimento per la generazione delle risposte alle prossime richieste inviate dall'utente.

Postcondizioni: Il nuovo documento viene caricato nel *server*, eventualmente sovrascrivendo quello preesistente.

UC12: Cancellazione del documento attualmente caricato

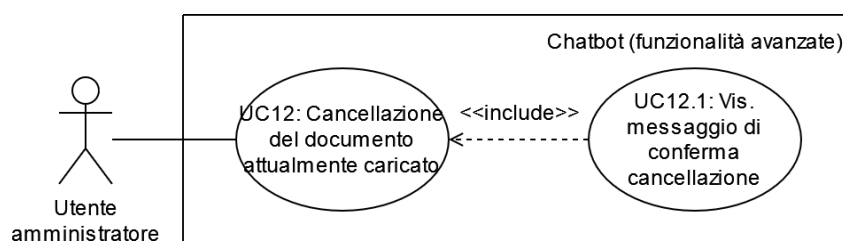


Figura 3.16: UC12: Cancellazione del documento attualmente caricato

Attori Principali: Utente amministratore.

Precondizioni: L'utente ha effettuato l'accesso alle funzionalità avanzate ed è stato caricato un documento nel *server* in precedenza.

Descrizione: L'utente ha la possibilità di eliminare il documento attualmente caricato nel *server*, dopo un opportuno messaggio di conferma.

Postcondizioni: Il documento viene eliminato dal *server*.

Inclusioni: UC12.1.

UC12.1: Visualizzazione messaggio di conferma di cancellazione

Attori Principali: Utente amministratore.

Precondizioni: L'utente amministratore vuole eliminare il documento attualmente caricato nel *server*.

Descrizione: Il sistema mostra un messaggio di conferma nel caso l'utente ammi-

nistratore voglia eliminare il documento caricato nel *server*.

Postcondizioni: Viene visualizzato il messaggio di conferma.

UC13: Visualizzazione del nome del documento caricato

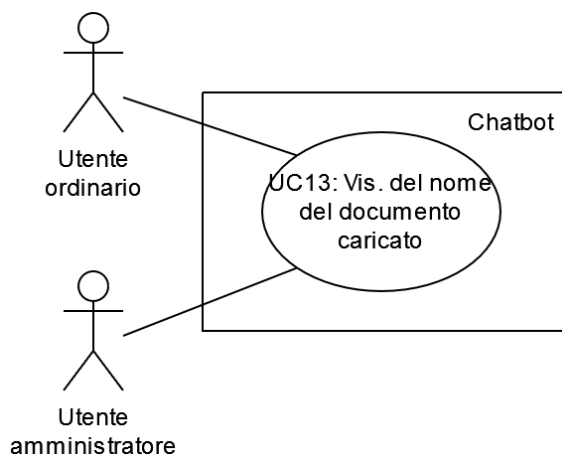


Figura 3.17: UC13: Visualizzazione del nome del documento caricato

Attori Principali: Utente ordinario, utente amministratore.

Precondizioni: È stato caricato un documento nel *server* in precedenza.

Descrizione: L'utente ha la possibilità di visualizzare il nome del documento a cui viene fatto riferimento per la generazione delle risposte da parte del *chatbot*.

Postcondizioni: Viene visualizzato il nome del documento attualmente caricato nel *server*.

3.5 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli *use case* effettuata sul progetto è stata stilata una tabella che traccia i requisiti in rapporto ai casi d'uso, ampliando così i requisiti che sono stati inizialmente definiti in accordo con il tutor aziendale. Ciascun requisito è identificato da un codice strutturato nel seguente modo:

$$R[\text{importanza}][\text{tipo}]-[\text{numero}]$$

dove

- l'importanza può essere **O** per i requisiti obbligatori oppure **D** per quelli desiderabili;
- il tipo può essere **F** per i requisiti funzionali, **Q** per quelli qualitativi oppure **V** per quelli di vincolo;
- il numero è un valore numerico progressivo che identifica univocamente un requisito.

Nelle tabelle 3.1, 3.2 e 3.3 sono riassunti, rispettivamente, i requisiti funzionali, qualitativi e di vincolo, e l'eventuale loro tracciamento con gli *use case* delineati in fase di analisi.

3.5.1 Tabelle dei requisiti

Requisito	Descrizione	Fonte
ROF-1	L'interfaccia deve poter permettere all'utente di selezionare il modello di <i>question answering</i>	UC1, UC1.1, UC1.2, UC1.3
ROF-2	L'interfaccia deve poter permettere all'utente di selezionare l'algoritmo di <i>rank fusion</i>	UC2, UC2.2, UC2.3
ROF-3	L'interfaccia deve poter permettere la scelta di utilizzare o meno il <i>knowledge graph</i>	UC3, UC3.1, UC3.2
ROF-4	L'interfaccia deve presentare un pulsante che permette di eliminare la cronologia della conversazione, nel caso l'utente abbia inviato almeno una richiesta al <i>chatbot</i>	UC4
Continua nella prossima pagina...		

Tabella 3.1 – Continuazione della tabella

Requisito	Descrizione	Fonte
ROF-5	L'interfaccia deve presentare un pulsante tramite il quale è possibile effettuare l'accesso alle funzionalità avanzate dopo l'inserimento di una <i>password</i>	UC5, UC5.1
ROF-6	Se l'utente tenta di effettuare l'accesso alle funzionalità avanzate dopo aver inserito una <i>password</i> non corretta, l'interfaccia deve mostrare un relativo messaggio di errore	UC5.2
ROF-7	L'interfaccia deve presentare una <i>chat</i> con la quale interagire con il LLM	UC6, UC6.1, UC6.2
ROF-8	Nella risposta fornita dal LLM nella <i>chat</i> devono essere presenti il messaggio prodotto da quest'ultimo così come le fonti consultate per produrlo	UC6.2.2, UC6.2.3
ROF-9	Nel caso la richiesta effettuata dall'utente non sia pertinente con il documento caricato, il <i>chatbot</i> deve rispondere con un relativo messaggio di errore	UC6.2.1.2
ROF-10	Nel caso la richiesta effettuata dall'utente contenga dei contenuti malevoli, il <i>chatbot</i> deve rispondere con un messaggio che spieghi le ragioni per cui non è possibile generare una risposta	UC6.2.1.1
ROF-11	Nel caso l'utente abbia effettuato l'accesso alle funzionalità avanzate, l'interfaccia deve presentare un pulsante tramite il quale è possibile uscire dalla modalità amministratore	UC7
Continua nella prossima pagina...		

Tabella 3.1 – Continuazione della tabella

Requisito	Descrizione	Fonte
ROF-12	Nel caso l'utente abbia effettuato l'accesso alle funzionalità avanzate, l'interfaccia deve presentare un pulsante tramite il quale è possibile ricalcolare gli IDF delle parole che compongono il documento caricato	UC8
ROF-13	Nel caso l'utente abbia effettuato l'accesso alle funzionalità avanzate, l'interfaccia deve presentare un pulsante tramite il quale è possibile ricalcolare i vettori di <i>embedding</i> delle sezioni del documento caricato	UC9
ROF-14	Nel caso l'utente abbia effettuato l'accesso alle funzionalità avanzate, l'interfaccia deve presentare un pulsante tramite il quale è possibile rigenerare il <i>knowledge graph</i> relativo al documento caricato	UC10
ROF-15	Nel caso l'utente abbia effettuato l'accesso alle funzionalità avanzate, l'interfaccia deve presentare un <i>widget</i> con il quale è possibile caricare un nuovo documento nel <i>server</i>	UC11
ROF-16	Nel caso l'utente abbia effettuato l'accesso alle funzionalità avanzate e sia stato caricato un documento in precedenza, l'interfaccia deve presentare un pulsante tramite il quale è possibile eliminare il documento caricato, dopo un opportuno messaggio di conferma	UC12, UC12.1
Continua nella prossima pagina...		

Tabella 3.1 – Continuazione della tabella

Requisito	Descrizione	Fonte
ROF-17	Nel caso sia stato caricato un documento in precedenza, l'interfaccia dell'applicativo deve mostrare il suo nome. Se non è presente nessun documento nel <i>server</i> , l'interfaccia deve mostrare il messaggio “ <i>Nessun documento caricato</i> ”	UC13, UC13.1

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione	Fonte
ROQ-1	L'applicativo sviluppato deve essere accessibile da tutti i maggiori <i>browser</i>	Azienda
ROQ-2	Il lavoro svolto deve essere opportunamente documentato	Azienda
ROQ-3	Il codice prodotto deve essere coperto dai <i>test</i> di unità	Azienda
ROQ-4	Le ricerche effettuate dal sistema devono produrre un risultato in tempi ragionevoli	Azienda
ROQ-5	Le risposte prodotte dal <i>chatbot</i> devono essere testate, dopo aver definito un opportuno metodo di <i>test</i>	Azienda, OO5, OO6
RDQ-6	L'applicativo sviluppato deve essere fruibile comodamente anche da <i>mobile</i>	Azienda
RDQ-7	L'applicativo sviluppato deve poter essere integrato in alcuni strumenti di Zucchetti S.p.A per segnalare le azioni da svolgere direttamente nell'interfaccia di tali strumenti	OD7
Continua nella prossima pagina...		

Tabella 3.2 – Continuazione della tabella

Requisito	Descrizione	Fonte
RDQ-8	I modelli utilizzabili nell'applicativo devono essere soggetti a <i>fine-tuning</i> per migliorare la qualità delle risposte prodotte	OD8

Tabella 3.2: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Fonte
ROV-1	L'applicativo sviluppato deve essere un'applicazione <i>web</i>	Azienda
ROV-2	Gli algoritmi di ricerca lessicale e di ricerca semantica devono essere eseguiti come <i>query</i> in un database PostgreSQL	Azienda

Tabella 3.3: Tabella del tracciamento dei requisiti di vincolo

Capitolo 4

Information retrieval

4.1 Introduzione

Per essere in grado di rispondere alle domande poste dall'utente, il sistema dovrà fornire al LLM la documentazione caricata nel *server* come contesto, da allegare al *prompt*: tuttavia, tali modelli presentano dei limiti riguardo la lunghezza massima del *prompt*, che con il testo di una documentazione completa verrebbero ampiamente oltrepassati. È quindi necessario sviluppare un sistema che permetta di allegare al *prompt* del LLM soltanto le parti della documentazione che sono utili per rispondere alla richiesta avanzata dall'utente, cercandole all'interno del testo completo del documento caricato, ovvero un sistema di *information retrieval*.

Nell'informatica e nelle scienze dell'informazione, l'*information retrieval* (IR) è il processo di identificazione e recupero delle risorse di un sistema informativo che rispondono ad una determinata richiesta effettuata dall'utente, espressa sotto forma di una *query* di ricerca. Nel caso del recupero di documenti o di parti di essi, tali *query* possono essere basate sul testo o su altri tipi di indicizzazione basati sul contenuto¹.

Lo sviluppo di un sistema di *information retrieval* efficace è stato il principale oggetto del progetto di stage in questione. Infatti, il recupero delle informazioni più rilevanti con la richiesta (o *query*) dell'utente è fondamentale per far produrre al

¹ *What is Information Retrieval? | A Comprehensive Information Retrieval (IR) Guide*. URL: <https://www.elastic.co/what-is/information-retrieval>.

LLM una risposta che sia quanto più esaustiva, completa e dettagliata possibile.

4.2 Panoramica ad alto livello del sistema

Per effettuare la ricerca delle parti del documento più pertinenti con la richiesta dell'utente vengono utilizzati due diversi algoritmi di ricerca: il *Best Match 25*, che effettua una ricerca per *keywords* (lessicale), e una ricerca semantica con calcolo della *cosine similarity* tra il vettore di *embedding* della *query* dell'utente e i vettori relativi alle singole sezioni della documentazione dell'applicativo in questione. Questi due algoritmi assegnano un punteggio di pertinenza (calcolato su criteri diversi) a ciascuna sezione del documento caricato in relazione alla richiesta dell'utente, e producono come *output* due differenti liste di risultati (anche chiamate *ranking*), ordinate in modo decrescente rispetto al punteggio ottenuto dai due algoritmi. I due *ranking* vengono utilizzati come *input* per un terzo algoritmo, detto di *rank fusion*, che, come suggerisce il nome, ha il compito di “fonderli” per ottenere in *output* un nuovo *ranking* basato su entrambi i criteri degli algoritmi di ricerca precedentemente utilizzati, quindi potenzialmente più preciso. In questo modo, è possibile intercettare in modo efficace sia una richiesta basata su *keywords* che una di natura più semantica.

Opzionalmente, i risultati con il punteggio più alto nella lista finale possono essere utilizzati per formulare una *query* sul *knowledge graph* della documentazione, ovvero una struttura dati a grafo (che nel contesto di questo progetto è non orientato) i cui nodi rappresentano le sezioni e gli archi che li collegano rappresentano le relazioni semantiche che sussistono tra loro. Tale *query* recupera i nodi che vengono attraversati dai tre migliori percorsi che collegano i nodi precedentemente selezionati, in modo tale da ottenere tutte le informazioni rilevanti con la richiesta iniziale dell'utente.

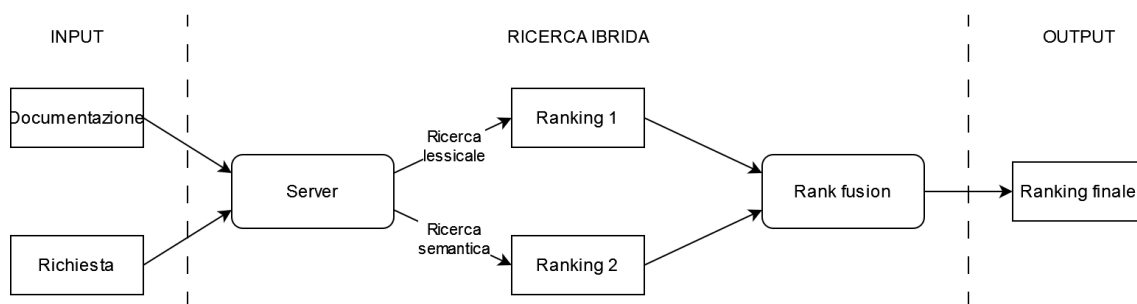


Figura 4.1: Panoramica ad alto livello senza knowledge graph

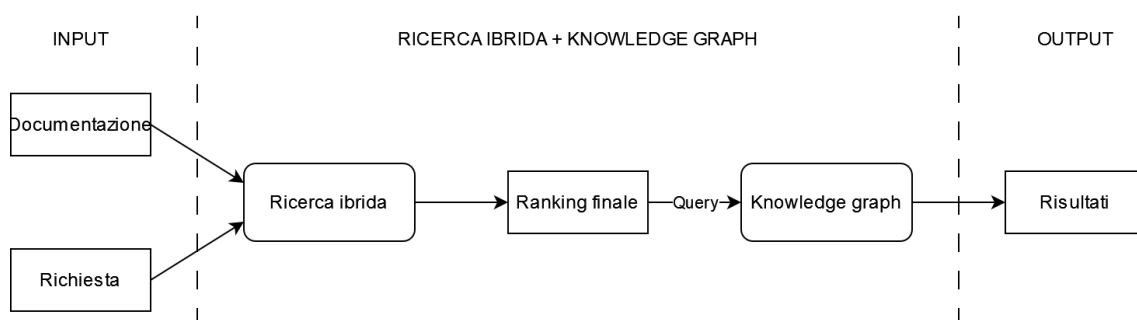


Figura 4.2: Panoramica ad alto livello con knowledge graph

4.3 Data preparation

Prima di caricare un documento all'interno dell'applicativo è necessario che esso rispetti un determinato formato, con il fine di rendere più agevole la ricerca delle informazioni al suo interno. Questo processo prende il nome di *data preparation*.

La *data preparation*² è il processo di raccolta, pulizia, trasformazione e organizzazione dei dati per renderli utilizzabili in contesti analitici e operativi. Questo processo può coinvolgere diverse fasi, che includono la raccolta di dati grezzi da varie fonti, la loro pulizia da errori e incongruenze e l'organizzazione degli stessi in formati adeguati all'analisi. La preparazione dei dati è un processo cruciale al fine di garantire una qualità dei dati adeguata ad estrarre informazioni valide.

Nell'applicativo sviluppato è possibile inserire due diversi formati di *file* di documentazione, ovvero PDF e SAM, che necessitano di una *data preparation* dedicata,

²A *Fresh Look at Data Preparation*. URL: <https://www.iri.com/blog/business-intelligence/a-fresh-look-at-data-preparation/>.

che tenga conto delle caratteristiche dei *file* tipici di quei formati per effettuare il *chunking* (descritto alla sezione successiva), ovvero la suddivisione del testo, nel modo più agevole ed efficace possibile.

I requisiti che i *file* PDF devono rispettare sono i seguenti:

- i titoli delle sezioni devono essere scritti in grassetto;
- i titoli delle sezioni devono avere un *font* con dimensione non inferiore ai 13 pt;
- la dimensione del *font* dei titoli deve essere strettamente maggiore a quella del *font* del contenuto delle sezioni;
- il contenuto del primo e dell'ultimo decimo di ciascuna pagina che compone il *file* non verrà considerato: quello spazio dovrà quindi essere dedicato all'intestazione e al *footer*.

Per quanto riguarda i *file* SAM, invece, è sufficiente strutturare il documento in modo tale da utilizzare correttamente i *tag header* [H1], [H2], [H3], [H4], [H5] e [H6], dato che la suddivisione avviene in base ad essi. Per entrambi i formati, inoltre, è preferibile non includere sezioni eccessivamente ampie, in modo tale da non dover ricorrere al *chunking a dimensione fissata* (descritto alla sezione 4.4.2) in quanto, se una sezione viene suddivisa in più *chunk*, il sistema di *information retrieval* potrebbe non essere in grado di recuperarli tutti qualora fosse necessario, portando così alla generazione di una risposta potenzialmente imprecisa o non completa.

4.4 Chunking

Dal momento che i documenti che vengono caricati nell'applicativo contengono un'elevata quantità di testo, prima di effettuare la ricerca delle informazioni vera e propria, è necessario svolgere l'operazione di *chunking* del testo.

Il *chunking* è il processo di suddivisione del testo in frammenti più piccoli, o *chunk*,

in modo da poterli gestire più facilmente da un punto di vista computazionale³⁴.

Di seguito viene descritta la strategia di *chunking* adottata.

4.4.1 Section-level chunking

La prima fase della suddivisione del testo consiste nell'identificare tutte le sezioni che compongono il documento in questione per poi creare un *chunk* per ogni sezione individuata. Questo approccio permette di ottenere dei blocchi che si concentrano esclusivamente su un unico argomento, preservando così il contenuto semantico.

Tuttavia, non tutte le sezioni del documento hanno la stessa dimensione, e ciò può rappresentare un problema perché alcune sezioni possono avere una dimensione troppo elevata, ottenendo in questo modo un *chunking* poco efficace.

4.4.2 Chunking a dimensione fissata con overlapping

Per risolvere il problema della suddivisione per sezioni, è stata adottata un'altra strategia di *chunking* per le sole sezioni troppo lunghe. Essa consiste nel suddividerle in segmenti a dimensione fissata, che in questo progetto è di 300 parole, cioè circa 2000 caratteri, prendendo in considerazione anche gli spazi e i segni di punteggiatura. Se si eseguisse una semplice suddivisione di questo tipo ci sarebbe il rischio di perdere troppo contenuto semantico, in quanto i *chunk* che compongono una stessa sezione non prendono in considerazione il contesto degli altri *chunk* che la compongono. Per mitigare questa problematica è stata utilizzata la strategia dell'*overlapping*: invece di iniziare un nuovo *chunk* alla fine di quello precedente, esso viene costruito in modo tale che il suo contenuto sia parzialmente sovrapposto al contenuto di quello precedente. In questo modo è possibile catturare in modo migliore il contesto per ciascun *chunk*, stabilendo una sorta di collegamento tra di essi, al prezzo di maggiori requisiti di archiviazione.

³*Breaking up is hard to do: Chunking in RAG applications.* URL: <https://stackoverflow.blog/2024/06/06/breaking-up-is-hard-to-do-chunking-in-rag-applications>.

⁴*Text Splitters: Smart Text Division with Llamaindex.* URL: <https://gustavo-espindola.medium.com/%EF%B8%8F-text-splitters-smart-text-division-with-llamaindex-e4bf8d805ad0>.

4.5 Ricerca lessicale

La ricerca lessicale classifica i documenti in base quanti termini della *query* dell'utente compaiono in ciascun documento, ma senza considerare in alcun modo il significato della richiesta. Questo tipo di ricerca viene effettuata con dei *match* esatti, ed eventuali sinonimi non vengono considerati. Tuttavia, mediante lo *stemming* dei termini (che verrà descritto successivamente) è possibile intercettare anche piccole variazioni delle parole.

4.5.1 Best Match 25

*Best Match 25*⁵ (BM25), conosciuto anche con il nome di *Okapi BM25*, è un algoritmo di *ranking* utilizzato principalmente dai motori di ricerca per stimare la rilevanza di un insieme di documenti in relazione ad una data *query*.

Il BM25 si basa sul modello *bag-of-words*, ovvero un modello testuale che usa una rappresentazione del testo come collezione non ordinata (*bag*) di parole. Esso, dunque, ignora l'ordine delle parole, e quindi anche la grammatica, ma prende in considerazione la molteplicità dei termini.

