

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Ingegneria «Enzo Ferrari»
Corso di Laurea Magistrale in Ingegneria Informatica (LM-32)

Progettazione e Sviluppo di un Sistema RAG On-Premise per la Gestione della Documentazione Tecnica e della Conoscenza Aziendale

Relatore:
Prof. Giovanni Simonini

Correlatore:
Dott. Marco Taccini

Candidato:
Francesco Lamberti
Matricola 196534

ANNO ACCADEMICO 2025/2026

Un osservatore consapevole è entrato nell'Occhio.
Chissà che succederà ora.

È il momento di scoprirlo?

> Sì.

Questa musica è una novità per me,
ma è un onore farne parte.

Solanum, Outer Wilds

Sommario

La conoscenza operativa di un'azienda manifatturiera è distribuita su sorgenti eterogenee e in larga parte non strutturate: la manualistica del sistema di gestione del magazzino, le procedure interne e le conversazioni in cui il personale e i tecnici dell'assistenza hanno posto e risolto problemi reali. Ritrovare un'informazione puntuale è difficile, e quando la soluzione non si trova si ricorre all'assistenza esterna del fornitore, a pagamento, anche per problemi già risolti in passato.

Questa tesi presenta la progettazione, lo sviluppo e la valutazione di un sistema di *Retrieval-Augmented Generation* (RAG) multimodale, che risponde a domande in linguaggio naturale citando i documenti da cui la risposta è tratta. Il sistema integra le tre sorgenti in un unico spazio di ricerca e tratta esplicitamente la componente visuale della documentazione: le tabelle vengono convertite in formato strutturato e le immagini — in prevalenza acquisizioni dell'interfaccia del gestionale — descritte da un modello visione-linguaggio. Alle conversazioni è dedicata una pipeline a sé, che le anonimizza e ne estrae schede di problema strutturate. Due vincoli attraversano l'intera progettazione: il sistema in esercizio opera interamente su infrastruttura interna, senza che alcun dato ne lasci il perimetro, e il recupero è condizionato dai permessi di chi pone la domanda, così che un contenuto non autorizzato non entri nel contesto del modello generativo.

Il sistema è stato valutato su 137 domande costruite a partire da documenti non riservati del corpus. La strategia di suddivisione adottata raggiunge un'accuratezza di 0,883 sulle risposte generate, contro 0,759 e 0,745 delle due alternative; le tecniche di riformulazione della query non producono miglioramenti che ne giustifichino il costo; la pipeline di estrazione dalle conversazioni produce schede corrette con accuratezza 0,975 su un campione di quaranta. Il risultato più significativo riguarda però il rapporto fra le misure: già a budget moderati le tre strategie di suddivisione recuperano i chunk giusti in misura comparabile, ma divergono nettamente sulla qualità della risposta finale. Con il modello generativo impiegato, la preparazione dei dati incide quindi più della sofisticazione della pipeline di recupero.

Parole chiave: Retrieval-Augmented Generation, Large Language Models, ricerca semantica, multimodalità, controllo degli accessi, deployment on-premise.

Indice

Sommario **ii**

1 Introduzione **1**

1.1 Contesto e motivazione 1

1.2 Obiettivi del lavoro 2

1.3 Vincolo di esecuzione locale 3

1.4 Contributi della tesi 4

1.5 Struttura della tesi 5

2 Revisione della letteratura **6**

2.1 Large Language Models 6

2.1.1 L'architettura Transformer 6

2.1.2 Encoder, decoder e varianti architetturali 7

2.1.3 Addestramento, capacità e limiti 7

2.1.4 Modelli a pesi aperti 9

2.1.5 Quantizzazione 10

2.1.6 Servizio di inferenza 10

2.2 Rappresentazioni semantiche 11

2.2.1 Dalle rappresentazioni sparse a quelle dense 11

2.2.2 Embedding di frasi e documenti 11

2.2.3 Misure di similarità 12

2.2.4 Modelli multilingue e a granularità multipla 12

2.2.5 Valutazione dei modelli di embedding 13

2.3 Metodi di recupero 13

2.3.1 Recupero lessicale 13

2.3.2 Recupero denso 14

2.3.3 Recupero ibrido 14

2.3.4 Ricerca approssimata dei vicini più prossimi . . . 15
2.3.5 Metriche di valutazione del recupero . . . 16
2.4 Riordinamento dei candidati . . . 16
2.4.1 Bi-encoder e cross-encoder . . . 16
2.4.2 Il paradigma retrieve-then-rerank . . . 17
2.5 Retrieval-Augmented Generation . . . 17
2.5.1 Il paradigma . . . 17
2.5.2 Anatomia di una pipeline . . . 18
2.5.3 Suddivisione dei documenti . . . 19
2.5.4 Trasformazione della query . . . 20
2.5.5 Architetture iterative e agentiche . . . 21
2.5.6 Valutazione dei sistemi RAG . . . 21
2.6 Database vettoriali . . . 22
2.6.1 Requisiti funzionali . . . 22
2.6.2 Filtraggio sui metadati . . . 22
2.6.3 Qdrant . . . 23
2.6.4 Alternative . . . 23
2.7 RAG multimodale . . . 24
2.7.1 Il problema . . . 24
2.7.2 Modelli visione-linguaggio . . . 24
2.7.3 Parsing di documenti complessi . . . 24
2.7.4 Strategie di indicizzazione dei contenuti visuali . . . 25
2.7.5 Le tabelle come caso particolare . . . 26
2.8 Sintesi . . . 26

3 Contesto e problema **27**
3.1 Il contesto aziendale . . . 27
3.2 L'assistenza tecnica esterna . . . 27
3.3 Le sorgenti della conoscenza . . . 28
3.3.1 Documentazione del sistema di gestione del magazzino . . . 28

3.3.2 Conversazioni con l'assistenza 29
3.3.3 Procedure e fogli di lavoro interni 30
3.3.4 Natura multimodale del corpus 30
3.4 Definizione del problema 31
3.4.1 Limiti della ricerca tradizionale 31
3.4.2 Il costo del recupero 31
3.4.3 Perché un modello linguistico da solo non è sufficiente 32
3.5 Obiettivo del sistema 33