Data una query Q contenente le *keywords* q_1, \dots, q_n , lo score BM25 del documento D è il seguente:

$$score_{BM25}(D, Q) = \sum_{i=1}^n IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

dove

- $f(q_i, D)$ è il numero di occorrenze della *keyword* q_i nel documento D ;
- $|D|$ è la lunghezza del documento D in parole;
- $avgdl$ è la lunghezza media dei documenti;

⁵Stephen Robertson e Hugo Zaragoza. «The Probabilistic Relevance Framework: BM25 and Beyond». In: *Foundations and Trends® in Information Retrieval* 3.4 (2009), pp. 333–389. ISSN: 1554-0669. DOI: [10.1561/1500000019](https://doi.org/10.1561/1500000019). URL: <http://dx.doi.org/10.1561/1500000019>.

- k_1 e b sono parametri liberi, che in assenza di particolari ottimizzazioni, sono scelti come $k_1 \in [1.2, 2.0]$ (nel contesto di questo progetto di stage è stato scelto $k_1 = 1.5$) e $b = 0.75$;
- $IDF(q_i)$ è il peso *inverse document frequency* del termine q_i , ovvero una misura dell'importanza della parola q_i in un insieme di documenti. Una parola che compare in molti documenti avrà un basso IDF, mentre una parola che compare in un solo documento avrà un IDF più alto, in quanto viene considerata più specifica.

L'IDF del termine q_i viene calcolato nel modo seguente:

$$IDF(q_i) = \ln \left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1 \right)$$

dove N è il numero totale dei documenti e $n(q_i)$ è il numero di documenti che contengono il termine q_i .

4.5.2 Stemming dei termini e rimozione delle stop words

Prima di eseguire il calcolo degli IDF e dei punteggi BM25, si effettua lo *stemming* dei termini che compongono la *query* e i documenti, ovvero si rimuove la *desinenza*_G dei termini, ottenendo quindi il loro *tema*_G. Ciò consente di mappare nello stesso tema le parole che differiscono soltanto nella desinenza, considerandole quindi uguali. In questo modo si ottiene una ricerca più efficace, dato che non è necessario inserire nella *query* di ricerca le parole chiave esattamente come compaiono nei documenti. Oltre allo *stemming*, vengono anche rimosse tutte le *stop words*, ovvero tutte le parole che sono ritenute prive di significato (degli esempi di *stop words* italiane possono essere “il”, “di”, “al”, ecc).

4.6 Ricerca semantica

La ricerca semantica⁶⁷ si distingue dalla ricerca lessicale, in quanto essa non cerca dei *match* esatti con le parole che compongono la *query* dell'utente. Questo tipo di ricerca interpreta l'intento dell'utente e il significato contestuale dei termini che compaiono nella *query*, al fine di generare risultati più rilevanti. Con la ricerca semantica, quindi, viene considerato anche il contesto della ricerca, sinonimi, variazioni delle parole e interrogazioni in linguaggio naturale. I maggiori motori di ricerca, come Google e Bing, fanno uso di algoritmi di ricerca semantica per migliorare i risultati delle ricerche fatte dagli utenti.

4.6.1 Sentence embedding

Per intercettare il significato semantico dei documenti e della richiesta dell'utente, viene utilizzata la tecnica del *sentence embedding*⁸. In *Natural Language Processing (NLP)_G*, un *sentence embedding* è una rappresentazione numerica di una frase nella forma di un *vettore_G* di numeri reali, chiamato vettore di *embedding*, che codifica le informazioni semantiche di tale frase. Il *sentence embedding* è un'estensione del *word embedding*, che consiste nel convertire singole parole in vettori. I vettori di *embedding* che sono vicini tra loro nello spazio vettoriale mappano frasi (o parole) che hanno un significato simile.

4.6.1.1 BERT e Sentence-BERT

La conversione delle frasi in vettori di *embedding* viene effettuata tramite un LLM basato su *Sentence-BERT* (o SBERT, acronimo di *Sentence Bidirectional Encoder Representations from Transformers*) per la lingua italiana, chiamato

⁶ *What is Semantic Search? | A Comprehensive Semantic Search Guide*. URL: <https://www.elastic.co/what-is/semantic-search>.

⁷ Hannah Bast, Björn Buchhold e Elmar Haussmann. «Semantic Search on Text and Knowledge Bases». In: *Foundations and Trends® in Information Retrieval* 10.2-3 (2016), pp. 119–271. ISSN: 1554-0669. DOI: [10.1561/15000000032](https://doi.org/10.1561/15000000032). URL: <http://dx.doi.org/10.1561/15000000032>.

⁸ *Sentence Embedding Methods — A Survey*. URL: <https://medium.com/@busra.oguzoglu/sentence-embedding-methods-a-survey-7c62857f7b43>.

[nickprock/sentence-bert-base-italian-xxl-uncased](#)

nel portale HuggingFace. SBERT è una versione modificata di BERT, utilizzata per ottenere delle prestazioni migliori di quest'ultimo nella conversione di frasi complesse in vettori.

BERT

BERT⁹¹⁰ è un modello di linguaggio preaddestrato su grandi quantità di testo, che per produrre il vettore di *embedding* di una frase utilizza un approccio che prevede la tokenizzazione della frase, con l'aggiunta di *token* speciali, come [CLS] (che sta per *classify*), da anteporre ad ogni frase fornita in input al modello, e [SEP] (che sta per *separate*), da porre alla fine di esse. Ogni strato *transformer* di BERT calcola una rappresentazione per ogni *token* basata sul contesto bidirezionale della frase, ovvero considerando sia il contesto sinistro che destro del *token* in questione. Questo modello utilizza anche meccanismi di auto-attenzione, cioè dei meccanismi che calcolano il peso di attenzione di ciascun *token*, in modo da bilanciare l'importanza di ogni parola nella frase rispetto a tutte le altre. Per recuperare il vettore di *embedding* della frase iniziale, è possibile utilizzare quello corrispondente al *token* [CLS] (cioè il primo) della sequenza finale, oppure calcolare la media dei vettori di *embedding* di tutti i *token* che compongono la frase. I vettori prodotti da BERT sono composti da 768 dimensioni.

SBERT

I test hanno dimostrato che il *sentence embedding* effettuato da BERT con i *token* [CLS] non raggiunge un'efficacia sufficiente: spesso, infatti, gli *embeddings* prodotti da questo modello sono di qualità peggiore rispetto alla media degli *embeddings* delle parole che compongono la frase senza considerare il contesto.

Come anticipato in precedenza, per ottenere dei risultati di qualità migliore, viene

⁹*google-bert/bert-base-uncased*. URL: <https://huggingface.co/google-bert/bert-base-uncased>.

¹⁰Jacob Devlin et al. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». In: *CoRR* abs/1810.04805 (2018). arXiv: [1810.04805](https://arxiv.org/abs/1810.04805). URL: <http://arxiv.org/abs/1810.04805>.

utilizzato SBERT¹¹, ovvero una versione modificata di BERT. SBERT ha ottenuto prestazioni superiori nel *sentence embedding* facendo *fine-tuning* degli *embeddings* dei *token* [CLS] di BERT utilizzando un'architettura di *rete neurale siamese* (o gemella) sul *dataset* *Stanford Natural Language Inference (SNLI)_G*. Come nel caso di BERT, anche i vettori prodotti da SBERT sono formati da 768 dimensioni.

4.6.2 Cosine similarity

La *cosine similarity*¹² è una misura di similarità tra due vettori non nulli di numeri reali. Essa è il *coseno_G* dell'angolo compreso tra i due vettori, quindi è indipendente dalla lunghezza di questi ultimi, ma dipende solo dai loro angoli. La *cosine similarity* è sempre compresa nell'intervallo [-1, 1].

Dati due vettori n -dimensionali non nulli A e B , la *cosine similarity* tra A e B può essere ottenuta invertendo la formula del *prodotto scalare_G*, cioè

$$A \cdot B = \|A\| \|B\| \cos(\theta)$$

dove θ è la misura dell'angolo tra A e B .

La *cosine similarity* tra A e B , quindi, è calcolata nel modo seguente:

$$S_C(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \cdot \sqrt{\sum_{i=1}^n B_i^2}}$$

dove A_i e B_i sono le i -esime componenti dei vettori A e B rispettivamente.

Il risultato di similarità calcolato in questo modo può variare da -1, indicando che i due vettori sono esattamente opposti, fino a 1, indicando che i due vettori sono esattamente gli stessi, con 0 che indica che i due vettori sono ortogonali. I valori intermedi rappresentano una similarità o diversità intermedia.

Nel contesto di questo progetto, la *cosine similarity* viene utilizzata per comparare

¹¹Nils Reimers e Iryna Gurevych. «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks». In: *CoRR* abs/1908.10084 (2019). arXiv: [1908.10084](https://arxiv.org/abs/1908.10084). URL: <http://arxiv.org/abs/1908.10084>.

¹²COSINE DISTANCE, COSINE SIMILARITY, ANGULAR COSINE DISTANCE, ANGULAR COSINE SIMILARITY. URL: <https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/cosdist.htm>.

il vettore di *embedding* relativo alla richiesta dell'utente con quelli corrispondenti ai documenti su cui effettuare la ricerca, ottenendo così una lista di risultati ordinata (in modo decrescente) secondo i punteggi ottenuti da ciascun documento.

4.7 Differenze tra i due tipi di ricerca

In questa sezione vengono presentati degli esempi pratici che evidenziano in modo chiaro le differenze tra i due algoritmi di ricerca precedentemente presentati. A questo proposito, il *dataset* di esempio utilizzato è il seguente:

Codice	Testo
1	Il Siamese è un gatto di origine asiatica, del Siam, dal corpo elegante e longilineo e la particolare testa triangolare.
2	Il Maine Coon è una razza naturale con un manto di lunghezza intermedia. Il nome di questo gatto significa "procione del Maine".
3	Il Canadian Sphynx è una razza di gatto con la caratteristica di non presentare alcun pelo sul suo mantello.
4	Il Beagle è una razza di cani da caccia di taglia piccola, oggi abbastanza diffusi come cani da compagnia.
5	Il Chihuahua è una razza canina di piccole dimensioni.
6	I cactus sono compresi nella vasta tipologia delle piante grasse.
7	Il salice piangente è un albero della famiglia delle Salicacee. È uno dei salici maggiormente utilizzati come pianta ornamentale.
Continua nella prossima pagina...	

Tabella 4.1 – Continuazione della tabella

Codice	Testo
8	Pino è il nome comune di un genere di alberi e arbusti sempreverdi, appartenente alla famiglia Pinaceae.

Tabella 4.1: Dataset di esempio

Per semplicità, tutti gli esempi che seguono limiteranno i risultati delle ricerche ai primi tre e gli *score* saranno troncati a tre cifre decimali.

4.7.1 Ricerca per keywords presenti nel dataset

Questo esempio analizza il caso in cui l'utente effettua una ricerca con delle *keywords* che sono direttamente presenti nel *dataset* di riferimento. Con le *keywords* “pianta ornamentale”, la ricerca lessicale recupera i seguenti risultati:

Score	Codice	Testo
3.053	7	Il salice piangente è un albero della...
1.499	6	I cactus sono compresi nella vasta...

Tabella 4.2: Risultati della ricerca lessicale per la richiesta “pianta ornamentale”

Mentre con la ricerca semantica, per le stesse *keywords*, si ottiene:

Score	Codice	Testo
0.596	8	Pino è il nome comune di un genere di...
0.565	6	I cactus sono compresi nella vasta...
0.527	7	Il salice piangente è un albero della...

Tabella 4.3: Risultati della ricerca semantica per la richiesta “pianta ornamentale”

È evidente che in questo caso la ricerca lessicale recupera i risultati migliori: infatti essa pone correttamente il documento richiesto in prima posizione, con un netto distacco di punteggio rispetto al secondo, mentre la ricerca semantica lo classifica

addirittura come terzo.

Questo perché gli algoritmi di ricerca lessicale lavorano in modo più efficace quando l'utente fornisce una *query* composta da parole chiave che sono presenti nel *dataset* di riferimento così come sono state scritte dall'utente. Inoltre, con sole due *keywords*, il modello che si occupa di convertire le frasi in vettori di *embedding* non è in grado di compiere una conversione efficace, in quanto manca un vero e proprio contenuto semantico nella richiesta.

4.7.2 Ricerca per keywords non presenti nel dataset

Questo esempio analizza il caso in cui l'utente effettua una ricerca che contiene delle *keywords* che non sono presenti nel *dataset* di riferimento. Con la richiesta “felino dal pelo medio”, la ricerca lessicale recupera i seguenti risultati:

Score	Codice	Testo
1.780	3	Il Canadian Sphynx è una razza di gatto...

Tabella 4.4: Risultati della ricerca lessicale per la richiesta “felino dal pelo medio”

Mentre con la ricerca semantica, per le stesse *keywords*, si ottiene:

Score	Codice	Testo
0.689	2	Il Maine Coon è una razza naturale con un...
0.644	3	Il Canadian Sphynx è una razza di gatto...
0.618	5	Il chihuahua è una razza canina di piccole...

Tabella 4.5: Risultati della ricerca semantica per la richiesta “felino dal pelo medio”

Rispetto al caso precedente, ora è la ricerca semantica ad essere quella più efficace: con questo algoritmo, infatti, il documento più pertinente viene posto in prima posizione, mentre la ricerca lessicale recupera un solo documento, e non è quello richiesto.

Ciò accade perché delle parole nella richiesta soltanto una compare nel *dataset*, ovvero nel documento recuperato dalla ricerca lessicale, ma non è ciò che l'utente si

aspetterebbe di ricevere come risposta dopo l'invio di quella richiesta. La ricerca semantica invece recupera correttamente il documento desiderato, dal momento che la richiesta è sufficientemente articolata da possedere un contenuto semantico, permettendo così al modello SBERT di effettuare una conversione efficace.

4.7.3 Considerazioni

Con i due esempi presentati, sebbene in entrambi i casi i risultati nelle prime posizioni non siano del tutto estranei alla richiesta, si evince che non esiste un tipo di ricerca che funzioni meglio in tutte le occasioni, ma la loro efficacia dipende fortemente dalla struttura della *query* dell'utente. Dato che non è possibile predire in che modo egli andrà a comporre la richiesta per svolgere la ricerca, è necessario fare utilizzo di un terzo algoritmo, il cui compito è quello di unire gli aspetti positivi dei due tipi di ricerca, prendendo in considerazione i criteri su cui ciascuna di esse si basa.

4.8 Ricerca ibrida

La ricerca ibrida¹³ (in inglese *hybrid search*) è una tecnica che sfrutta più algoritmi di ricerca, che operano secondo diversi criteri, per migliorare la precisione e la rilevanza dei risultati ottenuti. Nel contesto di questo progetto, la ricerca ibrida viene effettuata per combinare i punti di forza della ricerca lessicale basata sulle *keywords* e quelli della ricerca semantica, ottenendo così, nel complesso, una ricerca più efficace.

4.8.1 Algoritmi di rank fusion

Per implementare la ricerca ibrida è necessario unificare i *ranking* ottenuti dagli algoritmi di ricerca sotto un'unica lista finale di risultati. Per fare ciò, si utilizza un algoritmo di *rank fusion*. Solitamente, l'utilizzo di un algoritmo di *rank fusion* per-

¹³ *What is Hybrid Search?* URL: <https://medium.com/@qdrant/what-is-hybrid-search-2a0c30d0f3d2>.

mette di ottenere un *set* di risultati molto più efficace rispetto a quello che produce un singolo algoritmo di ricerca.

Ci sono due tipi di algoritmi di *rank fusion*: quelli *rank-based*, che prendono in considerazione soltanto l'ordine dei documenti in ogni lista di risultati osservata, trascurando completamente i punteggi di pertinenza dei documenti, e quelli *score-based*, che a differenza dei precedenti, come suggerisce il nome, prendono in considerazione anche i punteggi di pertinenza.

4.8.1.1 Reciprocal Rank Fusion

*Reciprocal Rank Fusion*¹⁴¹⁵ (RRF) uno degli algoritmi di *rank fusion rank-based* più utilizzati. L'algoritmo RRF considera la posizione dei documenti nelle classifiche ottenute con gli altri algoritmi di ricerca ed assegna un punteggio maggiore ai documenti che compaiono nelle prime posizioni in più liste. Ciò produce quindi un *ranking* finale più affidabile e di maggiore qualità.

Sia L l'insieme di tutti i *ranking* prodotti dagli altri algoritmi di ricerca. Dato un documento d , il suo punteggio RRF è

$$score_{RRF}(d) = \sum_{l \in L} \frac{1}{k + r(d, l)}$$

dove k è una costante (tipicamente impostata a 60) e $r(d, l)$ è la posizione del documento d all'interno del *ranking* l .

Tuttavia, essendo il RRF un algoritmo *rank-based*, ha il difetto di non considerare i punteggi dei documenti ottenuti nelle liste originali, perdendo così un'informazione molto importante.

4.8.1.2 Relative Score Fusion

Per migliorare l'efficacia della ricerca ibrida si può utilizzare al posto del RRF un algoritmo *score-based*, avendo così a disposizione un'informazione aggiuntiva con cui

¹⁴Assegnazione dei punteggi nella ricerca ibrida (RRF). URL: <https://learn.microsoft.com/it-it/azure/search/hybrid-search-ranking>.

¹⁵Hybrid search with Re-ranking. URL: <https://medium.com/@sowmiyajaganathan/hybrid-search-with-re-ranking-ff120c8a426d>.

poter costruire un *ranking* finale più preciso, ovvero gli *score* ottenuti dai vari documenti con gli algoritmi di ricerca precedentemente utilizzati. Uno dei più semplici algoritmi *score-based* è il *Relative Score Fusion*¹⁶ (RSF): esso consiste semplicemente nel normalizzare i punteggi delle liste di risultati ottenute per poi unirle producendone una nuova, ordinata per punteggio (nel caso un documento sia presente in più di una lista, si seleziona quello che ha ottenuto il punteggio maggiore).

La normalizzazione è necessaria a causa della differenza di ordini di grandezza dei punteggi assegnati dagli algoritmi di ricerca: per esempio, il BM25 assegna punteggi nell'intervallo $[0, +\infty[$, mentre i punteggi della *cosine similarity* appartengono all'intervallo $[0, 1]$, rendendo così potenzialmente insensate le operazioni di confronto tra *score* ottenuti da algoritmi diversi. La normalizzazione, quindi, serve per portare i punteggi tutti sulla stessa scala (tipicamente tra 0 e 1), in modo da rendere applicabili i confronti tra di essi per produrre il *ranking* finale.

Una volta normalizzati i punteggi, dunque, è possibile unire agevolmente tutte le liste di risultati sotto un'unica lista, ordinata in modo decrescente secondo i punteggi normalizzati.

Dato un insieme R che contiene tutti i *ranking* ottenuti dagli algoritmi di ricerca, lo *score* RSF che questo algoritmo assegna al documento d è

$$score_{RSF}(d, R) = \max_{r \in R} \{n(d, r)\}$$

con n definita come

$$n(d, r) = \begin{cases} score(d, r) & \text{se } \forall x \in r \ score(x, r) \in [0, 1] \\ f(score(d, r)) & \text{altrimenti} \end{cases}$$

dove $score(d, r)$ è il punteggio del documento d all'interno del *ranking* r e f è una qualsiasi *funzione monotona crescente* $f : [0, +\infty[\rightarrow [0, 1]$, utile appunto per normalizzare gli *score* nell'intervallo $[0, 1]$ delle liste generate dagli algoritmi che possono produrre punteggi al di fuori di tale intervallo.

¹⁶*Distribution-Based Score Fusion (DBSF), a new approach to Vector Search Ranking*. URL: <https://medium.com/plain-simple-software/distribution-based-score-fusion-dbsf-a-new-approach-to-vector-search-ranking-f87c37488b18>.

L'algoritmo RSF non impone una specifica funzione di normalizzazione: nel progetto di stage in oggetto, inizialmente sono stati svolti dei *test* scegliendo f come

$$g(x) = \frac{\arctan(x)}{\pi/2}$$

oppure

$$h(x) = 1 - e^{-x}$$

per portare gli *score* del BM25 nell'intervallo $[0, 1]$, seguendo il suggerimento del tutor aziendale. Successivamente sono stati effettuati dei test utilizzando il *min-max scaling*_G, che viene usato anche in contesti reali in cui viene applicato questo algoritmo, come ad esempio nel caso di Weaviate¹⁷ (un *database vettoriale*_G *open source*) e LlamaIndex¹⁸ (un *framework open-source* per applicazioni basate sui LLM). Nel complesso, il *min-max scaling* è risultato essere la funzione di normalizzazione che permette di generare la miglior lista di punteggi finale.

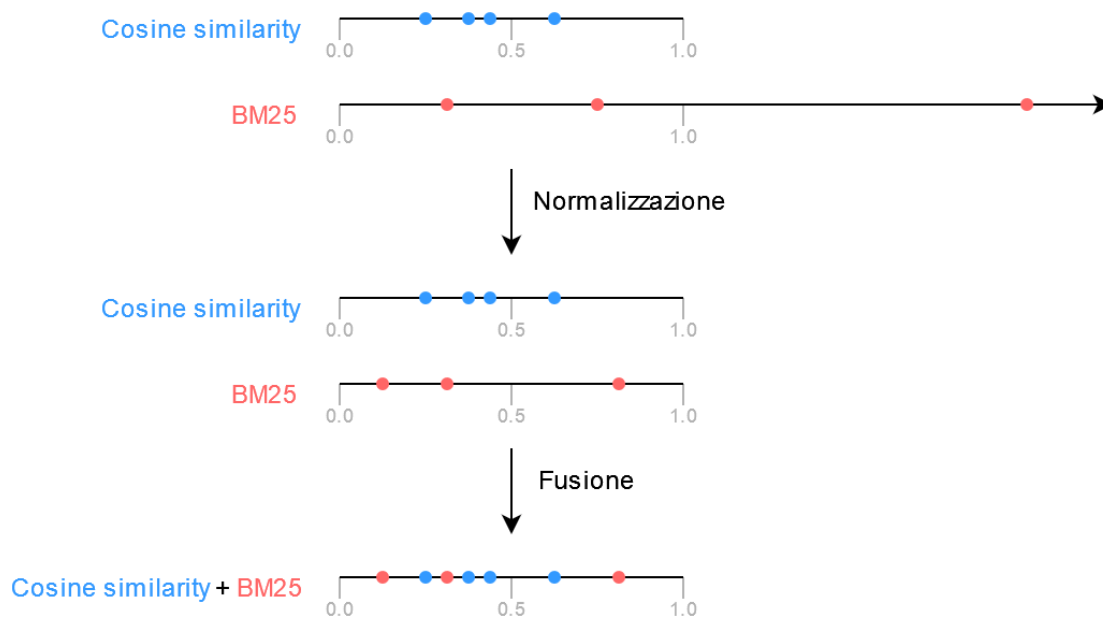


Figura 4.3: Esempio di applicazione del RSF con normalizzazione

¹⁷ Weaviate: The AI-native database developers love. URL: <https://weaviate.io/>.

¹⁸ LlamaIndex, Data Framework for LLM Applications. URL: <https://www.llamaindex.ai/>.

Ad ogni modo, nonostante tutti i punteggi siano sulla stessa scala, si presenta un ulteriore problema: un approccio di questo tipo funziona bene soltanto se le *distribuzioni*_G dei punteggi date dai diversi algoritmi di ricerca sono molto simili. Infatti, gli *score* assegnati da uno stesso algoritmo tendono ad essere concentrati vicino al loro valore medio (che non è correlato alla rilevanza della ricerca) con una *varianza*_G troppo bassa per raggiungere gli estremi (0 e 1).

Per esempio, considerando il *dataset* che raccoglie tutti gli articoli del Codice civile italiano, con la *query* “lavorare oltre il proprio orario di lavoro normale” la ricerca con BM25 e quella con *cosine similarity* recuperano, rispettivamente, i seguenti risultati¹⁹ (limitati ai primi dieci per ciascun algoritmo ed approssimando gli *score* a tre cifre decimali):

Score	Score normalizzato	Codice
17.210	1.000	art2108
11.618	0.480	art2079
11.483	0.467	art2167
11.132	0.435	art2147
10.196	0.348	art2127
9.847	0.315	art2225
9.358	0.270	art1699
8.215	0.164	art2578
7.843	0.129	art2094
7.393	0.087	art2

Tabella 4.6: Risultati della ricerca con BM25 per la richiesta “lavorare oltre il proprio orario di lavoro normale”

¹⁹I risultati ottenuti dal BM25 sono stati normalizzati tra 0 e 1 tramite *min-max scaling*.

Score	Codice
0.597	art2108
0.578	art2107
0.534	art2130
0.507	art2124
0.504	art2243
0.500	art2094
0.489	art2100
0.473	art2131
0.471	art2246
0.470	art2133

Tabella 4.7: Risultati della ricerca con cosine similarity per la richiesta “lavorare oltre il proprio orario di lavoro normale”

Si può notare che i punteggi del BM25 sono prevalentemente concentrati nell’intervallo $[0.27, 0.48]$, con una media complessiva di 0.206, mentre i punteggi della *cosine similarity* sono tutti compresi nell’intervallo $[0.47, 0.6]$ con una media di 0.476. Utilizzando quindi il RSF per la fusione delle due liste, i risultati trovati tramite la ricerca semantica sarebbero “privilegiati” nel *ranking* finale, in quanto più vicini a 1.

Siccome è molto raro che diversi algoritmi di ricerca producano risultati con punteggi che hanno una distribuzione simile, soprattutto se questi lavorano considerando criteri differenti, l’algoritmo *Relative Score Fusion* non si rivela essere efficace.

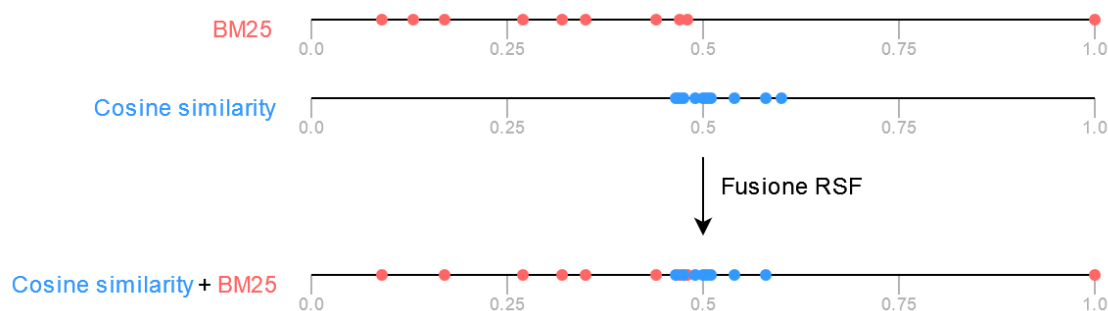


Figura 4.4: Applicazione del RSF sui due ranking precedenti

4.8.1.3 Distribution-Based Score Fusion

L'algoritmo *Distribution-Based Score Fusion*²⁰ (DBSF) è una variante del *Relative Score Fusion* ed è stato progettato per risolvere i problemi e le criticità derivanti da quest'ultimo: per ogni lista di risultati esso esegue una *standardizzazione* data dalla formula del *min-max scaling*, ma al posto del massimo e del minimo viene usata la media delle distribuzioni degli *score* aumentata e diminuita del triplo della *deviazione standard* σ_r , rispettivamente. Anche in questo algoritmo, nel caso in cui uno stesso documento compaia in più di una lista, viene tenuto quello con punteggio standardizzato più alto. Nello specifico, dato un insieme R contenente tutti i *ranking* prodotti dagli algoritmi di ricerca, il punteggio che questo algoritmo assegna al documento d è il seguente:

$$score_{DBSF}(d, R) = \max_{r \in R} \{std_{DBSF}(d, r)\}$$

con

$$std_{DBSF}(d, r) = \begin{cases} \frac{score(d, r) - (\mu_r - 3\sigma_r)}{(\mu_r + 3\sigma_r) - (\mu_r - 3\sigma_r)} & \text{se } \sigma_r > 0 \\ -K & \text{altrimenti} \end{cases}$$

dove

- $score(d, r)$ è il punteggio del documento d nel *ranking* r ;
- μ_r è la media dei punteggi del *ranking* r ;
- σ_r è la deviazione standard dei punteggi del *ranking* r ;
- $-K$ è un numero negativo di modulo arbitrariamente grande. Impostato con modulo molto grande, questo parametro è utile per escludere dalla fusione tutte le liste di punteggi che hanno deviazione standard pari a 0, cioè composte da valori tutti identici tra loro. In una situazione reale, un'eventualità del genere è praticamente impossibile: pertanto, se si riscontra una situazione di questo

²⁰*Distribution-Based Score Fusion (DBSF), a new approach to Vector Search Ranking.*

tipo, è molto probabile che sia stato commesso un errore nell'implementazione dell'algoritmo che ha generato la lista in questione.

L'immagine sottostante illustra come, nell'esempio della *query* alla sezione precedente, i risultati alle tabelle 4.6 e 4.7 vengono fusi dall'algoritmo DBSF:

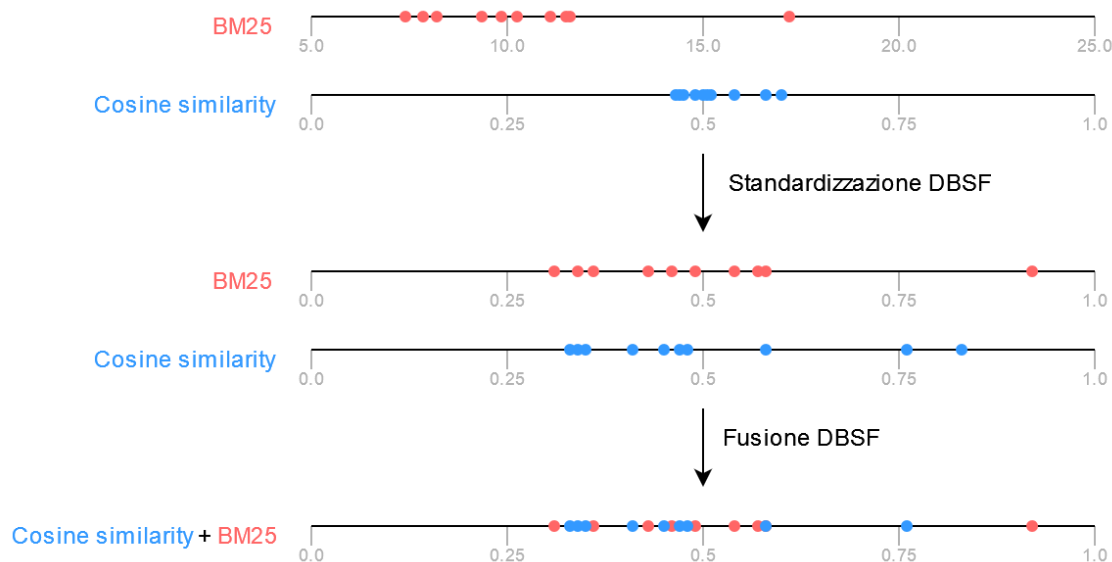


Figura 4.5: Applicazione del DBSF sui due ranking precedenti

4.8.1.3.1 Osservazioni

Nel caso del DBSF si parla più propriamente di *standardizzazione*, e non di normalizzazione (e quindi la funzione che utilizza non può essere un vero e proprio *min-max scaling*): infatti, dal momento che nella formula utilizzata vengono impiegati degli estremi che non sono costanti, ma che variano a seconda della distribuzione dei punteggi forniti all'algoritmo, la funzione ottenuta non è del tipo $f : [0, +\infty[\rightarrow [0, 1]$, bensì $g : [0, +\infty[\rightarrow] - \infty, +\infty[$.

È possibile infatti costruire delle situazioni di esempio in cui la funzione del DBSF restituisce valori al di fuori dell'intervallo $[0, 1]$. Si consideri una distribuzione a con media μ_a e varianza σ_a^2 , con $\sigma_a > 0$ e si assuma sia l'unico *ranking* fornito in input all'algoritmo DBSF. Poiché 0 è un valore lecito come *score* per un documento d_0 ,

allora lo *score* DBSF di quel documento sarà

$$score_{DBSF}(d_0, \{a\}) = \frac{0 - (\mu_a - 3\sigma_a)}{(\mu_a + 3\sigma_a) - (\mu_a - 3\sigma_a)} = \frac{-\mu_a + 3\sigma_a}{6\sigma_a}$$

da cui si può notare che per distribuzioni con media maggiore del triplo della deviazione standard sia possibile ottenere valori negativi. In particolare, è possibile calcolare il limite per μ_a tendente a $+\infty$ dell'espressione di prima:

$$\lim_{\mu_a \rightarrow +\infty} \frac{-\mu_a + 3\sigma_a}{6\sigma_a} = -\infty$$

essendo $\sigma_a > 0$ una quantità finita. In generale, dato un valore di *score* $x \in a$, la funzione di standardizzazione del DBSF restituisce un valore negativo anche in presenza di una deviazione standard positiva se $\mu_a > x + 3\sigma_a$.

È possibile inoltre costruire un esempio opposto, in cui il risultato restituito dalla funzione sia positivo e arbitrariamente grande. Si consideri una distribuzione b con media μ_b e varianza σ_b^2 , con $\sigma_b > 0$ e si assuma sia l'unica lista passata come input all'algoritmo. Poiché il dominio della funzione è $[0, +\infty[$, è possibile calcolare il limite di tale funzione per lo *score* $x \in b$ che tende a $+\infty$:

$$\lim_{x \rightarrow +\infty} score_{DBSF}(x, \{b\}) = \lim_{x \rightarrow +\infty} \frac{x - (\mu_b - 3\sigma_b)}{(\mu_b + 3\sigma_b) - (\mu_b - 3\sigma_b)} = +\infty$$

essendo μ_b e σ_b quantità finite e $\sigma_b > 0$. In generale, la standardizzazione DBSF per un valore di *score* $x \in b$ risulta maggiore di 1 se $x - \mu_b + 3\sigma_b > 6\sigma_b$, cioè, semplificando, $x - \mu_b > 3\sigma_b$.

Un esempio pratico di distribuzione di *score* che porta ad un risultato negativo può essere il seguente:

$$r = \{0, 40, 41, 42, 43, 44, 46, 47, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58\}$$

con media $\mu_r = 46.55$ e deviazione standard $\sigma_r = 11.939$. Calcolando la standardizzazione DBSF per ciascun punteggio si ottengono tutti valori appartenenti all'intervallo $[0, 1]$ al di fuori del valore relativo allo *score* pari a 0, che risulta essere -0.15 (infatti, si ha che $\mu_r = 46.55 > 0 + 3\sigma_r = 35.817$, a riprova quanto detto in precedenza).

Un esempio di distribuzione che porta ad un risultato maggiore di 1, invece, può essere il seguente:

$$s = \{40, 41, 42, 43, 44, 46, 47, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 1000\}$$

con media $\mu_s = 96.55$ e deviazione standard $\sigma_s = 207.334$. Anche in questo caso i risultati che la funzione di standardizzazione del DBSF restituisce per questa distribuzione sono tutti nell'intervallo $[0, 1]$, tranne il valore corrispondente allo *score* di 1000, che vale 1.226 (infatti, si ha che $1000 - \mu_s = 903.45 > 3\sigma_s = 622.002$).

È importante sottolineare che queste distribuzioni sono state costruite “a mano” per rappresentare dei casi limite e, nonostante siano teoricamente possibili, non rispecchiano in alcun modo delle situazioni tipiche: nella pratica sono casi estremamente rari e si verificano generalmente in presenza di *outlier*_G (come 0 in *r* e 1000 in *s*), soprattutto quando le distribuzioni sono fortemente concentrate attorno alla media, come in quelle riportate negli esempi di cui sopra. Infatti, in una *distribuzione normale*_G solo lo 0.2% dei valori di *score* può produrre risultati negativi o maggiori di 1. Queste peculiarità, dunque, non pregiudicano il corretto funzionamento del sistema e viene comunque garantita la monotonia. Inoltre, dato che nel progetto le liste di *score* date in input al DBSF hanno lunghezza massima pari a venti elementi, anche in presenza di *outlier* eventuali valori al di fuori dell'intervallo $[0, 1]$ si discostano di poco dagli estremi di quest'ultimo, come è possibile notare anche nel caso degli esempi appena presentati (che utilizzano, appunto, delle distribuzioni composte da venti elementi).

In ogni caso, dai test effettuati emerge che, tra i tre algoritmi di *rank fusion* analizzati, il DBSF produce generalmente la lista con i risultati più pertinenti.

4.9 Knowledge graph

Per aumentare ulteriormente la qualità del contesto che verrà fornito al *chatbot* è possibile recuperare tutte le sezioni del documento caricato che sono “collegate” a quelle ottenute dal processo di *information retrieval*. Per fare ciò è possibile sfruttare

il *knowledge graph*²¹ del documento in questione.

Un *knowledge graph*, conosciuto anche come *semantic graph* o *semantic network*, è un *grafo*_G i cui vertici rappresentano entità del mondo reale (come ad esempio oggetti, eventi o concetti) e gli archi rappresentano le relazioni che sussistono tra di esse.

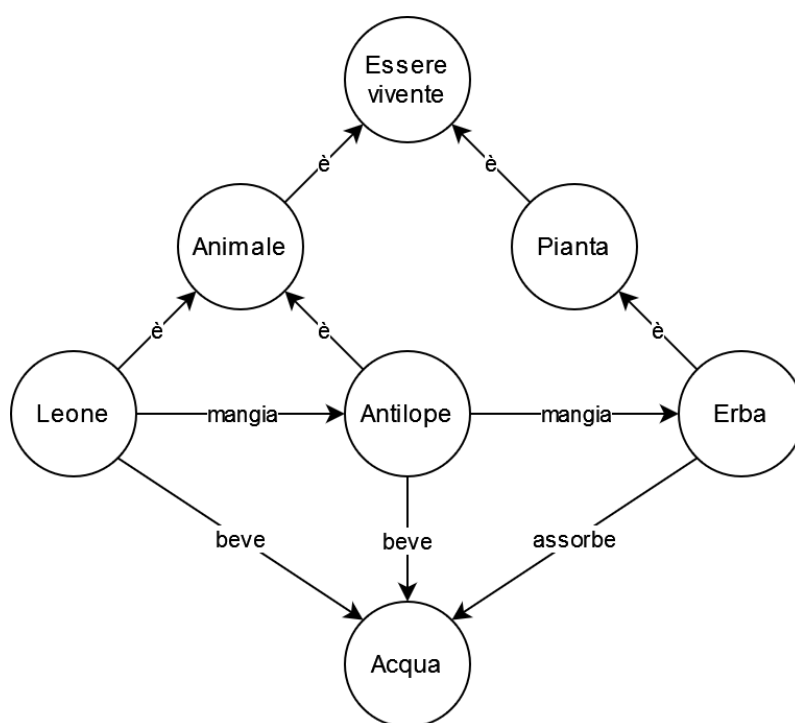


Figura 4.6: Esempio di knowledge graph

Solitamente i *knowledge graph* sono utilizzati dai maggiori motori di ricerca, come Google o Bing, e dai maggiori *social network*, come LinkedIn e Facebook. Tuttavia, con i recenti sviluppi della *data science* e del *machine learning*, gli ambiti d'uso dei *knowledge graph* sono aumentati.

Nel progetto di stage in oggetto, il *knowledge graph* è relativo al documento che è stato caricato nel *server*: i nodi rappresentano i vari *chunk* che compongono il suo contenuto e gli archi rappresentano le connessioni semantiche che intercorrono tra di essi. Tale grafo è non orientato.

²¹ *What Is a Knowledge Graph?* URL: <https://www.ibm.com/topics/knowledge-graph>.

4.9.1 Generazione del grafo e query di attraversamento

Il *knowledge graph* della documentazione viene costruito al momento del caricamento di un nuovo *file* nel *server* sulla base degli *embeddings* dei *chunk* calcolati in precedenza. Per ciascuna coppia di vettori di embeddings $\{x, y\}$ relativi a due *chunk* della documentazione caricata viene quindi calcolata la *cosine similarity* $S_C(x, y)$, e se essa risulta in un valore maggiore o uguale a 0.2 viene costruito un arco (non orientato) che collega x con y .

Per aumentare il contesto del *chatbot* viene eseguita una *query*²² che ha lo scopo di recuperare tutti i nodi attraversati percorrendo i tre migliori percorsi che collegano tutti i nodi relativi ai *chunk* ottenuti dal sistema di *information retrieval*.

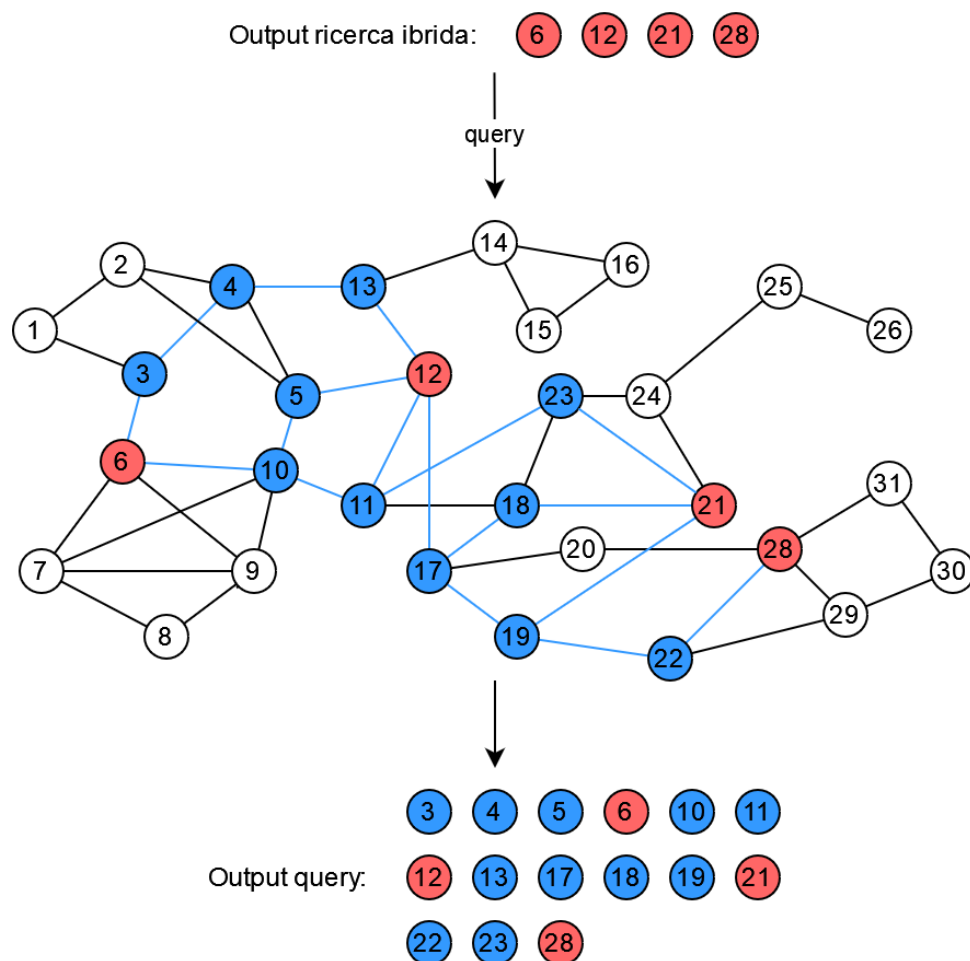


Figura 4.7: Rappresentazione della query di attraversamento di un knowledge graph

²²Advanced RAG with graph path traversal. URL: https://github.com/neuml/txtai/blob/master/examples/58_Advanced_RAG_with_graph_path_traversal.ipynb.