4 Architettura del sistema **34**
4.1 Requisiti e vincoli progettuali 34
4.1.1 Requisiti funzionali 34
4.1.2 Requisiti non funzionali 35
4.1.3 Il vincolo di esecuzione locale 36
4.1.4 Vincoli sul trattamento dei dati 37
4.2 Panoramica del sistema 37
4.3 Preprocessing e ingestion 39
4.3.1 Architettura della pipeline di ingestion 39
4.3.2 Esecuzione dell'ingestion 40
4.4 Storage e indicizzazione 41
4.4.1 Il database vettoriale 41
4.4.2 Organizzazione delle collection 41
4.4.3 Schema dei metadati 42
4.4.4 Configurazione dell'indice 42
4.5 Controllo degli accessi 43
4.5.1 Modello di autorizzazione 43
4.5.2 Associazione documenti–gruppi 44
4.5.3 Filtraggio in fase di recupero 45
4.5.4 Garanzie e superficie di attacco residua 45
4.6 Backend e orchestrazione 46

4.7 Frontend e autenticazione 46
4.7.1 Interfaccia conversazionale 46
4.7.2 Gestione degli utenti e dei gruppi 47
4.8 Infrastruttura hardware 48
4.8.1 La macchina di inferenza 48
4.8.2 Servizio di inferenza 49

5 La pipeline RAG **51**
5.1 Preprocessing delle conversazioni 52
5.1.1 Struttura del dato grezzo 53
5.1.2 Anonimizzazione 53
5.1.3 Estrazione dei problemi 54
5.1.4 Deduplicazione 55
5.1.5 Approcci alternativi valutati 57
5.1.6 Gestione delle immagini nelle conversazioni 57
5.2 Preprocessing dei documenti 58
5.2.1 Estrazione della struttura 58
5.2.2 Da immagine a descrizione testuale 59
5.2.3 Da tabella a Markdown 61
5.2.4 Composizione e arricchimento 62
5.3 Suddivisione in chunk 62
5.3.1 Strategia adottata 62
5.3.2 Alternative valutate 63
5.3.3 Le schede di problema: un chunk per scheda 64
5.4 Generazione degli embedding 65
5.5 Recupero 65
5.5.1 Recupero denso 65
5.5.2 Recupero sparso 66
5.5.3 Recupero ibrido 66
5.5.4 Dimensionamento di k_1 67

5.6 Recupero filtrato per permessi . . . 67
5.6.1 Dalla sessione utente al filtro . . . 67
5.6.2 Applicazione del filtro nella ricerca . . . 67
5.6.3 Comportamento in assenza di documenti autorizzati . . . 69
5.6.4 Costo del filtraggio . . . 70
5.7 Riordinamento dei candidati . . . 70
5.7.1 Dimensionamento di k_2 . . . 71
5.8 Costruzione del contesto . . . 71
5.8.1 Selezione e ordinamento dei chunk . . . 71
5.8.2 Cronologia della conversazione . . . 72
5.8.3 Prompt di sistema . . . 72
5.9 Generazione della risposta . . . 73
5.9.1 Modello generativo . . . 73
5.9.2 Citazioni e tracciabilità . . . 73
5.10 Scelte progettuali e alternative scartate . . . 74
5.10.1 Assenza di trasformazione della query . . . 74
5.10.2 Architettura agentica . . . 74
5.10.3 Indici basati su riassunti . . . 75
5.10.4 Coerenza con il vincolo di esecuzione locale . . . 75

6 Risultati **77**

6.1 Impostazione sperimentale . . . 77
6.1.1 Il corpus di valutazione . . . 77
6.1.2 Generazione delle domande . . . 78
6.1.3 Supervisione con modello giudice . . . 78
6.1.4 Assegnazione dei chunk di riferimento . . . 79
6.1.5 Output strutturato . . . 80
6.1.6 Organizzazione dei dati . . . 80
6.1.7 Metriche . . . 80
6.1.8 Protocollo end-to-end . . . 81

6.1.9 Griglia sperimentale . . . 82
6.1.10 Riproducibilità . . . 82
6.2 Valutazione del recupero . . . 83
6.2.1 Confronto fra le configurazioni di recupero . . . 83
6.2.2 Confronto fra le strategie di suddivisione . . . 86
6.3 Valutazione end-to-end . . . 89
6.3.1 Risultati per strategia di suddivisione . . . 89
6.3.2 Discussione dei risultati end-to-end . . . 90
6.4 Valutazione delle tecniche di trasformazione della query . . . 91
6.5 Valutazione dell'estrazione dei problemi . . . 92
6.6 Controllo degli accessi . . . 93
6.7 Analisi degli errori . . . 94
6.8 Limiti della valutazione . . . 94
6.9 Discussione . . . 96

7 Conclusioni **97**
7.1 Contributi . . . 97
7.1.1 Un sistema RAG completo in un contesto reale . . . 97
7.1.2 Estrazione di conoscenza da sorgenti non strutturate . . . 98
7.1.3 Trattamento della multimodalità . . . 98
7.1.4 Controllo degli accessi integrato nel recupero . . . 99
7.1.5 Una metodologia di valutazione riutilizzabile . . . 99
7.1.6 Fattibilità dell'esecuzione interamente locale . . . 99
7.2 Limiti del lavoro . . . 100
7.3 Sviluppi futuri . . . 101
7.3.1 Misura dell'effetto sulle richieste di assistenza . . . 101
7.3.2 Citazioni a grana di pagina . . . 102
7.3.3 Autorizzazione a grana più fine . . . 102
7.3.4 Recupero multimodale nativo . . . 102
7.3.5 Recupero adattivo . . . 103

7.3.6 Apprendimento dal feedback degli utenti . . . 103
7.3.7 Estensione ad altre sorgenti aziendali . . . 103
7.4 Considerazioni finali . . . 104