Un valore di *threshold* così basso (0.2 su 1.0) garantisce il rilevamento di tutte le connessioni semantiche tra i vari *chunk*, che in combinazione con la *query* utilizzata rende possibile il recupero di alcune parti della documentazione che possano completare il contesto estratto in precedenza. In particolare, ciò è utile per prelevare tutti i frammenti che compongono una stessa sezione di documentazione suddivisa tramite *chunking* a dimensione fissata, data l'eventualità che con la sola ricerca ibrida ne vengano tralasciati alcuni.

Dunque, con l'utilizzo del *knowledge graph* il *chatbot* produrrà una risposta finale più completa e precisa.

Capitolo 5

Generazione della risposta

5.1 Introduzione

Lo scopo finale di questo progetto di stage è fornire una risposta a tutte le possibili richieste inviate dall'utente. A questo proposito viene fatto utilizzo di un *Large Language Model* specializzato nel produrre risposte, ovvero un modello di *Question Answering*.

Vengono inoltre gestiti (in modo diverso) i casi eccezionali, per esempio la risposta ad una domanda non pertinente con il *file* di riferimento.

5.2 Question Answering

Un modello di *Question Answering*¹ (QA) è un LLM che è in grado di recuperare la risposta ad una domanda da un dato contesto. Ci sono generalmente due possibilità per svolgere questo tipo di *task*: *Extractive QA* e *Generative QA*.

5.2.1 Extractive Question Answering

Le prime prove di generazione delle risposte nell'applicativo sviluppato come progetto di stage sono state fatte con modelli che effettuano il QA in modo *estrat-*

¹ *What is Question Answering?* URL: <https://huggingface.co/tasks/question-answering>.

*tivo*². Ciò significa che il modello estrae in modo diretto la risposta alla domanda dal contesto fornito, senza compiere nessuna ulteriore elaborazione. Il contesto può essere un semplice documento testuale, una tabella o anche un *file* HTML.

Questi modelli presentano i seguenti vantaggi:

- recuperano risposte precise ed accurate, che sono facilmente verificabili nel contesto;
- sono più piccoli, più leggeri e più efficienti dal punto di vista computazionale rispetto ai modelli generativi;
- gestiscono bene anche l'estrazione di risposte da documenti molto grandi, come libri o articoli di ricerca.

Tuttavia, presentano anche i seguenti svantaggi:

- questi sistemi eccellono quando la risposta alla domanda fornita è presente in modo esplicito nel contesto. Essi faticano invece con domande che richiedono la sintesi di informazioni oppure con il recupero di dati che non sono esplicitamente indicati nel contesto;
- documenti che presentano un formato non standard potrebbero rappresentare un problema per questi modelli;
- l'accuratezza dei fatti di questi sistemi dipende fortemente dalla qualità del documento fornito come contesto, in quanto non sono in grado di generare risposte in modo autonomo.

5.2.2 Generative Question Answering

Nonostante gli svariati pregi di cui godono i modelli estrattivi, i loro difetti rappresentano una limitazione eccessiva per il tipo di domande che il sistema relativo al progetto di stage in questione deve essere in grado di rispondere. Per questo

²*Extractive Question Answering*. URL: <https://docs.cloud.deepset.ai/docs/extractive-question-answering>.

motivo è stato deciso di passare ad un modello *generativo*³.

I modelli generativi, a differenza di quelli estrattivi, possono generare liberamente del testo basato sul contesto come risposta alla domanda fornita.

I vantaggi nell'utilizzo di un modello di questo tipo sono i seguenti:

- il modello può estrarre e combinare informazioni provenienti da fonti diverse per produrre una risposta coerente;
- le risposte prodotte sono più naturali;
- i modelli sono addestrati su un vasto volume di dati, con varie lingue, permettendo così la generazione di risposte anche in lingua diversa rispetto alla domanda o al contesto;
- i modelli generativi hanno modeste capacità di ragionamento, e sono quindi in grado di analizzare i fatti e trarre conclusioni.

Tuttavia essi presentano anche alcune limitazioni e svantaggi:

- i modelli generativi possono soffrire di *allucinazioni*, ovvero la possibilità di generare una risposta non corretta con un alto grado di sicurezza su quanto scritto. Ciò può essere causato da alcuni *bias* nei dati utilizzati nell'addestramento o dall'incapacità del modello di differenziare le informazioni reali da quelle false;
- generalmente, i modelli generativi sono più lenti rispetto a quelli estrattivi;
- le risposte generate, a volte, possono includere contenuti inappropriati o di parte.

Nel progetto di stage in questione è possibile scegliere tra tre modelli generativi per la generazione della risposta finale. I modelli che sono stati scelti e testati sono:

- **Mixtral 8x7B**⁴: è un modello pre-addestrato sviluppato dall'azienda francese Mistral AI. È composto da un totale di 46.7 miliardi di parametri ma ne utilizza

³*Retrieval Augmented Generation (RAG) Question Answering*. URL: <https://docs.cloud.deepset.ai/docs/generative-question-answering>.

⁴*mistralai/Mixtral-8x7B-v0.1*. URL: <https://huggingface.co/mistralai/Mixtral-8x7B-v0.1>.

soltanto 12.9 miliardi per ogni *token*, dunque l’elaborazione degli input e la generazione del testo avvengono alla stessa velocità di un modello da 12.9 miliardi di parametri. Ciò è dovuto grazie alla sua architettura *mixture of experts*⁵: i modelli che adottano questa architettura sono composti da un insieme di reti neurali “esperte” nello svolgimento di un determinato compito e da un *gate network*, il cui compito è determinare a quale rete neurale inviare ciascun *token*. Mixtral 8x7B si è rivelato più performante dei celebri GPT 3.5 e Llama 2;

- **Meta Llama 3 8B**⁶: è un modello *open-source* pre-addestrato sviluppato da Meta. È il successore di Llama 2. Ci sono tre versioni di questo modello, che differiscono nel numero di parametri: la versione più piccola, anche utilizzata nel progetto, è composta da 8 miliardi di parametri, quella intermedia presenta 70 miliardi di parametri e quella più grande e più performante possiede 405 miliardi di parametri. Llama 3 presenta eccellenti performance nel dialogo con l’utente, e durante lo sviluppo di questo modello è stata attribuita particolare attenzione alla sicurezza delle risposte generate;
- **Gemma 2 9B**⁷: disponibile nella versioni da 9 miliardi di parametri e da 27 miliardi di parametri, Gemma 2 è un modello pre-addestrato sviluppato da Google. Questo modello è più performante del suo predecessore (Gemma), con significativi miglioramenti nella sicurezza. La versione da 27 miliardi di parametri offre prestazioni equivalenti, e in alcuni casi persino superiori, rispetto a molti altri modelli di dimensioni doppie.

⁵*Mixture of Experts Explained*. URL: <https://huggingface.co/blog/moe#what-is-a-mixture-of-experts-moe>.

⁶*meta-llama/Meta-Llama-3-8B*. URL: <https://huggingface.co/meta-llama/Meta-Llama-3-8B>.

⁷*google/gemma-2-9b*. URL: <https://huggingface.co/google/gemma-2-9b>.

5.3 Retrieval-Augmented Generation

La *Retrieval-Augmented Generation*⁸ (RAG) è il processo di miglioramento dell'output di un *Large Language Model* tramite l'integrazione di informazioni autorevoli all'interno del *prompt* che vadano oltre ai dati utilizzati durante il suo addestramento. Queste informazioni da allegare al *prompt* prendono il nome di *contesto*.

La RAG estende quindi le già avanzate capacità dei LLM a domini specifici o alla *knowledge base* interna di un'organizzazione, con il vantaggio di non dover riaddestrare il modello. Con l'utilizzo della RAG è inoltre possibile presentare la risposta all'utente facendo riferimento ai documenti originali consultati per la sua generazione: in questo modo, se l'utente lo desidera, ha la possibilità di visualizzare ulteriori approfondimenti o chiarimenti grazie alle fonti citate. Ciò aumenta la sicurezza e la fiducia nella risposta fornita dal modello di intelligenza artificiale.

Dal punto di vista dello sviluppatore, oltre ad essere un approccio conveniente (riaddestrare un modello di grandi dimensioni può essere un'operazione estremamente dispendiosa in termini di tempo), optando per questa via è possibile testare ed eventualmente migliorare l'applicativo in modo più efficiente. Ad esempio, si possono consultare le fonti che compongono il contesto nel *prompt* del LLM per verificare la correttezza delle risposte generate e apportare le dovute correzioni. Inoltre, gli sviluppatori possono anche limitare o vietare il recupero di eventuali informazioni sensibili, a diversi livelli di autorizzazione, per garantire che il LLM generi sempre risposte appropriate.

Nel contesto di questo progetto di stage, il processo di RAG viene alimentato dal sistema di *information retrieval* descritto al capitolo precedente: il contesto a cui il LLM deve fare riferimento al momento della generazione della risposta è quindi costituito dai risultati ottenuti dalla ricerca ibrida.

⁸ Cos'è la RAG (Retrieval-Augmented Generation)? URL: <https://aws.amazon.com/it/what-is/retrieval-augmented-generation/>.

5.4 Gestione dei casi eccezionali

Oltre alle normali richieste pertinenti con la documentazione caricata all'interno dell'applicativo, il *chatbot* sviluppato deve far fronte ad alcune possibili eccezioni. Queste si configurano principalmente in tre casi, ordinati in base all'urgenza della loro gestione:

1. le richieste che fanno riferimento ad attività illegali;
2. le richieste non pertinenti con il contesto fornito al LLM;
3. le richieste per cui non è possibile recuperare nessun *chunk* di documentazione rilevante come contesto da fornire al LLM.

Di seguito viene descritta la modalità di gestione di ciascuno di questi tre casi.

5.4.1 Moderazione dei contenuti

È di fondamentale importanza prendere le dovute precauzioni per fare in modo che il modello non risponda a domande che facciano riferimento ad azioni illegali o con l'obiettivo di danneggiare cose o persone. A tal fine, viene fatto utilizzo di un altro LLM specializzato nell'individuare e determinare l'entità di una richiesta potenzialmente dannosa.

Il modello scelto per svolgere questo compito è Llama Guard 3⁹: si tratta di una variante del modello di Meta Llama 3.1 da 8 miliardi di parametri, a cui è stato applicato un *fine-tuning* per specializzare il modello nella classificazione delle richieste in termini di sicurezza. Nello specifico, dato un *prompt* in input a Llama Guard 3, quest'ultimo analizza sia la domanda posta dall'utente (*prompt classification*) che la potenziale risposta (*response classification*): esso risponde con la stringa “safe” se la domanda e la risposta non presentano contenuti illeciti, altrimenti risponde con la stringa “unsafe” seguita una lista di codici che rappresentano tutte le violazioni che la domanda o la risposta contengono. Questo modello agisce in linea con la

⁹*meta-llama/Llama-Guard-3-8B*. URL: <https://huggingface.co/meta-llama/Llama-Guard-3-8B>.

tassonomia dei pericoli definita dalla MLCommons¹⁰¹¹.

Nel progetto di stage, questo LLM agisce da filtro per le richieste dell'utente: la richiesta viene inoltrata al sistema di *information retrieval* per la costruzione del contesto soltanto se essa è stata classificata come “safe” da Llama Guard 3. In caso contrario, la richiesta viene bloccata per impedire la generazione della risposta finale, e al suo posto il *chatbot* risponde con un messaggio esplicativo del motivo per cui non è possibile rispondere a tale domanda, indicando tutte le violazioni che essa contiene.

Dai *test* effettuati, Llama Guard 3 si è dimostrato capace di bloccare in modo molto efficace tutti gli input che lo richiedevano, soddisfacendo pienamente le aspettative.

5.4.2 Zero-shot classification

Nel caso in cui la richiesta dell'utente superi il controllo di moderazione dei contenuti, è opportuno effettuare un ulteriore controllo per verificare se il contesto selezionato dal processo di *information retrieval* sia inerente con la domanda posta dall'utente. Questa verifica addizionale viene implementata tramite un LLM specializzato nella *task* di *zero-shot classification*¹². Ciò consiste nel riconoscere e classificare dati in categorie, senza aver necessariamente incontrato degli esempi di queste durante la fase di addestramento del modello in questione.

Il modello di *zero-shot classification* utilizzato nel progetto di stage si chiama

MoritzLaurer/mDeBERTa-v3-base-xnli-multilingual-nli-2mil7¹³

nel portale HuggingFace: esso si basa su BERT ed è una versione modificata del modello multilingue pre-addestrato mDeBERTa-v3-base¹⁴ di Microsoft. Il modello

¹⁰MLCommons | Better AI for Everyone. URL: <https://mlcommons.org/>.

¹¹MLCommons AI Safety v0.5 Benchmark POC Taxonomy of Hazards. URL: <https://drive.google.com/file/d/1V8KFfk8awaAXc83nZZzDV2bHgPT8jbJY/view>.

¹²What is Zero-Shot Classification? URL: <https://huggingface.co/tasks/zero-shot-classification>.

¹³microsoft/mdeberta-v3-base. URL: [MoritzLaurer/mDeBERTa-v3-base-xnli-multilingual-nli-2mil7](https://huggingface.co/MoritzLaurer/mDeBERTa-v3-base-xnli-multilingual-nli-2mil7).

¹⁴microsoft/mdeberta-v3-base. URL: <https://huggingface.co/microsoft/mdeberta-v3-base>.

utilizzato è stato ottenuto tramite *fine-tuning* da mDeBERTa-v3-base su *dataset* che contengono più di 2.7 milioni di terne formate da premessa, ipotesi ed etichetta (*implicazione, contraddizione o nessuna*) nelle 27 lingue più parlate.

Nell'applicativo sviluppato, la *zero-shot classification* viene utilizzata per decidere se allegare o meno le fonti consultate alla risposta generata dal modello di *question answering* nel messaggio finale del *chatbot*. Infatti, dato che il modello di *question answering* è configurato in modo tale da non generare una risposta esaustiva qualora l'utente effettuasse una richiesta estranea al contesto, non avrebbe senso includere le fonti in un messaggio di questo tipo. In particolare, il modello di *zero-shot classification* si occupa di assegnare alla risposta generata un'etichetta tra "pertinente" e "impossibile": se essa viene classificata come "pertinente" allora, insieme al messaggio, verranno mostrate anche le fonti, altrimenti verrà visualizzato soltanto il messaggio.

5.4.3 Altri casi eccezionali

Ulteriori casi eccezionali sono dati da una ricerca non andata a buon fine, ovvero che non ha recuperato nessun *chunk* di documentazione da allegare come contesto al LLM di *question answering*. Quest'eventualità si può verificare prevalentemente in due occasioni:

- l'utente ha inviato una richiesta che non comprende nessuna parola chiave presente nel testo della documentazione caricata oppure la ricerca semantica non ha rilevato nessuna sezione affine alla domanda posta. Solitamente ciò avviene a causa di pesanti errori di battitura;
- si è verificata un'anomalia durante l'esecuzione di uno degli algoritmi di ricerca utilizzati dal sistema di *information retrieval*. Solitamente ciò si verifica a causa di un errore di programmazione o per l'impossibilità di stabilire la connessione al *database*.

In entrambi i casi, il sistema risponderà con un messaggio finalizzato ad informare l'utente dell'impossibilità di generare una risposta e presentando il problema che si è riscontrato.

Capitolo 6

Progettazione e implementazione

6.1 Introduzione

La progettazione e l'implementazione del prodotto *software* sono state, senza alcun dubbio, le attività più impegnative e dispendiose in termini di tempo durante il periodo di tirocinio.

Dalla definizione dell'architettura del sistema alla codifica vera e propria, ogni passaggio ha comportato numerosi cicli di riflessione, *refactoring*_G e verifica con l'obiettivo di sviluppare un applicativo robusto e incline ad eventuali espansioni future, ma al contempo garantire che il prodotto finale rispondesse pienamente alle aspettative di Zucchetti S.p.A., soddisfacendo ogni requisito definito nell'attività di analisi dei requisiti (capitolo 3).

6.2 Tecnologie e strumenti utilizzati

La scelta delle tecnologie da utilizzare per un progetto *software* è un aspetto cruciale che influenza direttamente l'intero sviluppo dello stesso. Per questo motivo, in questo progetto di stage, la selezione delle tecnologie e degli strumenti è stata il risultato di un'approfondita analisi del problema da affrontare.

In particolare, come anche richiesto dal tutor aziendale, è importante tenere conto del caso in cui l'utente carichi nel *server* un *file* di documentazione di grandi dimensioni. Anche in tale scenario, dunque, l'applicativo dovrà garantire un funzio-

namento efficiente, assicurando che il caricamento, l'elaborazione e il recupero delle informazioni avvengano in tempi ragionevoli. Gran parte della scelta tecnologica, quindi, è stata condotta in funzione del caso appena descritto.

Di seguito vengono presentate le tecnologie e gli strumenti che sono stati utilizzati per lo sviluppo del progetto e le ragioni alla base della loro scelta.

Python

Il linguaggio scelto per la codifica del progetto è Python. Si tratta di un linguaggio di programmazione interpretato che presenta una sintassi moderna dalla graduale curva di apprendimento. È stato scelto questo linguaggio principalmente per la vastità di librerie che permettono di interfacciarsi con gli strumenti di intelligenza artificiale (come ad esempio i LLM) tramite API semplici ed intuitive. Inoltre, è stato consigliato l'utilizzo della libreria Txtai dal tutor aziendale, che è, appunto, una libreria per Python.



Figura 6.1: Logo di Python

Hugging Face

Hugging Face è una piattaforma online che mette a disposizione degli sviluppatori modelli di intelligenza artificiale pre-addestrati, *dataset* e altre risorse. Questo portale si è rivelato utile per il progetto per accedere ai LLM per effettuare il *sentence-embedding* e la *zero-shot classification*. In aggiunta, i modelli di Hugging Face si integrano perfettamente con la libreria Txtai.



Figura 6.2: Logo di Hugging Face

Txtai

Txtai è una libreria *open-source* per Python che implementa un *database vettoriale*, un *database relazionale*_G, un *database a grafo*_G, permette l'interazione con LLM di ogni tipo e molto altro. Questa libreria è stata utilizzata principalmente per utilizzare i modelli di intelligenza artificiale presenti su Hugging Face, per implementare il *knowledge graph* e per calcolare i vettori di *embedding*. Per quanto riguarda la persistenza di questi ultimi, invece, non è stato utilizzato il database messo a disposizione da Txtai ma sono stati salvati in un database PostgreSQL, come verrà descritto successivamente (sezione 6.4.2).



Figura 6.3: Logo di Txtai

Groq

Groq è una società americana che fornisce delle API gratuite che consentono una comunicazione quasi istantanea con alcuni modelli di intelligenza artificiale generativi molto pesanti che operano in *cloud*. Alcuni esempi di LLM che mette gratuitamente a disposizione Groq sono Llama 3 (nelle versioni da 8 miliardi, 70 miliardi o 405 miliardi di parametri) e Mixtral 8x7B, entrambi utilizzati nel progetto.



Figura 6.4: Logo di Groq

NetworkX e Matplotlib

NetworkX e Matplotlib sono due librerie per Python. NetworkX consente la creazione, la manipolazione e l'analisi di reti complesse, mentre Matplotlib offre delle API intuitive che permettono la creazione di grafici di ogni tipo.

Queste librerie sono state utilizzate in combinazione per testare la creazione del *knowledge graph* e la sua *query* di attraversamento tramite la visualizzazione grafica dello stesso.



Figura 6.5: Loghi di NetworkX e Matplotlib

PostgreSQL

PostgreSQL (spesso abbreviato come “*Postgres*”) è uno dei più popolari ed utilizzati *Database Management System (DBMS)* *open-source*. Nel progetto è stato impiegato per la creazione e la gestione del *database* utilizzato per memorizzare i vettori di *embedding* calcolati e per memorizzare i *chunk* del testo del *file* di documentazione caricato dall'utente.



Figura 6.6: Logo di PostgreSQL

Psycopg

Psycopg è la libreria più popolare utilizzata per la comunicazione con i *database* PostgreSQL in Python. Le sue principali caratteristiche comprendono la *thread safety* (diversi *thread* possono condividere la stessa connessione), l'efficienza e la sicurezza (infatti questa libreria è per lo più implementata in linguaggio C come *wrapper* di libpq)



Figura 6.7: Logo di Psycopg

PyMuPDF

PyMuPDF è una libreria per Python ad alte prestazioni che permette l'estrazione del testo, l'analisi, la conversione e la manipolazione di documenti PDF. Nel progetto di stage questo strumento è stato utilizzato per estrarre il contenuto testuale della documentazione nel caso l'utente la carichi come *file* PDF.

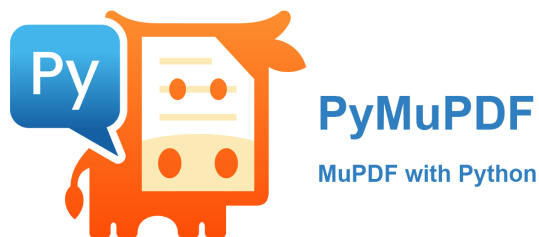


Figura 6.8: Logo di PyMuPDF

Streamlit

Streamlit è un *framework open-source* per Python utilizzato per lo sviluppo di applicazioni web di *data science* oppure basate sull'intelligenza artificiale. La scelta di questo *framework* deriva dalla semplicità d'uso delle API che Streamlit mette a disposizione, rendendo quindi meno dispendioso in termini di tempo lo sviluppo dell'interfaccia grafica del prodotto. In questo modo è stato possibile dedicare la maggior parte delle risorse nella progettazione del sistema e nello studio degli algoritmi coinvolti.



Figura 6.9: Logo di Streamlit

6.3 Progettazione

La progettazione¹ (in inglese *design*) è una fase fondamentale del *ciclo di vita del software*_G. Sulla base dei requisiti prodotti dall'analisi, essa definisce come tali requisiti saranno soddisfatti, entrando nel merito della struttura che dovrà essere data al sistema *software* che deve essere realizzato.

La prima fase della progettazione è la *progettazione ad alto livello*: con essa viene definita l'architettura *software* dell'applicativo, ovvero la struttura complessiva del sistema in termini dei principali moduli di cui esso è composto e le relazioni macroscopiche che sussistono fra di essi. Segue poi la *progettazione di dettaglio*, con la quale viene descritto il sistema da implementare in maniera molto vicina alla codifica, descrivendo non solo le classi in astratto ma anche i loro attributi e metodi, con relativi tipi e *signature*_G.

Per descrivere la struttura del sistema nel dettaglio viene fatto uso dei diagrammi delle classi², ovvero dei diagrammi UML che utilizzano il paradigma *object-oriented*

¹Sommerville, *Software Engineering*.

²Booch, Rumbaugh e Jacobson, *Unified Modeling Language User Guide*.

per descrivere le classi che faranno parte del codice del prodotto *software* in questione e le relazioni tra di esse.

6.3.1 Design pattern utilizzati

Un *design pattern*³ nomina, astrae e individua gli aspetti chiave di una struttura di *design* comune utilizzata per organizzare il codice in modo che sia riutilizzabile. Esso identifica le classi e le istanze coinvolte, i loro ruoli e collaborazioni, e definisce le responsabilità di ciascuna di esse. Ogni *design pattern* si concentra su un particolare problema di *design* orientato agli oggetti, descrivendo quando esso può essere applicato e le conseguenze del suo utilizzo.

Esistono quattro tipi di *design pattern*:

- i *design pattern creazionali* risolvono problematiche inerenti all'istanziamento degli oggetti;
- i *design pattern strutturali* risolvono problematiche inerenti alla struttura delle classi e degli oggetti;
- i *design pattern comportamentali* si occupano dell'interazione tra gli oggetti;
- i *design pattern architetturali* operano ad un livello più ampio rispetto alle tipologie di *pattern* precedentemente presentate. Essi esprimono degli schemi di base per strutturare l'architettura di un sistema *software*, e vengono inoltre descritti i sottosistemi che compongono questi schemi, insieme ai loro ruoli e relazioni tra di essi.

L'utilizzo dei *design pattern* è particolarmente utile in progetti di notevoli dimensioni: ciò comporta un'organizzazione del codice più efficace e una manutenibilità più agevole ed efficiente dello stesso, con conseguente maggior propensione all'estensibilità dell'intero progetto.

Pur essendo un progetto di dimensioni piuttosto contenute, quanto sviluppato nel

³Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1995.

periodo di tirocinio ha una complessità sufficientemente grande da rendere apprezzabile l'impiego di alcuni *design pattern*. Di seguito vengono presentati i *pattern* utilizzati e le motivazioni alla base della loro applicazione.

6.3.1.1 Model-View-Update

L'architettura *software* del prodotto è data dal *design pattern* architetturale implementato dal *framework* Streamlit, ovvero il *Model-View-Update*⁴⁵⁶ (MVU), anche conosciuto come *architettura Elm*, dal momento che si è sviluppato con il linguaggio funzionale Elm⁷. Con il MVU l'implementazione di programmi ad interfaccia grafica (in inglese *Graphical User Interface* (GUI)) risulta più semplice ed immediata.

Questo *design pattern* è simile al *Model-View-Controller* (MVC): in quest'ultimo il *Controller* risponde agli input dell'utente e aggiorna di conseguenza l'interfaccia grafica (*View*) e il modello (*Model*). Nel MVU, invece, il “*Controller*” non manipola la *View* in modo diretto, ma solo il *Model* può essere modificato: ciò permette una semplificazione del “*Controller*”, che consiste infatti in una funzione, chiamata solitamente *funzione Update*. La *View* viene ridisegnata da una funzione integrata nel *framework* MVU, tipicamente chiamata *funzione View*, che attraversa una struttura dati ad *albero*_G, contenuta del *Model*, chiamata *Virtual-DOM*⁸ (*Virtual Document Object Model*), che descrive la struttura della pagina da renderizzare a schermo (ciascun nodo dell'albero rappresenta un *widget*, ovvero un elemento grafico, da renderizzare). Quando la funzione *Update*, in risposta ad un evento, modifica il *Virtual-DOM*, la funzione *View* attraversa il *Virtual-DOM* aggiornato e ricostruisce nuovamente l'intera *View*. Per rendere questo processo più efficiente, spesso viene usato un algoritmo *diff*_G per confrontare il nuovo *Virtual-DOM* con quello prece-

⁴*Model-View-Update (MVU) – How Does It Work?* URL: <https://thomasbandt.com/model-view-update>.

⁵*Python and the Model-View-Update GUI Revolution.* URL: <https://xc-jp.github.io/blog-posts/2022/11/22/python-model-view-update-frameworks.html>.

⁶*Model View Update - Part 1.* URL: <https://elmprogramming.com/model-view-update-part-1.html>.

⁷*Elm - delightful language for reliable web applications.* URL: <https://elm-lang.org/>.

⁸*Virtual DOM.* URL: <https://elmprogramming.com/virtual-dom.html>.

dente, per poi ridisegnare soltanto le parti che differiscono.

Dal momento che è il *framework* ad occuparsi automaticamente dell'ispezione dello stato del *Virtual-DOM* e di ricostruire la *View*, ciò permette al programmatore di concentrarsi nella manipolazione del *Virtual-DOM* in risposta agli input dell'utente nell'interfaccia grafica.

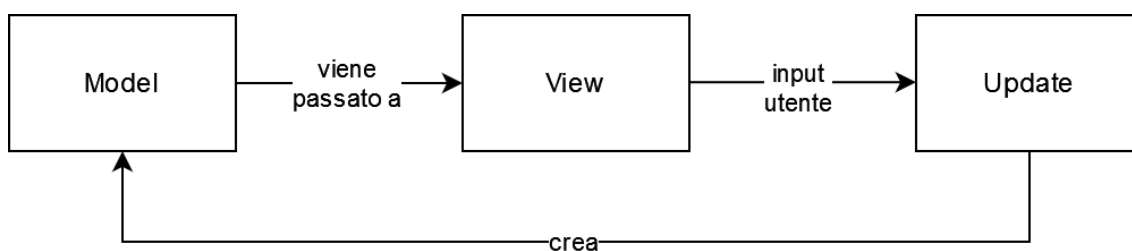


Figura 6.10: Rappresentazione del flusso di dati nell'architettura MVU

6.3.1.2 Adapter

*Adapter*⁹¹⁰ è un *design pattern* strutturale che permette ad oggetti con interfacce incompatibili tra loro di collaborare, adattando l'interfaccia di un oggetto, chiamato *adaptee*, in modo che possa essere compatibile con l'oggetto che lo utilizza, chiamato *client*. Ciò avviene grazie ad uno speciale oggetto, chiamato, appunto, *adapter*, che agisce come *wrapper* per l'oggetto *adaptee*. Il *client*, dunque, non utilizzerà in modo diretto l'*adaptee* ma chiamerà i metodi dell'*adapter*, il quale passerà la richiesta all'*adaptee* nel formato e nell'ordine che quest'ultimo si aspetta. Oltre a convertire l'interfaccia di un oggetto, l'*adapter* consente anche di nascondere la complessità delle operazioni dell'oggetto adattato.

Esistono due varianti del *design pattern Adapter*, ovvero il *Class adapter* e l'*Object adapter*. La differenza tra le due alternative consiste nel modo in cui l'*adaptee* viene adattato: nel primo caso si sfrutta l'ereditarietà (l'*adapter* estende l'*adaptee*), mentre nel secondo si sfrutta la composizione (l'*adapter* è composto dall'*adaptee*).

I diagrammi delle classi che descrivono questo *design pattern* nelle due varianti sono i seguenti:

⁹Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*.

¹⁰*Adapter*. URL: <https://refactoring.guru/design-patterns/adapter>.

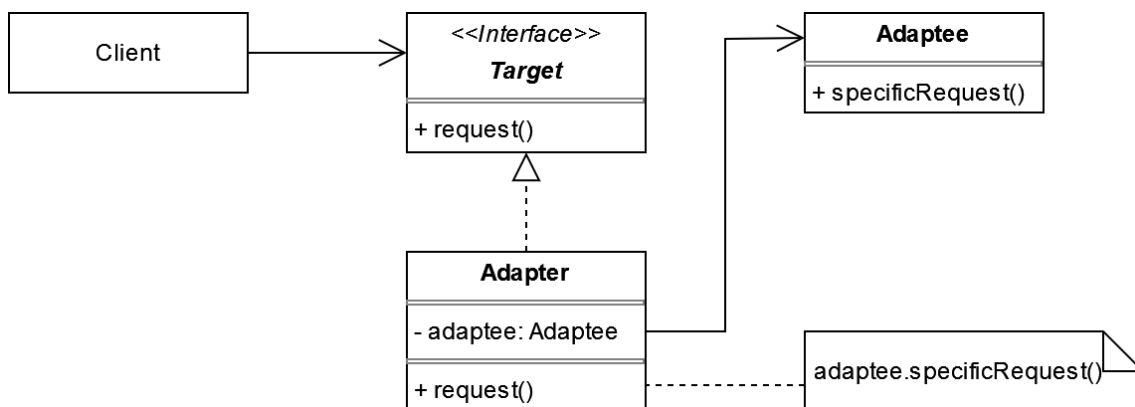


Figura 6.11: Diagramma delle classi del design pattern Object adapter

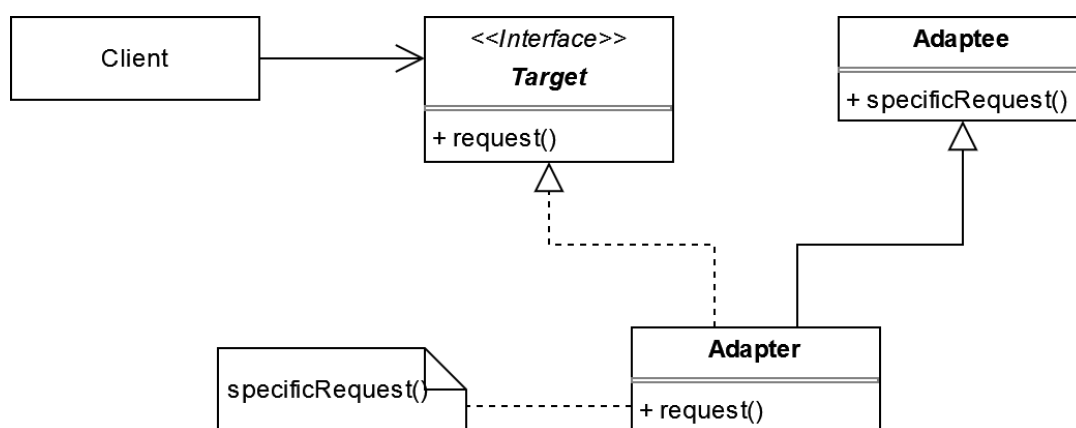


Figura 6.12: Diagramma delle classi del design pattern Class adapter

Nel progetto di stage si è fatto ampio utilizzo di questo *pattern* nella sua versione *Object adapter*, in quanto la composizione è una relazione che comporta una dipendenza meno forte rispetto all’ereditarietà. In particolare, è stato utilizzato nelle seguenti occasioni:

- per adattare l’interfaccia di PyMuPDF all’interfaccia della classe astratta Document tramite la classe PDFDocument;
- per adattare l’interfaccia di Psycopg2 alla classe Chatbot tramite la classe *adapter* DBAccess;
- per adattare l’interfaccia delle API del servizio Groq alla classe Chatbot e nascondere la complessità tramite la classe *adapter* QuestionAnsweringLLM;

- per adattare l'interfaccia di `Txtai` alla classe `Chatbot` e nascondere la complessità tramite le classi *adapter* `KnowledgeGraph`, `ZeroShotLLM` e `SimilarityLLM`.

6.3.1.3 Strategy

*Strategy*¹¹¹² è un *design pattern* comportamentale che permette di definire una famiglia di algoritmi, ognuno definito nella propria classe, chiamata *strategy*, e renderli interscambiabili.

La classe che lavora con questi algoritmi è chiamata *context*, che mantiene un riferimento ad una delle strategie. L'esecuzione degli algoritmi, dunque, viene delegata agli oggetti *strategy*. Inoltre, l'oggetto *client* si occupa di passare l'algoritmo più appropriato al *context*: quest'ultimo infatti potrebbe anche non conoscere quali sono le classi *strategy* concrete, dal momento che esso lavora con ciascuna di esse tramite un'unica interfaccia generica, implementata da tutte le classi *strategy*, che espone un solo metodo, utilizzato per avviare l'esecuzione dell'algoritmo selezionato.

Con questo *design pattern* risulta molto agevole aggiungere una nuova strategia: è infatti sufficiente creare una nuova classe che implementa l'unico metodo dell'interfaccia comune. Ciò rende quindi il sistema più facilmente estensibile e più resistente ai cambiamenti. Il diagramma delle classi che descrive questo *design pattern* è il seguente:

¹¹Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*.

¹²*Strategy*. URL: <https://refactoring.guru/design-patterns/strategy>.

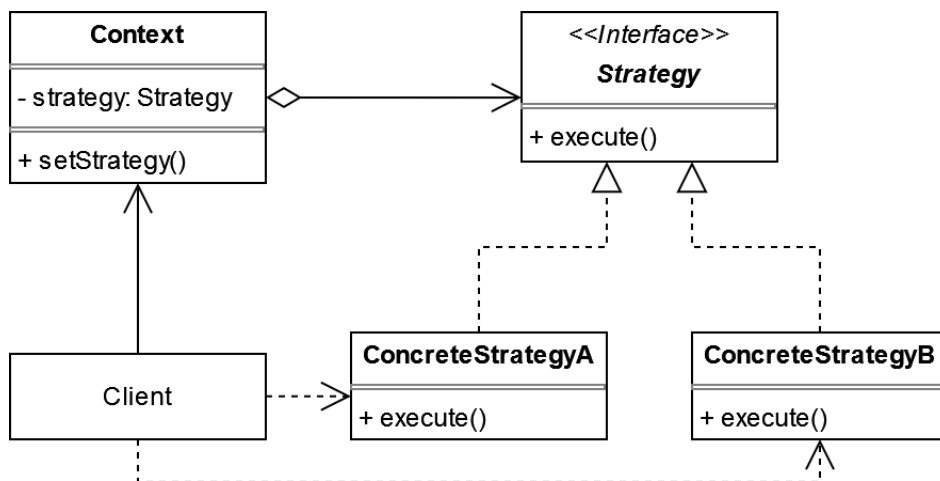


Figura 6.13: Diagramma delle classi del design pattern Strategy

Nel progetto di stage, il *pattern Strategy* è stato utilizzato per gestire la selezione dei diversi algoritmi di *rank fusion* e per facilitare l'eventuale aggiunta futura di nuovi algoritmi di questo tipo.

6.3.1.4 Template method

*Template method*¹³¹⁴ è un *design pattern* comportamentale che consiste nel suddividere un algoritmo in diversi *step* atomici, ciascuno implementato in un metodo di una superclasse, e utilizzarli per definirne la struttura in un altro metodo chiamato *template method*. A questo punto è possibile definire delle classi che estendono tale superclasse per fornire delle implementazioni diverse dell'algoritmo contenuto nel *template method* facendo *override* di alcuni dei metodi della superclasse che definiscono gli *step* di questo algoritmo.

Gli *step* dichiarati nella superclasse possono essere dei metodi già implementati dalla superclasse stessa oppure possono essere dei metodi astratti. Le sottoclassi devono obbligatoriamente fornire delle implementazioni ai metodi astratti, mentre non è obbligatorio fare *override* dei metodi non astratti.

Il diagramma delle classi che rappresenta questo *design pattern* è il seguente:

¹³Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*.

¹⁴*Template method*. URL: <https://refactoring.guru/design-patterns/template-method>.

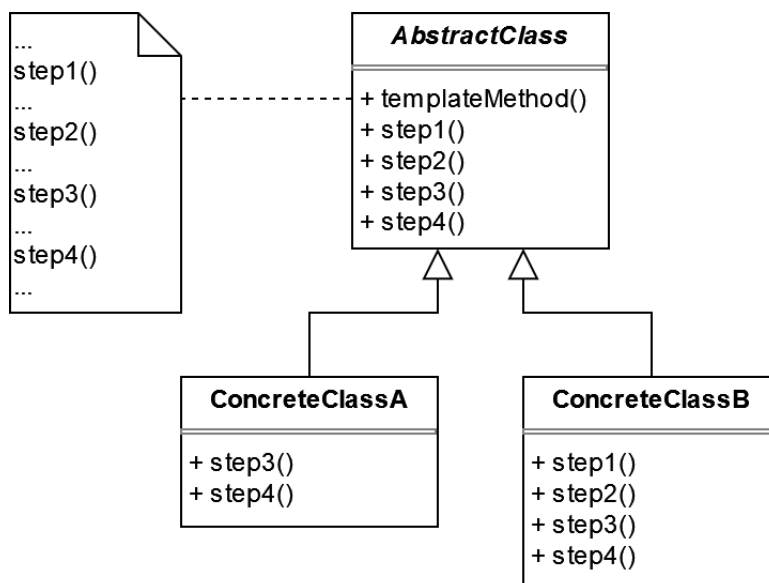


Figura 6.14: Diagramma delle classi del design pattern Template method

Il *pattern Template method* è stato utilizzato nel progetto di stage per fornire delle varianti all’algoritmo di *chunking* dei documenti (la suddivisione avviene in maniera diversa a seconda del formato del documento in questione). In particolare, il metodo `get_docs` della classe `Document`, ovvero il *template method*, viene suddiviso nei due *step* `get_sections` e `chunk_text`, e le due classi `PDFDocument` e `SAMDocument`, che estendono `Document`, forniscono una diversa implementazione del metodo (astratto) `get_sections`.

6.3.1.5 Dependency injection

*Dependency injection*¹⁵ è un *design pattern* architetturale che permette di ridurre l’accoppiamento del codice del progetto a cui viene applicato. Questa tecnica consiste nel sollevare un oggetto (*client*) che richiede l’utilizzo di altri oggetti dalla responsabilità di crearli, limitandosi dunque soltanto a “dichiarare” le sue dipendenze, rendendole così esplicite. Un *client* a cui non è applicato il *pattern Dependency injection* appare di questo tipo:

¹⁵*Dependency injection*. URL: https://en.wikipedia.org/wiki/Dependency_injection.

```
public class Client {  
    private Dependency dependency;  
    Client() {  
        this.dependency = new Dependency();  
    }  
}
```

Codice 6.1: Esempio di client senza Dependency injection

Con l'uso della *Dependency injection*, al *client* vengono fornite (“iniettate”) le sue dipendenze da un codice esterno, chiamato *injector*, che si occupa anche di costruire tutti gli oggetti di cui il *client* necessita.

Un altro vantaggio nell'impiego del *pattern Dependency injection* consiste nella facilità di effettuare i *test di unità*. Infatti in essi le dipendenze possono essere facilmente sostituite con *mock*_G o *stub*_G: non dovendo creare direttamente le dipendenze all'interno delle classi, è possibile passare delle versioni controllate di esse, migliorando così la copertura e la qualità dei *test*.