A Organizzazione del codice **105**

B Prompt **106**
B.1 Prompt di sistema della pipeline . . . 106
B.2 Prompt di preprocessing . . . 107
B.2.1 Anonimizzazione delle conversazioni . . . 107
B.2.2 Estrazione e deduplicazione dei problemi . . . 108
B.2.3 Descrizione delle immagini . . . 110
B.3 Prompt delle tecniche di trasformazione della query . . . 111
B.3.1 HyDE . . . 111
B.3.2 Step-back prompting . . . 111
B.3.3 Multi-query . . . 112
B.4 Prompt della campagna di valutazione . . . 113
B.4.1 Generazione delle domande . . . 113
B.4.2 Supervisione delle domande . . . 114
B.4.3 Assegnazione dei chunk di riferimento . . . 116
B.4.4 Giudizio delle risposte . . . 117

Bibliografia **118**

Nota sull'utilizzo dell'intelligenza artificiale **122**

Elenco delle Figure

2.1 Architettura del Transformer nella formulazione originale 8
2.2 Schema concettuale di una pipeline RAG generica 19

3.1 Distribuzione settimanale dei messaggi nell'arco 2021–2025 30

4.1 Schema generale dell'architettura 38
4.2 Pipeline di ingestion 40
4.3 Schermata iniziale dell'interfaccia conversazionale 47

5.1 Percorso di una richiesta 52
5.2 Acquisizione dell'interfaccia del gestionale: il modulo di panoramica delle unità di carico, con il menu contestuale aperto 60

6.1 Richiamo cumulato per configurazione di recupero, strategia Overlap 84
6.2 Richiamo cumulato dopo il solo recupero, prima del riordinamento, per le tre strategie di suddivisione nella rispettiva configurazione migliore 86
6.3 Richiamo cumulato dopo il riordinamento, per le tre strategie di suddivisione nella rispettiva configurazione migliore 87
6.4 Accuratezza della pipeline completa al variare del budget di token e di k 89
6.5 Accuratezza della configurazione migliore per ciascuna strategia di suddivisione . . 90

Elenco delle Tabelle

4.1 Componenti del sistema e tecnologie adottate 39
4.2 Campi del payload associato a ciascun chunk in Qdrant 42
4.3 I gruppi definiti e le funzioni aziendali corrispondenti 44
4.4 Mappa delle visibilità: per ogni gruppo di utenti, i gruppi di documenti accessibili . 44
4.5 Ripartizione della memoria video fra i modelli in esercizio 49

5.1 Esito del filtro per due profili di utente 69
5.2 Costo in tempo degli stadi della pipeline di risposta 76

6.1 Griglia dei valori di k esplorati per ciascuna strategia di suddivisione 82
6.2 Richiamo delle configurazioni di recupero a 4096 e 6144 token, strategia Overlap . 85
6.3 Accuratezza della pipeline completa 90
6.4 Valutazione manuale dell'estrazione dei problemi: schede finali corrette su quaranta esaminate 93

A.1 Repository del progetto e sezioni della tesi corrispondenti 105

Elenco dei Codici

4.1 Creazione della collection con vettori densi e sparsi 43
4.2 Tabella per l'autenticazione e l'associazione ai gruppi 48
4.3 Parametri di servizio del modello generativo 50
5.1 Estratto di conversazione dopo l'anonimizzazione 54
5.2 Esempio di scheda di problema prodotta dalla pipeline 56
5.3 Configurazione della pipeline di parsing 58
5.4 Prompt per la descrizione delle immagini 59
5.5 Descrizione generata per la Figura 5.2 61
5.6 Suddivisione del documento in chunk 62
5.7 Costruzione del filtro sui permessi e recupero ibrido filtrato con fusione RRF . . . 68
B.1 Prompt di sistema per la generazione della risposta 106
B.2 Prompt di anonimizzazione 107
B.3 Estrazione dei problemi dalle conversazioni 108
B.4 Consolidamento e deduplicazione dei problemi 109
B.5 Prompt per la descrizione delle immagini (ripetuto per comodità) 110
B.6 Generazione del documento ipotetico (HyDE) 111
B.7 Generazione della domanda astratta (step-back) 111
B.8 Generazione di riformulazioni alternative (multi-query) 112
B.9 Generazione del dataset di domande 113
B.10 Supervisione del set di domande 114
B.11 Assegnazione dei chunk di riferimento 116
B.12 Giudizio della risposta generata 117

1. Introduzione

In ogni azienda che gestisca processi complessi esiste un patrimonio di conoscenza operativa che non coincide con la documentazione ufficiale. È fatto di procedure scritte una volta e aggiornate raramente, di appunti interni, di accorgimenti trasmessi a voce e di soluzioni a problemi ricorrenti che vivono soltanto nella memoria di chi li ha già affrontati. È un patrimonio prezioso e fragile: prezioso perché è ciò che permette di risolvere in dieci minuti un problema che altrimenti richiederebbe due ore, fragile perché è distribuito, non indicizzato e legato alle persone.

I modelli linguistici di grandi dimensioni hanno reso praticabile un'interazione in linguaggio naturale con questo tipo di materiale, ma non lo conoscono: la loro competenza si ferma ai dati con cui sono stati addestrati, e la documentazione proprietaria di un'azienda non ne fa parte. Il paradigma della *Retrieval-Augmented Generation* (RAG) colma esattamente questa distanza, ancorando la risposta a documenti recuperati dal corpus aziendale invece che alla sola conoscenza del modello.

Questa tesi descrive la progettazione, lo sviluppo e la valutazione di un sistema di questo tipo, realizzato e messo in esercizio in un contesto industriale reale. Il sistema integra sorgenti eterogenee e tratta esplicitamente la componente visuale dei documenti; condiziona il recupero ai permessi dell'utente che pone la domanda; e, in esercizio, funziona interamente su infrastruttura aziendale, senza che alcun dato lasci il perimetro dell'organizzazione.