Nel progetto di stage è stato utilizzato questo *pattern* nella variante *Constructor injection*, ovvero dichiarando le dipendenze dei *client* nel loro metodo costruttore. Questo assicura che il *client* sia sempre in uno stato valido, dal momento che non è possibile istanziarlo senza le dipendenze di cui ha bisogno.

```
public class Client {  
    private Dependency dependency;  
    Client(Dependency dependency) {  
        this.dependency = dependency;  
    }  
}
```

Codice 6.2: Esempio di client con Constructor injection

Esiste un'altra variante del *pattern Dependency injection*, cioè *Setter injection*. Con il *Setter injection* le dipendenze vengono passate al *client* tramite appositi

metodi setter: ciò consente la manipolazione di queste ultime da parte dell'*injector* in qualsiasi momento, ma è più difficile garantire che tutte le dipendenze siano state correttamente fornite al *client* prima del suo utilizzo effettivo.

```
public class Client {  
    private Dependency dependency;  
    public void setDependency(Dependency dependency) {  
        this.dependency = dependency;  
    }  
}
```

Codice 6.3: Esempio di client con Setter injection

Tuttavia, l'adozione di questo *design pattern* porta anche dei difetti. Alcuni di questi sono i seguenti:

- separando la costruzione degli oggetti il codice può diventare più difficile da tracciare;
- quando esiste un'ovvia configurazione di default per alcune dipendenze del *client*, diventa oneroso separare la loro costruzione;
- tipicamente richiede uno sforzo iniziale di codifica maggiore.

6.4 Implementazione

L'*implementazione*¹⁶ è il processo di realizzazione effettiva di un programma eseguibile a partire dal suo *design*, definito nella fase di progettazione. Essa avviene tramite la scrittura del codice e l'assemblaggio delle varie componenti che andranno a far parte del sistema.

La progettazione e l'implementazione sono attività strettamente legate, ed è opportuno tenere in considerazione i problemi e le eventuali limitazioni legate all'implementazione quando si sviluppa il *design* del sistema *software*.

¹⁶Sommerville, *Software Engineering*.

La *repository* contenente il codice sorgente del progetto si trova su GitHub al seguente link:

<https://github.com/FabioMeneghini/ChatHelp>

Di seguito vengono discusse le scelte implementative.

6.4.1 MVU con Streamlit

Streamlit¹⁷¹⁸ fornisce un *web server* con un *Virtual-DOM* proprietario, che viene manipolato dal programmatore tramite le API di Streamlit, dichiarando l'esistenza di determinati *widget* nello *script* Python. Il *Virtual-DOM* viene poi convertito nel vero e proprio DOM del *browser*, visualizzando così la pagina *web*. L'intero *script* rappresenta la funzione *Update* del *framework* MVU: ogni input da parte dell'utente nell'interfaccia grafica viene trasmesso al *server* Streamlit, e quest'ultimo, in risposta, riesegue l'intero *script* Python per aggiornare il modello (in Streamlit, questa operazione viene anche chiamata *rerun*).

Il *server* Streamlit possiede un *interprete* per Python realizzato ad hoc, ed ogni *rerun* avviene in questo ambiente personalizzato. Ciò consente al *server* di effettuare alcune ottimizzazioni, ad esempio:

- eventuali librerie o dipendenze sono caricate una sola volta;
- gestisce lo stato delle sessioni in modo automatico, anche quando più *browser web* sono connessi al *server*;
- permette di renderizzare parti dell'interfaccia grafica anche prima del termine dell'esecuzione dello *script*;
- gestisce le eccezioni che occorrono durante l'aggiornamento della GUI, e mostra lo *stack trace*_G direttamente come *widget* nella pagina *web*. Ciò rende più semplice l'individuazione di errori di programmazione e la relativa correzione.

¹⁷*Python and the Model-View-Update GUI Revolution.*

¹⁸*Streamlit documentation.* URL: <https://docs.streamlit.io/>.

6.4.2 Algoritmi di ricerca con PostgreSQL

Nonostante la libreria *Txtai* implementi un database ed offra funzionalità sia per eseguire ricerche basate su BM25 sia per ricerche semantiche, si è comunque deciso di implementare queste ultime tramite *query* a un database PostgreSQL, nel quale vengono anche salvati i *chunk* che compongono il testo del *file* caricato nell'applicativo. Questa scelta è motivata dalla capacità di PostgreSQL di gestire in modo efficiente database costituiti da tabelle contenenti milioni di record, garantendo così un'elaborazione efficace ed efficiente di *file* di documentazione di grandi dimensioni da parte del *software* sviluppato, così come un rapido recupero delle informazioni rilevanti con la richiesta dell'utente tramite opportune *query* al database.

Nello specifico, il database impiegato per il progetto è composto da due tabelle:

- la prima, denominata *docs*, è utilizzata per contenere i *chunk* di testo e i relativi vettori di *embedding*;
- la seconda, chiamata *idf*, contiene gli indici IDF di tutte le parole che appaiono nel testo della documentazione caricata.

Le due tabelle sono indipendenti tra loro. La loro struttura è la seguente:

docs			idf		
PK	<u>codice</u>	smallint	PK	<u>lexeme</u>	text
	testo	varchar(4095)		cnt	int
	vettore	vector		n	int
	sezione	varchar(255)		idf	numeric

Figura 6.15: Tabelle docs e idf

6.4.2.1 Query di ricerca semantica e pgvector

Il campo *vettore* della tabella *docs* ha tipo *vector*: questo tipo non fa parte di quelli standard presenti in PostgreSQL ma è stato ottenuto grazie a *pgvector*¹⁹. Si

¹⁹ *Open-source vector similarity search for Postgres*. URL: [https://github.com/pgvector/pgvector/](https://github.com/pgvector/pgvector)

tratta di un'estensione *open-source* che permette di memorizzare vettori di *embedding* in database PostgreSQL grazie all'introduzione del tipo di dato `vector`. Inoltre, `pgvector` mette a disposizione dei nuovi operatori con i quali è possibile calcolare distanza tra due vettori. Questi sono:

- `<->` per calcolare la distanza euclidea;
- `<#>` per calcolare il prodotto interno;
- `<=>` per calcolare la *cosine similarity*;
- `<+>` per calcolare la distanza di Manhattan;
- `<~>` per calcolare la distanza di Hamming;
- `<%>` per calcolare la distanza di Jaccard.

Con l'utilizzo di questa estensione è quindi molto semplice costruire una *query* di ricerca semantica con la *cosine similarity*. La *query* utilizzata per implementare questo tipo di ricerca nel progetto è la seguente:

```
SELECT codice, vettore <=> '{vector}' AS score, testo, sezione
FROM docs
ORDER BY score
LIMIT 20;
```

Codice 6.4: Query di ricerca semantica con cosine similarity

dove `{vector}` è il vettore di *embedding* della richiesta inserita dall'utente.

I vettori di *embedding* relativi ai *chunk* e alla richiesta dell'utente vengono calcolati tramite le API di `Txtai` dal modello di *sentence similarity* selezionato da Hugging Face.

6.4.2.2 Query di ricerca lessicale

Per implementare la *query* di ricerca lessicale con BM25 in PostgreSQL si è fatto uso di due tipi di dati progettati per supportare questo genere di operazioni, ovvero

`tsvector` e `tsquery`²⁰²¹.

Il tipo `tsvector` rappresenta un documento in un modo tale da ottimizzare la ricerca testuale. Un valore di tipo `tsvector` è una lista ordinata di distinti *lessemi* (utilizzando la stessa nomenclatura della documentazione di PostgreSQL), ovvero parole a cui è stato applicato lo *stemming*. L'ordinamento e la rimozione dei duplicati e delle *stop words* avviene in automatico. Assieme a ciascun lessema, in un `tsvector` viene inoltre codificata la posizione di ciascun occorrenza di tale lessema all'interno del documento in questione.

Il tipo `tsquery`, similamente, rappresenta una *query* testuale. Un oggetto di tipo `tsquery` memorizza i lessemi che devono essere cercati e possono essere combinati tramite operatori logici (“&” per `and`, “|” per `or` e “!” per `not`).

In PostgreSQL è possibile effettuare una ricerca basata sul testo utilizzando direttamente i `tsvector` e le `tsquery` in combinazione con l'operatore `@@`.

Si consideri la tabella `pgweb`, composta dai campi `id`, `title` e `body`, che rappresenta pagine web. Un esempio di *query* di ricerca lessicale in PostgreSQL, riportato anche nella relativa documentazione, è il seguente:

```
SELECT title
FROM pgweb
WHERE to_tsvector(body) @@ to_tsquery('friend');
```

Codice 6.5: Esempio di query di ricerca lessicale

La *query* di esempio riportata restituisce il *ranking* formato dal campo `title` di tutti i *record* della tabella `pgweb` che contengono almeno un termine con lo stesso lessema di “`friend`”.

Tuttavia, in questo modo non viene utilizzato l'algoritmo BM25, che è un requisito che l'applicativo deve soddisfare, perciò non è stato possibile applicare in modo diretto questa soluzione. Piuttosto, il testo dei documenti presenti nella tabella

²⁰PostgreSQL: Documentation: 17: 8.11. Text Search Types. URL: <https://www.postgresql.org/docs/current/datatype-textsearch.html>.

²¹PostgreSQL: Documentation: 17: Chapter 12. Full Text Search. URL: <https://www.postgresql.org/docs/current/textsearch.html>.

`docs` viene convertito in un oggetto di tipo `tsvector` per effettuare lo *stemming* dei termini, contarne la molteplicità e selezionare i soli documenti che contengono parole che corrispondono agli stessi lessemi dei termini facenti parte della richiesta specificata dall'utente, convertita ad un oggetto di tipo `tsquery`, in modo da agevolare l'implementazione dell'algoritmo BM25 tramite *query* SQL. Come anticipato precedentemente, per determinare l'indice IDF per ogni lessema presente nei documenti nella tabella `docs` viene fatto uso di una seconda tabella, chiamata `idf`: ciò permette ridurre il tempo di calcolo dei punteggi BM25 determinando il valore degli IDF soltanto la prima volta, al momento dell'inserimento di un nuovo *file* di documentazione, per poi memorizzarli, in modo da poterli utilizzare direttamente ad ogni nuova ricerca che l'utente desidera effettuare.

Capitolo 7

Verifica e validazione

7.1 Introduzione

La verifica di un prodotto *software* rappresenta un'attività di fondamentale importanza nel ciclo di vita dello stesso. Essa avviene principalmente tramite due tipi di analisi, ovvero tramite l'*analisi statica* e l'*analisi dinamica*.

Di seguito vengono descritte nel dettaglio le modalità secondo cui sono state condotti questi due tipi di analisi dell'applicativo sviluppato durante il periodo di tirocinio.

7.2 Analisi statica

L'*analisi statica* esamina il codice sorgente del programma senza richiederne l'esecuzione. Essa è utile per identificare eventuali vulnerabilità nel codice e per migliorarne la qualità e la leggibilità, imponendo il rispetto di determinate convenzioni che aumentano la manutenibilità dell'applicazione per eventuali sviluppi futuri.

Il *framework* utilizzato nel progetto di stage per svolgere l'analisi statica è `pylint`¹. Questo *tool* assegna un punteggio da 0 a 10 a ciascun *file* analizzato, che ne indica la qualità del codice secondo i parametri specificati nel *file* di configurazione `.pylintrc` di `pylint`.

In particolare, nel caso di questo progetto, per ciascun *file* viene verificato che:

¹*PyLint*. URL: <https://pypi.org/project/pylint/>.

- il numero di caratteri in una singola linea di codice sia inferiore o uguale a 150;
- il numero di linee di codice di un singolo modulo sia inferiore o uguale a 200;
- la stringa usata come unità di indentazione sia composta da quattro spazi;
- il nome delle funzioni, delle variabili, degli attributi e dei parametri segua la convenzione dello `snake_case`;
- il nome delle costanti segua la convenzione dell'`UPPER_CASE`;
- il nome delle classi segua la convenzione del `PascalCase`;
- il numero di parametri di una funzione o di un metodo sia inferiore o uguale a 10;
- il numero di variabili locali di una funzione o di un metodo sia inferiore o uguale a 20;
- il numero di *statement* `return` o `yield` di una funzione o di un metodo sia inferiore o uguale a 5;
- il numero di *branch* di una funzione o di un metodo sia inferiore o uguale a 10;
- il numero di *statement* di una funzione o di un metodo sia inferiore o uguale a 80;
- il numero di metodi pubblici di una classe sia compreso tra 1 e 10;
- il numero di espressioni logiche in uno *statement* `if` sia inferiore o uguale a 5;
- il numero di blocchi di codice annidati di un metodo o una funzione sia inferiore o uguale a 5.

Il comando da terminale che permette di eseguire l'analisi statica con `pylint` è il seguente:

```
pylint path/to/file.py
```

I *file* che costituiscono il codice sorgente del progetto di stage sono stati tutti analizzati con `pylint`, e i punteggi ricevuti sono riportati nella tabella sottostante:

File	Punteggio
<code>chatbot.py</code>	10.00
<code>db_access.py</code>	10.00
<code>dbsf_rank_fusion.py</code>	10.00
<code>document.py</code>	10.00
<code>gui_chat.py</code>	10.00
<code>gui_main_window.py</code>	9.62
<code>gui_sidebar.py</code>	9.84
<code>knowledge_graph.py</code>	10.00
<code>pdf_document.py</code>	8.61
<code>question_answering_llm.py</code>	10.00
<code>rank_fusion.py</code>	10.00
<code>rrf_rank_fusion.py</code>	9.60
<code>sam_document.py</code>	10.00
<code>similarity_llm.py</code>	10.00
<code>zero_shot_llm.py</code>	10.00

Tabella 7.1: Punteggi assegnati ai file del progetto da `pylint`

Come è possibile notare, tutti i punteggi si assestano tra 8.61 e 10, con una media di 9.84, il che è un indice di buona qualità del codice.

7.3 Analisi dinamica

L'*analisi dinamica*, più comunemente conosciuta come *testing*, permette di generare maggior confidenza nella correttezza del programma in situazioni tipiche o nella corretta gestione di alcune situazioni di errore. È importante sottolineare che il *testing* non è sufficiente per dimostrare la correttezza dell'applicativo, ma dimostra

piuttosto l'assenza di specifici errori. Tuttavia, un adeguato insieme di *test* rappresenta una stima soddisfacente della correttezza dell'applicativo sviluppato, purché non si tratti di un *sistema critico*_G (in tal caso sono richieste dimostrazioni formali e rigorose).

I *test* definiti per il progetto di stage sono identificati da un codice univoco strutturato nel seguente modo:

T[tipo]-[numero]

dove

- il tipo può essere **U** per i *test* di unità, **I** per quelli di integrazione, **S** per quelli di sistema e **R** per quelli di regressione;
- il numero è un valore numerico progressivo che identifica univocamente un *test* del suo tipo.

Ciascun *test* presente nelle tabelle sottostanti, fatta eccezione per quelli di regressione, riporta una sigla rappresentante il suo stato:

- **P** indica che il *test* è stato implementato e passato;
- **N** indica che il *test* non è stato implementato.

7.3.1 Test di unità

I *test di unità* (in inglese *unit tests*) hanno lo scopo di verificare il comportamento delle unità atomiche che compongono il codice del programma. Nel progetto di stage, questi vengono eseguiti attraverso il *framework* `pytest`², che agevola la scrittura di *test* per Python di ridotta dimensione e con elevata leggibilità. Per eseguire i *test* di unità con `pytest`, è sufficiente definire ciascuno di essi in una funzione il cui nome inizia con la stringa “`test_`” all'interno di un *file* con nome che inizia sempre con “`test_`”, per poi eseguirli con il comando

```
pytest path/to/test_file.py
```

²*pytest documentation*. URL: <https://docs.pytest.org/en/stable/>.

L'esito di ciascun *test* viene determinato dal risultato dello *statement assert*: se il risultato dell'espressione booleana al suo interno è **True** allora il *test* verrà considerato superato, altrimenti verrà considerato non superato e il *framework* stamperà a terminale tutte le informazioni dettagliate riguardanti il *test* fallito.

Di seguito vengono riportati tutti i *test* di unità definiti.

Codice	Descrizione	Stato
TU-1	Verifica che il modello di <i>zero-shot classification</i> classifichi correttamente la risposta ad una domanda non pertinente con il contesto fornito	P
TU-2	Verifica che il modello di <i>zero-shot classification</i> classifichi correttamente la risposta ad una domanda pertinente con il contesto fornito	P
TU-3	Verifica che il metodo costruttore della classe <code>ZeroShotLLM</code> sollevi un'eccezione nel caso gli venga fornito un nome non valido per il modello	P
TU-4	Verifica che il metodo <code>get_vector</code> della classe <code>SimilarityLLM</code> restituisca un vettore di 768 dimensioni	P
TU-5	Verifica che il metodo costruttore della classe <code>SimilarityLLM</code> sollevi un'eccezione nel caso gli venga fornito un nome non valido per il modello	P
TU-6	Verifica il corretto funzionamento dell'algoritmo RRF in una situazione banale	P
TU-7	Verifica il corretto funzionamento dell'algoritmo RRF in una situazione tipica (non banale)	P
TU-8	Verifica il corretto funzionamento dell'algoritmo RRF nel caso di un solo <i>ranking</i> con un solo elemento	P
TU-9	Verifica il corretto funzionamento dell'algoritmo RRF nel caso di più <i>ranking</i> con più elementi con lo stesso punteggio	P
Continua nella prossima pagina...		

Tabella 7.2 – Continuazione della tabella

Codice	Descrizione	Stato
TU-10	Verifica il corretto funzionamento dell'algoritmo DBSF in una situazione banale	P
TU-11	Verifica il corretto funzionamento dell'algoritmo DBSF in una situazione tipica (non banale)	P
TU-12	Verifica il corretto funzionamento dell'algoritmo DBSF nel caso di un solo <i>ranking</i> con un solo elemento	P
TU-13	Verifica il corretto funzionamento dell'algoritmo DBSF nel caso di più <i>ranking</i> con più elementi con lo stesso punteggio	P
TU-14	Verifica che il metodo costruttore della classe <code>QuestionAnsweringLLM</code> sollevi un'eccezione nel caso gli venga fornito un nome non valido per il modello	P
TU-15	Verifica che il metodo <code>check</code> della classe <code>QuestionAnsweringLLM</code> identifichi correttamente le richieste pericolose	P
TU-16	Verifica che i modelli Llama 3, Mixtral e Gemma 2 non rispondano in modo esaustivo ad una domanda non pertinente al contesto fornito	P
TU-17	Verifica che i modelli Llama 3, Mixtral e Gemma 2 rispondano in modo esaustivo ad una domanda pertinente al contesto fornito	P
TU-18	Verifica che il metodo <code>get_path</code> della classe <code>Document</code> restituisca il corretto percorso del <i>file</i> in questione	P
TU-19	Verifica che il metodo <code>get_name</code> della classe <code>Document</code> restituisca il corretto nome del <i>file</i> in questione	P
Continua nella prossima pagina...		

Tabella 7.2 – Continuazione della tabella

Codice	Descrizione	Stato
TU-20	Verifica che il metodo costruttore della classe <code>Document</code> sollevi un'eccezione nel caso gli venga fornito un percorso non valido per il <i>file</i>	P
TU-21	Verifica che il metodo <code>get_docs</code> della classe <code>PDFDocument</code> restituisca il numero atteso di <i>chunk</i>	P
TU-22	Verifica che il contenuto dei <i>chunk</i> estratti dal metodo <code>get_docs</code> della classe <code>PDFDocument</code> sia quello atteso	P
TU-23	Verifica che i <i>chunk</i> estratti dal metodo <code>get_docs</code> della classe <code>PDFDocument</code> abbiano lunghezza inferiore a quella massima impostata	P
TU-24	Verifica che il metodo <code>get_docs</code> della classe <code>SAMDocument</code> restituisca il numero atteso di <i>chunk</i>	P
TU-25	Verifica che il contenuto dei <i>chunk</i> estratti dal metodo <code>get_docs</code> della classe <code>SAMDocument</code> sia quello atteso	P
TU-26	Verifica che i <i>chunk</i> estratti dal metodo <code>get_docs</code> della classe <code>SAMDocument</code> abbiano lunghezza inferiore a quella massima impostata	P
TU-27	Verifica il metodo <code>connect</code> della classe <code>DBAccess</code> impedisca la connessione al database in caso di credenziali errate	P
TU-28	Verifica che il metodo <code>connect</code> della classe <code>DBAccess</code> apra la connessione al database in caso di credenziali corrette	P
TU-29	Verifica che il metodo <code>db_close</code> della classe <code>DBAccess</code> chiuda la connessione	P
TU-30	Verifica che il metodo <code>get_bm25_rank</code> della classe <code>DBAccess</code> restituisca una lista di risultati	P
Continua nella prossima pagina...		