1.1 Contesto e motivazione

Il lavoro è stato svolto presso il reparto logistico di un'azienda del settore ceramico, il cui magazzino è gestito da un *Warehouse Management System* (WMS) commerciale, fornito e assistito da un'azienda esterna. Attorno al gestionale si è accumulata negli anni una conoscenza operativa distribuita fra manuali, procedure interne e conversazioni con i tecnici dell'assistenza: esiste ed è raggiungibile, ma è difficile da ritrovare nel momento in cui serve, perché gli strumenti di ricerca disponibili operano sulle parole e non sul significato, e restituiscono documenti invece di risposte.

La conseguenza più concreta riguarda proprio l'assistenza, che è un servizio a pagamento. Quando la soluzione a un problema non si trova, si apre una nuova richiesta al fornitore, anche per problemi già affrontati e risolti in passato: nelle conversazioni analizzate, poco meno di quattro problemi estratti su cinque ripetono un problema che si era già presentato. Un sistema che permetta al personale di trovare da sé la risposta, citando le fonti da cui è tratta, mira a ridurre queste richieste e con esse il costo dell'assistenza, oltre al carico sulle persone più esperte, a cui oggi si rivolge chi non trova ciò che cerca. Il contesto, il problema e le sorgenti sono descritti nel Capitolo 3.

1.2 Obiettivi del lavoro

Il lavoro si è posto quattro obiettivi.

Un sistema RAG a supporto degli utenti interni.
Rispondere a domande in linguaggio naturale sulla documentazione aziendale, citando per ogni risposta i documenti da cui l'informazione proviene, in modo che l'utente possa verificarla e risolvere in autonomia i problemi che oggi richiedono l'intervento dell'assistenza esterna.

Integrazione di fonti eterogenee e multimodali.
Portare nello stesso spazio di ricerca la documentazione tecnica, le conversazioni operative e i fogli interni, trattando esplicitamente le immagini e le tabelle della documentazione invece di scartarle.

Gestione degli accessi ai dati.
Fare in modo che ogni utente possa ottenere risposte soltanto dai documenti a cui è autorizzato ad accedere, agendo in fase di recupero e non come filtro a valle della risposta.

Valutazione quantitativa delle scelte progettuali.
Costruire una procedura di misurazione riutilizzabile su un corpus aziendale privo di dataset di riferimento, e usarla per decidere fra le alternative negli stadi che più incidono sulla qualità: la suddivisione dei documenti in porzioni di testo (*chunk*), la modalità di recupero e la trasformazione della query.

1.3 Vincolo di esecuzione locale

Una condizione al contorno attraversa l'intero lavoro e ne ha orientato ogni scelta tecnologica: il sistema doveva funzionare *interamente su infrastruttura aziendale*, senza che alcun dato — documenti, domande degli utenti, risposte generate — lasciasse il perimetro dell'organizzazione. Il ricorso a servizi di inferenza in cloud per il sistema in esercizio, pur essendo la strada più rapida e con modelli di qualità superiore, era escluso in partenza. Le sole elaborazioni affidate a servizi esterni sono state svolte una tantum, durante lo sviluppo e fuori dal percorso che risponde agli utenti, e hanno riguardato conversazioni già anonimizzate in locale e documenti non riservati, come precisato nella sezione 4.1.3.

Le ragioni dell'esclusione sono tre. La prima, e dirimente, è la riservatezza: la documentazione operativa, le comunicazioni interne e i dati di magazzino costituiscono informazione industriale sensibile, e la loro trasmissione a un fornitore esterno non era ammissibile. Non si tratta di una preferenza da bilanciare con altre considerazioni, ma di un requisito posto dall'azienda come condizione preliminare del progetto: qualunque architettura che, in esercizio, avesse comportato l'uscita di quei dati dal perimetro aziendale era da scartare. La seconda è la prevedibilità dei costi e del comportamento: un sistema interrogato in modo continuativo da più utenti, con contesti lunghi, produce un profilo di spesa a consumo difficile da stimare in anticipo, mentre l'hardware dedicato rappresenta un costo in larga parte fisso e noto fin dall'inizio. La terza è l'indipendenza dal fornitore, ovvero la possibilità di continuare a operare senza essere esposti a modifiche di prezzo, di disponibilità o di versione dei modelli decise altrove.

Questo vincolo non è un dettaglio implementativo: determina quali modelli sia realistico utilizzare, quanta memoria video sia disponibile per l'inferenza, quale sia il costo di ogni passaggio aggiuntivo della pipeline. Le conseguenze progettuali sono discusse in modo sistematico nel Capitolo 4, dove il vincolo viene riportato fra i requisiti di progetto e messo in relazione con l'hardware effettivamente impiegato.

1.4 Contributi della tesi

Il lavoro porta quattro contributi.

Il primo è una **pipeline RAG multimodale funzionante in esercizio su hardware locale**. Non si tratta di un prototipo dimostrativo: il sistema è installato, dotato di autenticazione e in uso, e ogni modello impiegato in esercizio è stato scelto entro il budget di memoria di una singola scheda grafica. Le scelte progettuali che ne derivano — dalla dimensione dei chunk alla rinuncia a stadi di elaborazione aggiuntivi — sono documentate insieme alle misure che le giustificano o, dove una misura non è stata condotta, alle prove che le hanno motivate.

Il secondo è una **pipeline per l'estrazione di conoscenza strutturata da conversazioni**. Le sorgenti prive di struttura documentale sono poco trattate nella letteratura sul RAG, che presuppone quasi sempre documenti con un inizio, una fine e una gerarchia interna. Il lavoro propone e verifica un metodo per ricostruire unità informative autocontenute — schede di problema con causa, soluzione e storia delle occorrenze — a partire da anni di messaggi non strutturati, e documenta anche l'approccio alternativo inizialmente sperimentato, che non ha funzionato.

Il terzo è l'**integrazione del controllo degli accessi nella fase di recupero**. Il sistema non filtra i risultati a valle, ma condiziona la ricerca ai permessi di chi interroga, così che un contenuto non autorizzato non entri mai nel contesto del modello generativo e non possa influenzare la risposta nemmeno indirettamente.