Tabella 7.2 – Continuazione della tabella

Codice	Descrizione	Stato
TU-31	Verifica che il metodo <code>get_embeddings_rank</code> della classe <code>DBAccess</code> restituisca una lista di risultati	P
TU-32	Verifica il corretto funzionamento del metodo <code>calculate_new_document</code> della classe <code>DBAccess</code>	P
TU-33	Verifica il corretto funzionamento del metodo <code>calculate_embeddings</code> della classe <code>DBAccess</code>	P
TU-34	Verifica il corretto funzionamento del metodo <code>calculate_idf</code> della classe <code>DBAccess</code>	P
TU-35	Verifica il corretto funzionamento del metodo <code>get_all_rows</code> della classe <code>DBAccess</code>	P
TU-36	Verifica il corretto funzionamento del metodo <code>get_document_name</code> della classe <code>DBAccess</code>	P
TU-37	Verifica che il metodo <code>get_qa_model_name</code> della classe <code>Chatbot</code> restituisca il nome corretto del modello di <i>question answering</i> impostato	P
TU-38	Verifica che il metodo <code>set_qa_model_name</code> della classe <code>Chatbot</code> imposti correttamente il modello di <i>question answering</i> selezionato	P
TU-39	Verifica il corretto funzionamento del metodo <code>regenerate_kg</code> della classe <code>Chatbot</code>	P
TU-40	Verifica il corretto funzionamento del metodo <code>generate_response</code> della classe <code>Chatbot</code>	P

Tabella 7.2: Tabella dei test di unità

Il *framework* `pytest` permette anche di misurare la *code coverage*, ovvero la percentuale di righe di codice sorgente coperte dai *test* sul totale di righe che compongono l'intero progetto. Per fare ciò è sufficiente posizionarsi nella *root* del progetto e lanciare il comando

```
pytest --cov
```

Con i *test* di unità definiti nella tabella precedente è stata raggiunta una *code coverage* del 91%.

7.3.2 Test di integrazione

I *test di integrazione* hanno lo scopo di verificare che le diverse componenti del sistema interagiscano tra di loro secondo le aspettative. Sebbene sia importante automatizzare i *test* di integrazione il più possibile per rendere la loro esecuzione più veloce, in questo caso tali *test* sono stati eseguiti in modo manuale a causa degli scenari coinvolti nei *test*, che includono relazioni tra componenti difficili da testare in un ambiente automatizzato, come ad esempio la verifica della corretta costruzione degli elementi grafici nella pagina *web* basati sul modello logico del programma.

Di seguito vengono riportati tutti i *test* di integrazione definiti.

Codice	Descrizione	Stato
TI-1	Verifica che la pagina <i>web</i> mostri correttamente tutti i gli elementi grafici	P
TI-2	Verifica che all'avvio dell'applicazione vengano correttamente inizializzate le variabili di sessione	P
TI-3	Verifica il corretto funzionamento dei controlli di validazione del <i>login</i>	P
TI-4	Verifica che sia possibile inviare un messaggio tramite la chat	P
TI-5	Verifica che la risposta del <i>chatbot</i> venga correttamente visualizzata all'interno della chat	P
TI-6	Verifica che sia possibile comunicare correttamente con il database PostgreSQL tramite Psycopg	P
TI-7	Dopo aver caricato un documento nel <i>server</i> , verifica che i lessemi nella tabella <i>idf</i> del database siano inerenti con il testo di tale documento	P
Continua nella prossima pagina...		

Tabella 7.3 – Continuazione della tabella

Codice	Descrizione	Stato
TI-8	Dopo aver caricato un documento nel <i>server</i> , verifica che i <i>chunk</i> nella tabella <i>docs</i> del database siano inerenti con il testo di tale documento	P
TI-9	Verifica che le API di Groq permettano di comunicare correttamente con i modelli di <i>question answering</i> selezionati	P
TI-10	Verifica che le API di txtai permettano di comunicare correttamente con il modello di <i>sentence similarity</i> di HuggingFace selezionato	P
TI-11	Verifica che le API di txtai permettano di comunicare correttamente con il modello di <i>zero-shot classification</i> di HuggingFace selezionato	P
TI-12	Verifica la corretta integrazione tra txtai, NetworkX e Matplotlib per la visualizzazione grafica del <i>knowledge graph</i> e del risultato della <i>query</i> di attraversamento	P

Tabella 7.3: Tabella dei test di integrazione

7.3.3 Test di sistema

Dopo aver completato i test sulle singole unità e verificato la loro corretta integrazione, si procede con il collaudo dell'intero sistema *software*. I *test di sistema* hanno lo scopo di verificare che l'applicativo sviluppato funzioni nel modo atteso, in tutte le sue funzionalità, soddisfacendo quindi i requisiti definiti durante l'attività di analisi dei requisiti. Essi vengono eseguiti manualmente, testando l'applicazione dal punto di vista dell'utente finale.

Di seguito vengono riportati tutti i *test* di sistema definiti.

Codice	Descrizione	Stato
TS-1	Verifica che l'applicazione <i>web</i> venga correttamente caricata e visualizzata sui maggiori <i>browser</i>	P
TS-2	Verifica il corretto funzionamento del sistema di <i>login</i> alle funzionalità avanzate	P
TS-3	Verifica che l'utente possa effettuare il <i>logout</i> dalle funzionalità avanzate	P
TS-4	Verifica che l'utente possa selezionare il modello di <i>question answering</i> desiderato	P
TS-5	Verifica che l'utente possa selezionare l'algoritmo di <i>rank fusion</i> desiderato	P
TS-6	Verifica che l'utente possa scegliere di utilizzare o meno il <i>knowledge graph</i>	P
TS-7	Verifica che l'utente possa eliminare la cronologia della conversazione	P
TS-8	Verifica che l'utente possa inviare richieste al <i>chatbot</i> e ricevere delle risposte opportune	P
TS-9	Una volta effettuato l'accesso alle funzionalità avanzate, verifica che l'utente possa ricalcolare gli IDF	P
TS-10	Una volta effettuato l'accesso alle funzionalità avanzate, verifica che l'utente possa ricalcolare i vettori di <i>embedding</i>	P
TS-11	Una volta effettuato l'accesso alle funzionalità avanzate, verifica che l'utente possa rigenerare il <i>knowledge graph</i>	P
TS-12	Una volta effettuato l'accesso alle funzionalità avanzate, verifica che l'utente possa caricare un nuovo documento nel <i>server</i>	P

Continua nella prossima pagina...

Tabella 7.4 – Continuazione della tabella

Codice	Descrizione	Stato
TS-13	Una volta effettuato l'accesso alle funzionalità avanzate, verifica che l'utente possa eliminare il documento attualmente caricato	P
TS-14	Verifica che il nome del documento caricato sia visibile nella pagina web dell'applicativo	P

Tabella 7.4: Tabella dei test di sistema

7.3.4 Test di regressione

I *test di regressione* hanno lo scopo di verificare l'assenza di determinati *bug* causati dall'introduzione di nuove componenti o dalla modifica di componenti già esistenti. Anche in questo caso, i *test* vengono eseguiti manualmente, testando l'applicazione direttamente tramite i *widget* dell'interfaccia grafica che coinvolgono i moduli che hanno subito modifiche.

Di seguito vengono riportati tutti i *test* di regressione definiti.

Codice	Descrizione
TR-1	Nel caso siano state apportate modifiche alle classi <code>GUIChat</code> , <code>GUISidebar</code> o <code>GUIMainWindow</code> , verifica che l'applicazione <i>web</i> venga correttamente caricata e visualizzata sui maggiori browser
TR-2	Nel caso siano state apportate modifiche alle classi <code>KnowledgeGraph</code> , <code>RankFusion</code> , <code>RRFRankFusion</code> o <code>DBSFRankFusion</code> , verifica il corretto funzionamento del processo di <i>information retrieval</i>
Continua nella prossima pagina...	

Tabella 7.5 – Continuazione della tabella

Codice	Descrizione
TR-3	Nel caso siano state apportate modifiche alle classi <code>Document</code> , <code>PDFDocument</code> o <code>SAMDocument</code> , verifica il corretto funzionamento del caricamento dei file nel <i>server</i> e del processo di suddivisione della documentazione in <i>chunk</i>
TR-4	Nel caso siano state apportate modifiche alla classe <code>DBAccess</code> , verifica che le comunicazioni con il database avvengano correttamente
TR-5	Nel caso siano state apportate modifiche alle classi <code>QuestionAnsweringLLM</code> , <code>ZeroShotLLM</code> o <code>Chatbot</code> , verifica la correttezza delle risposte fornite dal <i>chatbot</i>
TR-6	Nel caso siano state apportate modifiche alla classe <code>SimilarityLLM</code> , verifica la correttezza dei vettori di <i>embedding</i> calcolati dal modello di <i>sentence similarity</i>

Tabella 7.5: Tabella dei test di regressione

7.4 Validazione e accettazione

Durante il periodo di stage sono stati organizzati degli incontri periodici con il tutor aziendale al fine di monitorare lo stato di avanzamento del lavoro. Queste occasioni di confronto si sono rivelate utili per garantire che il prodotto *software* fino a quel momento sviluppato fosse in linea con quanto atteso da Zucchetti S.p.A. ed eventualmente apportare le dovute modifiche o miglioramenti. Questo dialogo ha contribuito a mantenere un allineamento tra gli obiettivi prefissati e il lavoro realizzato, aumentando così la qualità del prodotto finale.

L'ultima settimana ha avuto luogo la fase di *validazione*. Essa è una tappa cruciale del ciclo di vita del *software*: viene valutato il prodotto sviluppato per determinare se soddisfa o meno le aspettative del proponente e degli utenti finali. Il progetto è quindi stato testato, in presenza del tutor aziendale, in tutte le sue funzionalità per verificare se i requisiti definiti inizialmente con l'attività di analisi dei requisiti sono

stati soddisfatti.

L'applicativo è stato valutato positivamente dal tutor aziendale, dimostrando lo sviluppo di tutte le funzionalità richieste e il raggiungimento di tutti i requisiti e gli obiettivi obbligatori prefissati.

Capitolo 8

Conclusioni

8.1 Considerazioni finali

Il periodo di tirocinio del laureando Meneghini Fabio ha avuto l'obiettivo di studiare come l'utilizzo dei LLM possa influenzare la *user experience* di programmi molto complessi e che presentano una vasta varietà di funzionalità, con lo scopo di renderli fruibili anche per gli utenti meno esperti.

Ciò è avvenuto tramite lo sviluppo di un *chatbot* alimentato da un LLM specializzato nel rispondere alle domande dell'utente (descritto alla sezione 5.2). Dal momento che il modello utilizzato non possiede conoscenze riguardanti i programmi per cui deve essere in grado di rispondere, la maggior parte del tempo è stata spesa per progettare e sviluppare un sistema di *information retrieval* (descritto nel dettaglio al capitolo 4), grazie al quale è possibile recuperare il contesto necessario per fare in modo che il *chatbot* risponda in modo efficace alla domanda, e per definire e testare una valida strategia di *chunking* del testo del *file* di documentazione caricato dall'utente (descritto alla sezione 4.4), che produca *chunk* costruiti in modo tale da mantenere una sorta di “collegamento semantico” tra due *chunk* contigui.

Grazie alla dimensione dei modelli di *question answering* selezionati, al sistema di *information retrieval* sviluppato e alla strategia utilizzata per effettuare il *chunking* della documentazione, questo *chatbot* si è rivelato essere uno strumento effettivamente utile: esso infatti è stato in grado di rispondere in modo corretto e preciso per la maggior parte delle domande con cui è stato testato. Per alcune domande,

invece, le risposte fornite sono state incomplete a causa del mancato recupero di alcuni *chunk* rilevanti con la richiesta effettuata (possibili miglioramenti alla strategia di *chunking* vengono presentati alla sezione 8.4).

Si può dunque confermare che l'applicativo sviluppato può aiutare gli utenti non esperti a migliorare la *user experience* del programma in questione, in particolare la sua usabilità.

Tuttavia, i *file* di documentazione da fornire al *chatbot* devono rispettare un determinato “formato” affinché la suddivisione in *chunk* dello stesso avvenga nel modo atteso: seguendo le linee guida descritte alla sezione 4.3 per la stesura della documentazione, viene garantita un'efficace suddivisione in *chunk* e di conseguenza un migliore ritrovamento delle sezioni rilevanti con la richiesta dell'utente, con il risultato di ottenere una risposta finale molto precisa e dettagliata.

8.2 Raggiungimento degli obiettivi

Il progetto sviluppato soddisfa tutti i requisiti stabiliti con l'attività di analisi dei requisiti e sono stati raggiunti tutti gli obiettivi obbligatori concordati con il tutor aziendale, definiti nel *Piano di lavoro* e riportati alla sezione 3.2. Gli unici due obiettivi non raggiunti sono quelli desiderabili, ovvero:

- **OD7**: integrazione con strumenti di Zucchetti S.p.A. per segnalare le azioni da svolgere direttamente nell'interfaccia grafica dell'applicazione in questione;
- **OD8**: *fine-tuning* significativo dei modelli selezionati.

L'obiettivo OD8 non è stato raggiunto per via delle notevoli risorse necessarie per effettuare il *fine-tuning* dei modelli di *question answering*, mentre per l'obiettivo OD7 non c'è stato tempo a sufficienza, dal momento che lo sviluppo del sistema di *information retrieval* ha richiesto più tempo di quanto previsto.

8.3 Conoscenze acquisite

Il tirocinio svolto ha soddisfatto appieno le aspettative: nonostante il progetto sia risultato piuttosto impegnativo, è stato molto utile per approfondire diverse tec-

nologie ed algoritmi, per acquisire competenze nell'ambito dell'*information retrieval* e per sperimentare alcune reali applicazioni di strumenti di intelligenza artificiale.

In particolare, le principali nuove conoscenze e competenze maturate durante il periodo di stage sono le seguenti:

- approfondimento del linguaggio di programmazione Python;
- approfondimento del *framework* Streamlit per lo sviluppo di applicazioni *web*;
- competenze nell'utilizzo della libreria `txtai` per l'interazione con modelli di intelligenza artificiale (in questo caso con modelli di *sentence similarity* e di *zero-shot classification*);
- conoscenze negli algoritmi di *information retrieval* (in particolare ricerca lessicale con BM25, ricerca semantica con calcolo della *cosine similarity*, ricerca ibrida tramite fusione con algoritmi RRF o DBSF e attraversamento di un *knowledge graph*) e loro implementazione per casi d'uso reali;
- conoscenze nelle strategie di *chunking* di un documento testuale e loro implementazione per casi d'uso reali;
- inserimento in un contesto aziendale.

8.4 Possibili miglioramenti futuri

Durante il periodo in azienda sono state studiate alcune soluzioni avanzate, ma che non sono state implementate nel progetto finale per questioni di tempo. I maggiori miglioramenti che possono essere apportati all'applicativo sviluppato sono i seguenti:

- è possibile integrare tecniche di *chunking* più avanzate, come ad esempio il *chunking semantico*, che si basa sull'utilizzo di tecniche di NLP avanzate per suddividere il testo secondo i suoi "confini semantici", come ad esempio i cambi di argomento, oppure il *chunking dinamico*, che regola la dimensione dei *chunk* in base al contenuto del testo, per esempio facendo terminare un *chunk* ad una

naturale pausa linguistica. Questi approcci sono più flessibili e preservano il contenuto e il contesto in modo più efficace rispetto al *chunking* a dimensione fissata, ma richiedono lo sviluppo di algoritmi più complessi e adattivi che siano in grado di analizzare e comprendere a fondo la struttura del testo;

- è possibile implementare il *knowledge graph* costruendolo a partire da alcuni marcatori inseriti manualmente nel codice dei *file* SAM della documentazione: ciò renderebbe molto più veloce la costruzione del *knowledge graph*, dato che non verrebbe coinvolta la libreria *txtai* e il modello di *sentence similarity* per calcolare tutti i vettori di *embedding* relativi ai vari *chunk* di documentazione, creando così un grafo molto preciso, dato che gli archi verrebbero definiti manualmente. Tuttavia, ciò richiederebbe una grande quantità di lavoro sul *file* di documentazione prima di poterlo caricare ed utilizzare nell'applicativo, e inoltre questa soluzione non è attuabile nel caso dei *file* PDF;
- come proposto dal tutor aziendale, invece di escludere il primo e l'ultimo decimo dei *file* PDF caricati, è possibile implementare delle verifiche più sofisticate per determinare la presenza o l'assenza di *header* e *footer* nelle pagine che compongono il documento PDF: per esempio, si potrebbe verificare la presenza di *pattern* ricorrenti nel testo della prima e dell'ultima parte, ed escluderle solo in tal caso, rendendo così l'applicazione utile anche per *file* senza *header* o *footer*.

8.5 Valutazione personale

Il progetto sviluppato nel periodo di stage rappresenta per me un completamente di quanto imparato in questi tre anni di università, affinando in particolare ciò che è stato studiato nel corso di Ingegneria del Software, usando le metodologie e le *best practices* apprese per gestire e sviluppare con successo un progetto *software* che trattava di argomenti mai affrontati prima d'ora.

Inoltre, l'attività didattica di stage è stata un'esperienza assolutamente importante: essa si è rivelata molto utile per la mia crescita personale e professionale, permettendomi di acquisire in autonomia nuove conoscenze e competenze informatiche e

di esplorare tematiche decisamente interessanti che non vengono trattate nei corsi universitari, avendo così una visione più consapevole di alcune delle tecnologie informatiche più note ed utilizzate, come ad esempio alcune tecniche per il reperimento delle informazioni da parte dei motori di ricerca o dai più noti *social network*, così come una reale applicazione di strumenti di intelligenza artificiale, come i *Large Language Model*, in un progetto *software*.

Tutto ciò ha contribuito ad ampliare e sviluppare molte competenze trasversali utili sia nell'ambito lavorativo che in quello accademico, solidificando così le basi che rappresentano il punto di partenza per il mio percorso di laurea magistrale.

Acronimi e abbreviazioni

API Application Program Interface.

BERT Bidirectional Encoder Representations from Transformers.

BM25 Best Match 25.

CI Continuous Integration.

DB Database.

DBMS Database Management System.

DBSF Distribution-Based Score Fusion.

DOM Document Object Model.

GUI Graphical User Interface.

HTML HyperText Markup Language.

IDE Integrated Development Environment.

IDF Inverse Document Frequency.

ISO International Organization for Standardization.

ITS Issue Tracking System.

LLM Large Language Model.

MD Markdown.

MVC Model-View-Controller.

MVU Model-View-Update.

NLP Natural Language Processing.

PDF Portable Document Format.

QA Question Answering.

RAG Retrieval-Augmented Generation.

RRF Reciprocal Rank Fusion.

RSF Relative Score Fusion.

SBERT Sentence Bidirectional Encoder Representations from Transformers.

SNLI Stanford Natular Language Inference.

SQL Structured Query Language.

UI User Interface.

UML Unified Modeling Language.

UX User Experience.

Glossario

Albero In teoria dei grafi, un albero è un grafo non orientato nel quale due vertici qualsiasi sono connessi da uno e un solo cammino. Un albero è dunque un grafo non orientato, connesso e aciclico. [74](#)

Application Program Interface (API) In informatica, una API è un insieme di procedure messe a disposizione dei programmatori, tipicamente raggruppate per formare un *toolkit* per uno specifico compito da svolgere. Lo scopo delle API è di fornire un’astrazione tra l’*hardware* e il programmatore o tra il *software* a basso livello e quello ad alto livello, semplificando così la programmazione. [4](#)

Ciclo di vita del software In ingegneria del *software*, l’espressione *ciclo di vita del software* si riferisce al modo in cui una metodologia di sviluppo scompone l’attività di realizzazione di prodotti *software* in sottoattività fra loro coordinate, il cui risultato finale è la realizzazione del prodotto stesso e tutta la documentazione a esso associata: fasi tipiche includono lo studio, l’analisi, la progettazione, la realizzazione, il collaudo, la messa a punto, l’installazione, la manutenzione e l’estensione, il tutto a opera di uno o più sviluppatori *software*. [72](#)

Continuous Integration (CI) In ingegneria del *software*, la *Continuous Integration* è una pratica che si applica in contesti in cui lo sviluppo del *software* avviene attraverso un sistema di controllo versione. Consiste nell’allineamento frequente (“molte volte al giorno”) dagli ambienti di lavoro degli sviluppatori verso l’ambiente condiviso (*mainline*). Il concetto è stato originariamente proposto nel contesto dell’*extreme programming* come contromisura preventiva per il problema dell’*integration hell*, ovvero le difficoltà dell’integrazione

di porzioni di *software* sviluppati in modo indipendente su lunghi periodi di tempo e che di conseguenza potrebbero essere significativamente divergenti. [2](#)

Coseno In matematica, in particolare in trigonometria, dato un triangolo rettangolo, il coseno di uno dei due angoli interni adiacenti all'ipotenusa è definito come il rapporto tra le lunghezze del cateto adiacente all'angolo e dell'ipotenusa. Più in generale, il coseno di un angolo α , espresso in gradi o radianti, è una quantità che dipende solo da α , costruita usando la circonferenza unitaria. Definendo come $\cos(x)$ il valore del coseno dell'angolo x , si ottiene la funzione coseno, una funzione trigonometrica di fondamentale importanza nell'analisi matematica. [42](#)

Database a grafo Un database a grafo, è una tipologia di database che utilizza nodi e archi per rappresentare e archiviare l'informazione. I database a grafo sono spesso più veloci di quelli relazionali nell'associazione di insiemi di dati, e mappano in maniera più diretta le strutture di applicazioni orientate agli oggetti. Scalano più facilmente a grandi quantità di dati e non richiedono le tipiche e onerose operazioni di unione (*join*). Dipendono meno da un rigido schema entità-relazione e sono molto più adeguati per gestire dati mutevoli con schemi evolutivi. Al contrario, i database relazionali sono tipicamente più veloci nell'eseguire le stesse operazioni su un grande numero di dati. [69](#)

Database Management System (DBMS) Un *database management system* (DBMS) è un sistema *software* progettato per consentire la creazione, la manipolazione e l'interrogazione di una o più basi di dati in modo corretto ed efficiente. Talvolta ci si riferisce ai DBMS utilizzando impropriamente il termine *database*, a causa dell'accoppiamento tipicamente stretto tra l'archivio dati e il *software* di gestione. [70](#)

Database relazionale Un database relazionale è un database che organizza i dati memorizzati in relazioni, ovvero tuple di valori correlati che rappresentano un'entità univoca. Tuple diverse possono essere associate individuando valori chiave in comune. [69](#)

Database vettoriale Un database vettoriale è un database in cui è possibile memorizzare vettori altamente dimensionali, oltre a dati di altri tipi. I database vettoriali implementano algoritmi che permettono di ricercare i *record* più “simili” a un dato vettore. [49](#)

Desinenza Il termine desinenza (dal latino *desinĕre*, “terminare”) è usato fin dal XVI secolo per indicare in diversi contesti linguistici la parte finale di una parola. Il significato tradizionale del termine si è assestato nell’identificare quella porzione della parola che risulta variabile nella flessione e che generalmente coincide con la sua parte finale. [39](#)

Deviazione standard La *deviazione standard* (anche conosciuta come *scarto quadratico medio*, *scarto tipo* o *scostamento quadratico medio*) è un indice di dispersione statistica, vale a dire un indicatore usato per fornire una stima sintetica della variabilità di una popolazione di dati o di una variabile casuale. È uno dei modi per esprimere la dispersione dei dati intorno a un indice di posizione, quale può essere, ad esempio, la media aritmetica o una sua stima. [52](#)

Diff In informatica, con *diff* si intende un programma che evidenzia le differenze tra due *file*. Per estensione viene così chiamato anche il *file* che contiene le differenze trovate. [74](#)

Distribuzione In statistica, in particolare nella statistica descrittiva, una distribuzione è una rappresentazione del modo in cui i diversi valori di una variabile si distribuiscono all’interno di un insieme di unità statistiche. [50](#)

Distribuzione normale In probabilità, la *distribuzione normale* (o distribuzione di Gauss) è una distribuzione di probabilità continua che è spesso usata come prima approssimazione per descrivere variabili casuali a valori reali che tendono a concentrarsi attorno a un singolo valor medio. Il grafico della funzione di densità di probabilità associata è simmetrico e ha una forma a campana, nota come “curva a campana”, “curva normale”, “curva gaussiana” o “curva degli errori”. [55](#)

Fine-tuning Nel contesto dei LLM, il *fine-tuning* è il processo che consiste nel prendere modelli pre-addestrati ed addestrarli ulteriormente su *dataset* più piccoli e specifici per affinare le loro capacità e migliorare le loro prestazioni in un particolare compito o dominio. Attraverso il *fine-tuning* si trasformano modelli ad uso generale in modelli specializzati. 11

Funzione monotona crescente In matematica, una funzione monotona crescente è una funzione che mantiene l'ordinamento tra insiemi ordinati. In altre parole, f è una funzione monotona crescente se per ogni x, y vale $x \leq y \Rightarrow f(x) \leq f(y)$. 48

Grafo Un grafo è una struttura matematica discreta che riveste interesse sia per la matematica che per un'ampia gamma di campi applicativi, come l'informatica. Esso è un insieme di elementi detti nodi o vertici che possono essere collegati fra loro da linee chiamate archi o spigoli. Più formalmente, si dice grafo una coppia ordinata $G = (V, E)$ di insiemi, con V insieme dei nodi ed E insieme degli archi, tali che gli elementi di E siano coppie di elementi di V , ovvero $E \subseteq V \times V$. 56

ISO ISO (*International Organization for Standardization*, in italiano *Organizzazione internazionale per la normazione*) è la più importante organizzazione a livello mondiale per la definizione di norme tecniche. 5

Issue Tracking System (ITS) Un *Issue Tracking System* (ITS) è un sistema *software* che gestisce e mantiene liste di *issue*, ovvero di problemi da risolvere. Gli ITS sono generalmente usati in contesti collaborativi, ma possono essere anche impiegati da singoli individui per gestire il tempo e la produttività. Questi *software*, oltre ad implementare un registro centralizzato per le *issue*, spesso includono strumenti per l'allocazione delle risorse e del tempo, per la gestione delle priorità e per la supervisione del *workflow*. 2

Kernel In informatica, il *kernel* o *nucleo* è un programma situato al centro del sistema operativo che ha generalmente un controllo completo dell'intero sistema

e fornisce un accesso sicuro e controllato dell'*hardware* ai processi in esecuzione sul computer. Dato che possono eventualmente esserne eseguiti simultaneamente più di uno, il *kernel* può avere anche la responsabilità di assegnare una porzione di *tempo-macchina* (*scheduling*) e di accesso all'*hardware* a ciascun programma (*multitasking*). 2

Large Language Model (LLM) Un LLM (*Large Language Model*, in italiano “modello linguistico di grandi dimensioni”) è un modello computazionale noto per la sua capacità di eseguire compiti generali di generazione del linguaggio e altre attività di NLP, come ad esempio la classificazione del testo. Essi acquisiscono queste abilità apprendendo relazioni statistiche da enormi quantità di testo durante un processo di addestramento intensivo. 7

Metodologia agile In ingegneria del *software*, con metodologia agile (o sviluppo agile del *software*, in inglese *agile software development*), si indica un insieme di metodi di sviluppo del *software* emersi a partire dai primi anni 2000 e fondati su un insieme di principi comuni, direttamente o indirettamente derivati dai principi del “Manifesto per lo sviluppo agile del *software*” (*Manifesto for Agile Software Development*, impropriamente chiamato anche “*Manifesto Agile*”) pubblicato nel 2001 da Kent Beck, Robert C. Martin, Martin Fowler e altri. 7

Min-max scaling Il *min-max scaling*, noto anche come normalizzazione *min-max*, è una tecnica comunemente utilizzata durante la pre-elaborazione dei dati che consiste nel normalizzare dei valori numerici appartenenti ad un dato intervallo in un intervallo specifico, tipicamente $[0, 1]$, prendendo in considerazione gli estremi dell'intervallo originale. In particolare, il valore $x \in [a, b]$ viene normalizzato tramite la formula

$$\text{minmax}_{a,b}(x) = \frac{(x - a)(d - c)}{b - a}$$

dove c, d sono gli estremi dell'intervallo $[c, d]$ su cui normalizzare x (anche chiamati valori *min-max*). 49

Mock Nella programmazione orientata agli oggetti, i *mock* sono degli oggetti che riproducono il comportamento di altri oggetti non ancora implementati. Essi sono utili per eseguire l'attività di *testing* in modo controllato e agevole. 80

Natural Language Processing (NLP) NLP (*Natural Language Processing*, in italiano “elaborazione del linguaggio naturale”) è una sottobrancha della linguistica, dell'informatica e dell'intelligenza artificiale che tratta l'interazione tra i *computer* e il linguaggio umano, in particolare sul come programmare i *computer* per elaborare e analizzare grandi quantità di dati di linguaggio naturale. Lo scopo è rendere la tecnologia in grado di “comprendere” il contenuto dei documenti e le loro sfumature contestuali, in modo tale che possa quindi estrarre con precisione informazioni e idee contenute nei documenti, nonché classificare e categorizzare i documenti stessi. 40

Open source In informatica, con l'espressione *open source* si indica un *software* distribuito, generalmente in via gratuita, sotto i termini di una licenza *open source*, che ne concede lo studio, l'utilizzo, la modifica e la redistribuzione. Questo modello si pone in contrapposizione con l'idea di *software* proprietario che permette le sopracitate concessioni solo secondo i termini dettati dal detentore del *copyright*. 2

Outlier In statistica, con il termine *outlier* si definisce, in un insieme di osservazioni, un valore anomalo e aberrante, ossia un valore chiaramente distante dalle altre osservazioni disponibili. 55

Prodotto scalare In matematica, in particolare nel calcolo vettoriale, il prodotto scalare è un'operazione binaria che associa ad ogni coppia di vettori appartenenti ad uno spazio vettoriale definito sul campo reale un elemento di quel campo. 42

Refactoring In ingegneria del *software*, con *refactoring* (o *code refactoring*) si indica una tecnica per modificare la struttura interna di porzioni di codice senza

modificarne il comportamento esterno, applicata per migliorare alcune caratteristiche non funzionali del *software* quali la leggibilità, la manutenibilità, la riusabilità, l'estensibilità del codice nonché la riduzione della sua complessità, eventualmente attraverso l'introduzione a posteriori di *design pattern*. [67](#)

Signature In informatica, la *signature* o *firma* di un metodo è costituita da un insieme di informazioni che identificano univocamente il metodo stesso fra quelli della sua classe di appartenenza. Tali informazioni includono generalmente il nome del metodo, il numero e il tipo dei suoi parametri, sebbene nella terminologia tecnica dei diversi linguaggi la firma assuma talvolta un significato più specifico, includendo informazioni aggiuntive o non includendo alcune di quelle citate (per esempio il tipo del valore restituito). [72](#)

Sistema critico Un sistema critico (in inglese *critical system*) è un generico sistema che, in caso di mancato funzionamento, può provocare danni inaccettabili. [90](#)

SNLI SNLI (*Stanford Natural Language Inference*) è un *dataset* creato dai ricercatori della *Stanford University* composto da 570 mila coppie di frasi in lingua inglese scritte da esseri umani ed etichettate manualmente come implicazione, contraddizione o neutrale. Questo *dataset* è utilizzato principalmente come *benchmark* per valutare la capacità di un modello di comprendere e ragionare sulle relazioni tra frasi in linguaggio naturale. [42](#)

Stack trace In informatica, uno *stack trace* (traducibile letteralmente dall'inglese come "traccia dello stack"), chiamato anche *stack backtrace* o *stack trace-back*, è un elenco degli *stack frame* attivi in un determinato momento durante l'esecuzione di un programma. [82](#)

Stub *Stub* o anche *metodo stub*, è una porzione di codice utilizzata per simulare il comportamento di funzionalità *software* (come una *routine* su un sistema remoto) e può fungere anche da temporaneo sostituto di codice ancora da sviluppare. Sono pertanto utili durante il *porting* di *software*, l'elaborazione distribuita e in generale durante lo sviluppo di *software* e per il *testing*. [80](#)

Tema Il termine tema, in linguistica, indica la parte di parola che resta togliendo la desinenza. [39](#)

Thread safety In programmazione, l'espressione *thread safety*, viene utilizzata, nell'ambito del *multithreading*, per indicare la caratteristica di una porzione di codice che si comporta in modo corretto nel caso di esecuzioni multiple da parte di più *thread*. In particolare è importante che i vari *thread* possano avere accesso alle stesse informazioni condivise, ma che queste siano accessibili solo da un *thread* alla volta. [71](#)

UML In ingegneria del software, UML (*Unified Modeling Language*, in italiano “linguaggio di modellizzazione unificato”) è un linguaggio di modellazione e di specifica basato sul paradigma orientato agli oggetti. Gran parte della letteratura del settore utilizza UML per descrivere soluzioni analitiche e di design in modo conciso e comprensibile per un vasto pubblico. [12](#)

Varianza In statistica e in probabilità, la varianza è una misura della dispersione dei dati, ossia che misura quanto i valori di un dato insieme si discostano dalla media dei valori dell'insieme stesso. [50](#)

Vettore In matematica e in fisica, un vettore è un oggetto geometrico che possiede una lunghezza e una direzione. Essi possono essere sommati o scalati da numeri chiamati *scalari*, formando uno *spazio vettoriale*. Un vettore viene generalmente rappresentato come una freccia che connette un punto iniziale *A* con un punto terminale *B*. [40](#)

Bibliografia

Testi

- Booch, Grady, James Rumbaugh e Ivar Jacobson. *Unified Modeling Language User Guide*. Addison-Wesley, 1999 (cit. alle pp. [12](#), [72](#)).
- Gamma, Erich et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1995 (cit. alle pp. [73](#), [75](#), [77](#), [78](#)).
- Sommerville, Ian. *Software Engineering*. Pearson Education, 2016 (cit. alle pp. [10](#), [12](#), [72](#), [81](#)).

Articoli

- Bast, Hannah, Björn Buchhold e Elmar Haussmann. «Semantic Search on Text and Knowledge Bases». In: *Foundations and Trends® in Information Retrieval* 10.2-3 (2016), pp. 119–271. ISSN: 1554-0669. DOI: [10.1561/15000000032](https://doi.org/10.1561/15000000032). URL: <http://dx.doi.org/10.1561/15000000032> (cit. a p. [40](#)).
- Devlin, Jacob et al. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». In: *CoRR* abs/1810.04805 (2018). arXiv: [1810.04805](https://arxiv.org/abs/1810.04805). URL: <http://arxiv.org/abs/1810.04805> (cit. a p. [41](#)).
- Reimers, Nils e Iryna Gurevych. «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks». In: *CoRR* abs/1908.10084 (2019). arXiv: [1908.10084](https://arxiv.org/abs/1908.10084). URL: <http://arxiv.org/abs/1908.10084> (cit. a p. [42](#)).
- Robertson, Stephen e Hugo Zaragoza. «The Probabilistic Relevance Framework: BM25 and Beyond». In: *Foundations and Trends® in Information Retrieval* 3.4

(2009), pp. 333–389. ISSN: 1554-0669. DOI: [10.1561/1500000019](https://doi.org/10.1561/1500000019). URL: <http://dx.doi.org/10.1561/1500000019> (cit. a p. 38).

Siti web

A Fresh Look at Data Preparation. URL: <https://www.iri.com/blog/business-intelligence/a-fresh-look-at-data-preparation/> (cit. a p. 35).

Adapter. URL: <https://refactoring.guru/design-patterns/adapter> (cit. a p. 75).

Advanced RAG with graph path traversal. URL: https://github.com/neuml/txtai/blob/master/examples/58_Advanced_RAG_with_graph_path_traversal.ipynb (cit. a p. 57).

Assegnazione dei punteggi nella ricerca ibrida (RRF). URL: <https://learn.microsoft.com/it-it/azure/search/hybrid-search-ranking> (cit. a p. 47).

Breaking up is hard to do: Chunking in RAG applications. URL: <https://stackoverflow.blog/2024/06/06/breaking-up-is-hard-to-do-chunking-in-rag-applications> (cit. a p. 37).

Cos'è la RAG (Retrieval-Augmented Generation)? URL: <https://aws.amazon.com/it/what-is/retrieval-augmented-generation/> (cit. a p. 63).

COSINE DISTANCE, COSINE SIMILARITY, ANGULAR COSINE DISTANCE, ANGULAR COSINE SIMILARITY. URL: <https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/cosdist.htm> (cit. a p. 42).

Dependency injection. URL: https://en.wikipedia.org/wiki/Dependency_injection (cit. a p. 79).

Distribution-Based Score Fusion (DBSF), a new approach to Vector Search Ranking. URL: <https://medium.com/plain-simple-software/distribution-based-score-fusion-dbsf-a-new-approach-to-vector-search-ranking-f87c37488b18> (cit. alle pp. 48, 52).

draw.io. URL: <https://app.diagrams.net> (cit. a p. 2).

Elm - delightful language for reliable web applications. URL: <https://elm-lang.org/> (cit. a p. 74).

- Extractive Question Answering*. URL: <https://docs.cloud.deepset.ai/docs/extractive-question-answering> (cit. a p. 60).
- Git*. URL: <https://git-scm.com> (cit. a p. 2).
- GitHub: Let's build from here*. URL: <https://github.com> (cit. a p. 2).
- google-bert/bert-base-uncased*. URL: <https://huggingface.co/google-bert/bert-base-uncased> (cit. a p. 41).
- google/gemma-2-9b*. URL: <https://huggingface.co/google/gemma-2-9b> (cit. a p. 62).
- Hybrid search with Re-ranking*. URL: <https://medium.com/@sowmiyajaganathan/hybrid-search-with-re-ranking-ff120c8a426d> (cit. a p. 47).
- LaTeX – A document preparation system*. URL: <https://www.latex-project.org> (cit. a p. 2).
- LlamaIndex, Data Framework for LLM Applications*. URL: <https://www.llamaindex.ai/> (cit. a p. 49).
- meta-llama/Llama-Guard-3-8B*. URL: <https://huggingface.co/meta-llama/Llama-Guard-3-8B> (cit. a p. 64).
- meta-llama/Meta-Llama-3-8B*. URL: <https://huggingface.co/meta-llama/Meta-Llama-3-8B> (cit. a p. 62).
- microsoft/mdeberta-v3-base*. URL: <https://huggingface.co/microsoft/mdeberta-v3-base> (cit. a p. 65).
- microsoft/mdeberta-v3-base*. URL: [MoritzLaurer/mDeBERTa-v3-base-xnli-multilingual-nli-2mil7](https://huggingface.co/MoritzLaurer/mDeBERTa-v3-base-xnli-multilingual-nli-2mil7) (cit. a p. 65).
- mistralai/Mixtral-8x7B-v0.1*. URL: <https://huggingface.co/mistralai/Mixtral-8x7B-v0.1> (cit. a p. 61).
- Mixture of Experts Explained*. URL: <https://huggingface.co/blog/moe#what-is-a-mixture-of-experts-moe> (cit. a p. 62).
- MLCommons | Better AI for Everyone*. URL: <https://mlcommons.org/> (cit. a p. 65).
- MLCommons AI Safety v0.5 Benchmark POC Taxonomy of Hazards*. URL: <https://drive.google.com/file/d/1V8KFfk8awaAXc83nZZzDV2bHgPT8jbJY/view> (cit. a p. 65).

- Model View Update - Part 1*. URL: <https://elmprogramming.com/model-view-update-part-1.html> (cit. a p. 74).
- Model-View-Update (MVU) – How Does It Work?* URL: <https://thomasbandt.com/model-view-update> (cit. a p. 74).
- Open-source vector similarity search for Postgres*. URL: <https://github.com/pgvector/pgvector> (cit. a p. 83).
- Overleaf, Online LaTeX Editor*. URL: <https://it.overleaf.com> (cit. a p. 2).
- PostgreSQL: Documentation: 17: 8.11. Text Search Types*. URL: <https://www.postgresql.org/docs/current/datatype-textsearch.html> (cit. a p. 85).
- PostgreSQL: Documentation: 17: Chapter 12. Full Text Search*. URL: <https://www.postgresql.org/docs/current/textsearch.html> (cit. a p. 85).
- Pylint*. URL: <https://pypi.org/project/pylint/> (cit. a p. 87).
- pytest documentation*. URL: <https://docs.pytest.org/en/stable/> (cit. a p. 90).
- Python and the Model-View-Update GUI Revolution*. URL: <https://xc-jp.github.io/blog-posts/2022/11/22/python-model-view-update-frameworks.html> (cit. alle pp. 74, 82).
- Retrieval Augmented Generation (RAG) Question Answering*. URL: <https://docs.cloud.deepset.ai/docs/generative-question-answering> (cit. a p. 61).
- Sentence Embedding Methods — A Survey*. URL: <https://medium.com/@busra.oguzoglu/sentence-embedding-methods-a-survey-7c62857f7b43> (cit. a p. 40).
- Strategy*. URL: <https://refactoring.guru/design-patterns/strategy> (cit. a p. 77).
- Streamlit documentation*. URL: <https://docs.streamlit.io/> (cit. a p. 82).
- Template method*. URL: <https://refactoring.guru/design-patterns/template-method> (cit. a p. 78).
- Text Splitters: Smart Text Division with Llamaindex*. URL: <https://gustavo-espindola.medium.com/%EF%B8%8F-text-splitters-smart-text-division-with-llamaindex-e4bf8d805ad0> (cit. a p. 37).
- Unlocking the Power of Hybrid Search - A Deep Dive into Weaviate's Fusion Algorithms*. URL: <https://weaviate.io/blog/hybrid-search-fusion-algorithms>.

- User experience*. URL: https://en.wikipedia.org/wiki/User_experience (cit. a p. 5).
- Virtual DOM*. URL: <https://elmprogramming.com/virtual-dom.html> (cit. a p. 74).
- Visual Studio Code - Code Editing. Redefined*. URL: <https://code.visualstudio.com> (cit. a p. 2).
- Weaviate: The AI-native database developers love*. URL: <https://weaviate.io/> (cit. a p. 49).
- What Is a Knowledge Graph?* URL: <https://www.ibm.com/topics/knowledge-graph> (cit. a p. 56).
- What is Hybrid Search?* URL: <https://medium.com/@qdrant/what-is-hybrid-search-2a0c30d0f3d2> (cit. a p. 46).
- What is Information Retrieval? | A Comprehensive Information Retrieval (IR) Guide*. URL: <https://www.elastic.co/what-is/information-retrieval> (cit. a p. 33).
- What is Question Answering?* URL: <https://huggingface.co/tasks/question-answering> (cit. a p. 59).
- What is Semantic Search? | A Comprehensive Semantic Search Guide*. URL: <https://www.elastic.co/what-is/semantic-search> (cit. a p. 40).
- What is Zero-Shot Classification?* URL: <https://huggingface.co/tasks/zero-shot-classification> (cit. a p. 65).
- Zucchetti, il software per il successo di aziende e professionisti*. URL: <https://www.zucchetti.it/it/cms/home.html> (cit. a p. 1).