Il quarto è una **metodologia di valutazione riutilizzabile** per un corpus aziendale privo di dataset di riferimento. La procedura costruisce domande e riferimenti a partire dai documenti stessi, con supervisione automatica e controllo a campione da parte di colleghi del reparto, e — punto che si è rivelato decisivo — separa la generazione delle domande dall'assegnazione dei chunk di riferimento, così che strategie di suddivisione diverse siano confrontabili sulle stesse domande.

1.5 Struttura della tesi

Il resto dell'elaborato è organizzato come segue.

Il Capitolo 2 presenta la revisione della letteratura: i modelli linguistici di grandi dimensioni, le rappresentazioni semantiche dense, le diverse famiglie di metodi di recupero, il riordinamento dei candidati, il paradigma RAG e i database vettoriali, con un'ultima sezione dedicata alle estensioni multimodali.

Il Capitolo 3 descrive il contesto aziendale e il problema che il sistema deve risolvere — la riduzione delle richieste all'assistenza esterna, in larga parte relative a problemi già risolti — e le sorgenti di conoscenza da cui il sistema attinge.

Il Capitolo 4 descrive l'architettura del sistema realizzato: i requisiti e i vincoli, la fase di ingestion, la memorizzazione vettoriale, il modello di controllo degli accessi, i componenti di backend e frontend e l'infrastruttura hardware su cui tutto viene eseguito.

Il Capitolo 5 entra nel dettaglio della pipeline RAG: la preparazione delle sorgenti, la suddivisione in chunk, la generazione delle rappresentazioni, il recupero filtrato per permessi, il riordinamento, la costruzione del contesto e la generazione della risposta, e si chiude con le alternative progettuali valutate e scartate.

Il Capitolo 6 riporta i risultati della campagna sperimentale: la costruzione del dataset di valutazione, il confronto fra strategie di suddivisione, modalità di recupero e tecniche di trasformazione della query, la valutazione end-to-end e quella dell'estrazione dei problemi dalle conversazioni; discute inoltre le garanzie del controllo degli accessi, la collocazione degli errori residui e i limiti della valutazione.

Il Capitolo 7 riassume i contributi, discute i limiti del lavoro e indica le direzioni di sviluppo futuro. L'Appendice A descrive l'organizzazione del codice sviluppato; l'Appendice B riporta per esteso i prompt impiegati.

2. Revisione della letteratura

Questo capitolo presenta le basi teoriche su cui poggia il sistema descritto nel resto della tesi. Il percorso parte dai modelli linguistici di grandi dimensioni, che producono la risposta finale e i cui limiti motivano il paradigma della *Retrieval-Augmented Generation*; poi tocca le rappresentazioni semantiche e i metodi di recupero, che selezionano l'informazione su cui la risposta si fonda; quindi il riordinamento dei candidati, il paradigma RAG nel suo insieme e i sistemi di memorizzazione che lo rendono praticabile. Il capitolo si chiude sulle estensioni multimodali, che riguardano la parte non testuale dei documenti.

L'esposizione resta sul piano teorico, salvo brevi anticipazioni: le scelte concrete adottate nel progetto, e le ragioni per cui sono state preferite alle alternative, sono rimandate ai Capitoli 4 e 5.

2.1 Large Language Models

2.1.1 L'architettura Transformer

L'architettura che sta alla base della quasi totalità dei modelli linguistici contemporanei è il *Transformer*, introdotto da Vaswani et al. [39]. La sua novità rispetto alle architetture ricorrenti che la precedevano è l'abbandono completo della ricorrenza in favore di un unico meccanismo, l'attenzione, che mette in relazione ciascun elemento della sequenza con tutti gli altri in un solo passaggio.

Il meccanismo opera su tre proiezioni della sequenza di input: le *query* Q, le *chiavi* K e i *valori* V. L'attenzione a prodotto scalare scalato è definita come

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \tag{2.1}$$

dove d_k è la dimensione delle chiavi: il prodotto $QK^\top$ misura la compatibilità fra ogni posizione e ogni altra, e la softmax la trasforma nei pesi con cui si combinano i valori. Due conseguenze di questa formulazione tornano nel resto della tesi. La prima è che il costo cresce con il quadrato della lunghezza della sequenza, il che spiega perché la finestra di contesto di un modello sia finita

e costosa da estendere. La seconda è che, in generazione, chiavi e valori dei token già elaborati possono essere conservati e riutilizzati invece di essere ricalcolati a ogni passo: è la cache su cui si gioca il dimensionamento della memoria in fase di servizio (sezione 2.1.6).

Nel modello completo più attenzioni indipendenti — le *teste* — operano in parallelo su proiezioni diverse dello stesso input, e l'informazione sull'ordine delle posizioni, che l'attenzione di per sé ignora, viene iniettata con apposite codifiche posizionali. Ogni blocco alterna uno strato di attenzione multi-testa e una rete *feed-forward*, entrambi avvolti da connessioni residue e normalizzazione, e il modello è la ripetizione di questo blocco per decine di livelli (Figura 2.1).

2.1.2 Encoder, decoder e varianti architetturali

La formulazione originale è di tipo encoder-decoder, pensata per la traduzione automatica. Da essa si sono staccate due famiglie con vocazioni diverse.

I modelli **encoder-only**, di cui BERT [7] è il capostipite, elaborano l'intera sequenza in modo bidirezionale: ogni posizione vede tanto il contesto che la precede quanto quello che la segue. Sono addestrati con obiettivi di tipo *masked language modelling* — prevedere token nascosti a partire dal contesto — e producono rappresentazioni particolarmente adatte a compiti di comprensione: classificazione, estrazione, e soprattutto, per quanto riguarda questa tesi, la costruzione di rappresentazioni vettoriali dense di frasi e documenti.

I modelli **decoder-only**, a cui appartengono le famiglie GPT [2] e la gran parte dei modelli generativi contemporanei, sono auto-regressivi: predicono il token successivo condizionatamente a tutti i precedenti, con un'attenzione mascherata che impedisce a ciascuna posizione di vedere il futuro. Sono i modelli che generano testo, e quindi quelli che producono la risposta in un sistema RAG.

La distinzione è rilevante perché una pipeline RAG usa entrambe le famiglie, per compiti diversi: un encoder per rappresentare e confrontare, un decoder per generare.

2.1.3 Addestramento, capacità e limiti

Un modello generativo nasce da un **pre-addestramento** auto-supervisionato su grandi quantità di testo — la predizione del token successivo — in cui acquisisce la conoscenza linguistica e fattuale

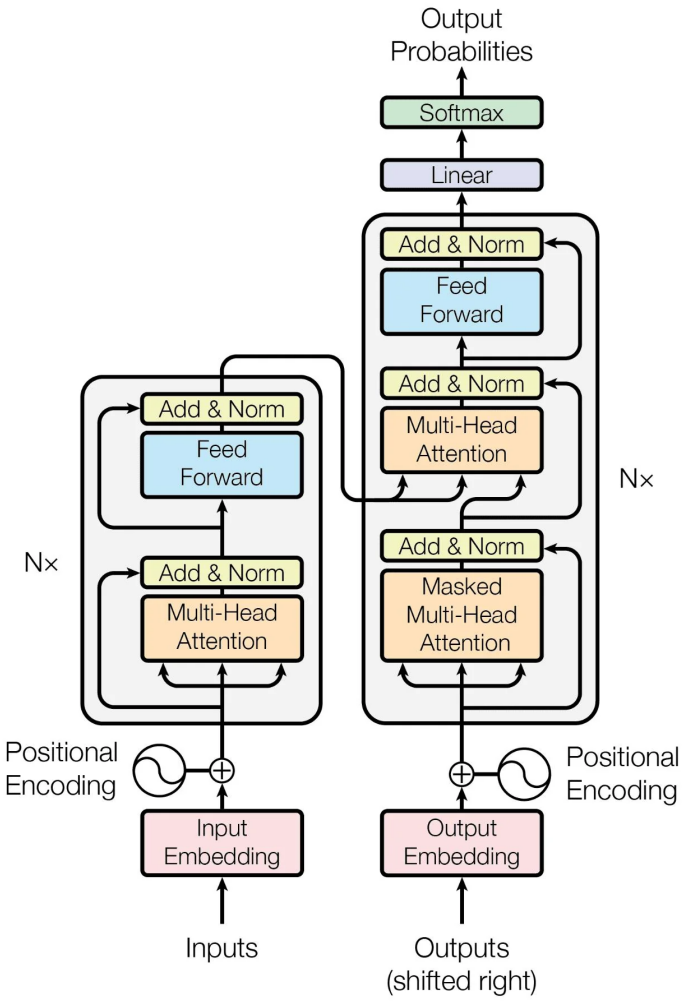


Figura 2.1: Architettura del Transformer nella formulazione originale: a sinistra la pila di encoder, a destra quella di decoder, entrambe ripetute N volte. Si riconoscono l'attenzione multi-testa, la rete feed-forward posizionale, le connessioni residue seguite da normalizzazione e le codifiche posizionali sommate agli embedding di input. Schema ridisegnato a partire da Vaswani et al. [39].

che poi esprime e, superata una certa scala, la capacità di affrontare compiti nuovi a partire da pochi esempi forniti nel prompt [2]. Una successiva fase di **allineamento**, tipicamente addestramento supervisionato su coppie istruzione-risposta seguito da ottimizzazione su preferenze umane [29], lo rende capace di seguire istruzioni: senza di essa il modello tenderebbe a continuare il testo invece di rispondere alla domanda.

Il pre-addestramento fissa però anche i limiti del modello. Quattro sono particolarmente rilevanti per questa tesi, e costituiscono la motivazione diretta del paradigma discusso nella sezione 2.5.

Conoscenza parametrica statica. Ciò che il modello sa è codificato nei suoi pesi e si riferisce al momento in cui è stato addestrato. Aggiornare quella conoscenza richiede un nuovo addestramento, che è costoso e non praticabile alla frequenza con cui la documentazione aziendale cambia. La documentazione proprietaria, per definizione, non compare affatto nei dati di pre-addestramento.

Allucinazioni. Interrogato su qualcosa che non conosce, un modello linguistico tende a produrre una risposta plausibile piuttosto che ad astenersi. Il fenomeno non è un difetto accidentale: è la conseguenza di un obiettivo di addestramento che premia la continuazione probabile del testo, non la sua verità. In un contesto aziendale, una risposta plausibile ma falsa su una procedura operativa è più dannosa di un'astensione.

Assenza di tracciabilità. La conoscenza parametrica non conserva il riferimento alla fonte da cui proviene. Un modello non può indicare da quale documento derivi un'affermazione, e l'utente non ha modo di verificarla se non consultando altrove.

Finestra di contesto finita. Il modello può elaborare solo un numero limitato di token per volta. Anche quando la finestra è ampia, il costo dell'attenzione cresce con il quadrato della lunghezza (Equazione 2.1), e la capacità del modello di usare efficacemente l'informazione al centro di contesti lunghi degrada — un fenomeno noto come *lost in the middle* [24]. Non è quindi una strategia praticabile inserire un intero corpus documentale nel prompt.

2.1.4 Modelli a pesi aperti

Accanto ai modelli proprietari accessibili solo tramite API si è affermata una famiglia di modelli i cui pesi sono pubblicamente scaricabili, cresciuta in particolare con LLaMA [38] e con le numerose famiglie che ne hanno seguito l'esempio. La disponibilità dei pesi è la condizione che rende possibile l'esecuzione su infrastruttura propria, e quindi l'intera classe di sistemi a cui questa tesi appartiene.

2.1.5 Quantizzazione

Un modello con decine di miliardi di parametri in precisione a 16 bit richiede una quantità di memoria superiore a quella di una scheda grafica di fascia consumer: un modello da 32 miliardi di parametri occupa, per i soli pesi, circa 64 GB. La *quantizzazione* riduce la precisione numerica con cui i pesi sono rappresentati — tipicamente a 4 bit — comprimendo l'occupazione di memoria dei pesi di un fattore prossimo a quattro.

Le tecniche più diffuse operano dopo l'addestramento. GPTQ [11] quantizza i pesi strato per strato minimizzando l'errore di ricostruzione sull'uscita dello strato stesso. AWQ, *Activation-aware Weight Quantization* [21], parte dall'osservazione che non tutti i pesi contribuiscono allo stesso modo: una piccola frazione dei canali, individuabile dall'ampiezza delle attivazioni corrispondenti, è sproporzionatamente importante per la qualità del modello, e va protetta dalla perdita di precisione. Invece di mantenerli in precisione più alta, AWQ scala opportunamente quei canali prima della quantizzazione, riducendone l'errore relativo, e mantiene così una qualità vicina a quella del modello originale a fronte di una riduzione sostanziale della memoria.

La quantizzazione non è quindi un ripiego, ma la tecnica che rende possibile ospitare modelli di dimensione utile su hardware accessibile.

2.1.6 Servizio di inferenza

Eseguire un modello per un singolo utente è diverso dal servirlo a più utenti concorrenti. Il collo di bottiglia della generazione auto-regressiva non è il calcolo ma la memoria: a ogni token generato il modello deve accedere alla cache delle chiavi e dei valori di tutti i token precedenti, e questa cache, che cresce con la lunghezza della sequenza e con il numero di richieste in corso, occupa memoria video e limita quante richieste possano essere servite in parallelo.

vLLM [19] affronta il problema con *PagedAttention*, un meccanismo ispirato alla memoria virtuale dei sistemi operativi: la cache viene allocata in blocchi di dimensione fissa anziché in aree contigue, riducendo drasticamente la frammentazione e permettendo di condividere blocchi fra richieste che hanno un prefisso comune. vLLM combina questo meccanismo con il *continuous batching*, una tecnica di schedulazione introdotta in precedenza che inserisce nuove richieste nel

batch man mano che altre terminano, invece di attendere il completamento dell'intero batch; ne risulta un aumento sostanziale del numero di richieste servibili in parallelo a parità di hardware.

2.2 Rappresentazioni semantiche

2.2.1 Dalle rappresentazioni sparse a quelle dense

Il modello vettoriale per il recupero dell'informazione, introdotto da Salton et al. [36], rappresenta documenti e interrogazioni come vettori in uno spazio le cui dimensioni sono i termini del vocabolario. Sono rappresentazioni *sparse*: la dimensionalità coincide con la cardinalità del vocabolario, e ciascun vettore ha valore non nullo solo sui termini effettivamente presenti.

Il limite strutturale di questa famiglia di rappresentazioni è che ogni termine è una dimensione a sé: due documenti che esprimono lo stesso concetto con parole diverse, se non condividono alcun termine, risultano ortogonali. È il problema del *vocabulary mismatch*, particolarmente severo nel question answering, dove la domanda dell'utente e il documento che la risolve spesso non usano le stesse parole.

Le rappresentazioni *dense* rovesciano l'impostazione: ogni testo è un vettore di poche centinaia o migliaia di dimensioni continue, apprese in modo che la vicinanza geometrica corrisponda alla vicinanza semantica. Il passaggio concettuale, reso popolare dagli embedding di parola statici [26] e maturato con quelli contestuali prodotti dai Transformer, è da «parole condivise» a «significato vicino».

2.2.2 Embedding di frasi e documenti

Un modello encoder produce una rappresentazione per ciascun token; per ottenere un unico vettore che rappresenti l'intero testo occorre un meccanismo di aggregazione, tipicamente la media delle rappresentazioni dei token o l'uso di un token speciale dedicato.

L'aggregazione da sola, però, non basta: le rappresentazioni prodotte da un encoder addestrato con obiettivi di masked language modelling non sono ottimizzate perché la similarità del coseno fra due vettori corrisponda alla similarità semantica fra i testi. Sentence-BERT [34] ha mostrato come colmare la lacuna con un addestramento a rete siamese su coppie di frasi, e l'approccio si

è poi consolidato nell'apprendimento *contrastivo*: il modello viene addestrato ad avvicinare nello spazio le coppie semanticamente correlate e ad allontanare le altre, usando come esempi negativi sia campioni casuali sia — più efficacemente — esempi difficili, cioè testi superficialmente simili ma non pertinenti.

2.2.3 Misure di similarità

Dati due vettori $\mathbf{u}$ e $\mathbf{v}$, le misure di prossimità più usate sono il prodotto scalare

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{d} u_i v_i, \tag{2.2}$$

e la similarità del coseno

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\|\mathbf{u}\| \, \|\mathbf{v}\|}. \tag{2.3}$$

La differenza fra le due è la sensibilità alla norma: il prodotto scalare premia i vettori di modulo maggiore, il coseno misura soltanto l'orientamento. Quando gli embedding sono normalizzati a norma unitaria, come avviene per molti modelli contemporanei, le due misure coincidono, e la similarità del coseno è legata in modo monotono decrescente alla distanza euclidea. La scelta della misura in fase di indicizzazione deve comunque essere coerente con quella usata in addestramento.

2.2.4 Modelli multilingue e a granularità multipla

Due caratteristiche dei modelli di embedding sono particolarmente rilevanti per un corpus aziendale italiano.

La prima è il **supporto multilingue**. La documentazione tecnica italiana è densa di terminologia inglese non tradotta, e un modello addestrato su una sola lingua tende a collocare i due lessici in regioni scollegate dello spazio. Un modello multilingue li allinea, permettendo di recuperare un documento che parla di *picking* a partire da una domanda che parla di prelievo.

La seconda è la **lunghezza di contesto**. Un modello che tronca l'input a poche centinaia di token impone implicitamente un limite alla dimensione dei chunk indicizzabili — un vincolo che interagisce direttamente con le strategie di suddivisione dei documenti (sezione 2.5.3).

BGE-M3 [4] è rappresentativo della generazione recente di modelli che affrontano entrambe le questioni, ed è interessante qui per una terza ragione: con un unico passaggio in avanti produce tre rappresentazioni distinte dello stesso testo. Una rappresentazione densa, ottenuta dallo stato finale del token speciale iniziale; una rappresentazione sparsa, che assegna un peso di importanza a ciascun termine del testo; e una rappresentazione multi-vettore a interazione tardiva, nello spirito di ColBERT [18]. La disponibilità simultanea delle prime due renderebbe il recupero ibrido, discusso nella sezione 2.3.3, praticamente gratuito in termini di calcolo aggiuntivo. Nel sistema realizzato, tuttavia, del modello viene usata soltanto la rappresentazione densa, servita tramite vLLM, mentre quella sparsa è calcolata a parte con BM25 (sezione 5.4).

2.2.5 Valutazione dei modelli di embedding

La qualità di un modello di embedding si misura su benchmark standardizzati. BEIR [37] raccoglie compiti di recupero eterogenei ed è costruito per valutare la capacità di generalizzare *zero-shot*, cioè su domini diversi da quelli di addestramento. MTEB [27] allarga lo spettro a otto famiglie di compiti — recupero, classificazione, clustering, similarità semantica e altri — su decine di dataset e più lingue.

Questi benchmark sono indispensabili per restringere il campo, ma il loro esito non si trasferisce automaticamente a un dominio aziendale specifico, caratterizzato da un lessico proprietario, da sigle interne e da una distribuzione delle domande che nessun benchmark generalista rappresenta. È precisamente questa distanza a motivare la costruzione di una procedura di valutazione dedicata, descritta nel Capitolo 6.

2.3 Metodi di recupero

2.3.1 Recupero lessicale

Il recupero lessicale opera sulle rappresentazioni sparse della sezione 2.2.1: assegna a ciascun documento un punteggio basato sulla sovrapposizione dei termini con l'interrogazione, pesata in modo da premiare i termini rari e da correggere l'effetto della lunghezza del documento. La funzione di ranking di riferimento è BM25 [35]:

$$\mathrm{BM25}(q,d) = \sum_{t \in q} \mathrm{IDF}(t) \cdot \frac{f(t,d) \cdot (k_1 + 1)}{f(t,d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\mathrm{avgdl}}\right)}, \tag{2.4}$$

dove $f(t,d)$ è la frequenza del termine t nel documento d, $|d|$ la lunghezza del documento, avgdl la lunghezza media nella collezione, e $\mathrm{IDF}(t)$ la frequenza inversa di documento, che pesa il termine in ragione della sua rarità. I parametri k_1 e b controllano rispettivamente la saturazione della frequenza — oltre un certo numero di occorrenze, ripetere un termine aggiunge poco — e l'intensità della normalizzazione per lunghezza.

Il recupero lessicale ha un punto di forza che nessun modello denso replica facilmente: la corrispondenza esatta. Codici, sigle, nomi di maschere e identificativi alfanumerici sono termini rari che BM25 pesa molto e che un embedding denso tende invece a diluire nel significato complessivo della frase. In un dominio come quello di questa tesi, fitto di riferimenti codificati, è una proprietà tutt'altro che marginale.

2.3.2 Recupero denso

Il recupero denso rappresenta interrogazione e documenti nello stesso spazio vettoriale e recupera per prossimità geometrica. L'architettura di riferimento è il *bi-encoder*, formalizzato per il question answering da Karpukhin et al. con Dense Passage Retrieval [17]: due encoder — distinti in DPR, con pesi condivisi in altri modelli — codificano separatamente la domanda e il passaggio, e il punteggio di rilevanza è il prodotto scalare fra i due vettori.

La proprietà cruciale di questa architettura è che le rappresentazioni dei documenti possono essere calcolate *una volta sola*, in fase di indicizzazione, e riutilizzate per ogni interrogazione futura. Al momento della ricerca resta da codificare la sola domanda e da cercare i vettori più vicini. È ciò che rende il recupero denso praticabile su collezioni di grandi dimensioni.

Il recupero denso attenua il *vocabulary mismatch*, ma paga proprio dove il lessicale è forte, sulla corrispondenza esatta di codici e sigle.

2.3.3 Recupero ibrido

I due approcci falliscono su casi diversi, il che li rende naturalmente complementari: la strategia *ibrida* esegue entrambe le ricerche e combina le graduatorie.

Il problema tecnico della combinazione è che i punteggi dei due canali non sono commensurabili: BM25 produce valori non normalizzati, su una scala che dipende dalla query e dalla collezione, la similarità del coseno valori in un intervallo fisso. Sommare direttamente i punteggi richiederebbe una normalizzazione che dipende dalla distribuzione dei punteggi stessi, ed è fragile.

La *Reciprocal Rank Fusion* [5] aggira il problema ignorando i punteggi e usando soltanto le posizioni in graduatoria. Dato un insieme di graduatorie R, il punteggio fuso del documento d è

$$\mathrm{RRF}(d) = \sum_{r \in R} \frac{1}{k + \mathrm{rank}_r(d)}, \tag{2.5}$$

dove $\mathrm{rank}_r(d)$ è la posizione di d nella graduatoria r e k una costante che attenua il divario fra i contributi delle prime posizioni (nel lavoro originale $k = 60$). L'unico parametro ha un valore di riferimento che non richiede taratura sulla collezione, e il metodo si è dimostrato sorprendentemente robusto, spesso superiore a schemi di combinazione più elaborati.

2.3.4 Ricerca approssimata dei vicini più prossimi

La ricerca esatta del vicino più prossimo richiede di confrontare l'interrogazione con ogni vettore dell'indice: un costo lineare nella dimensione della collezione, accettabile per migliaia di elementi e proibitivo per milioni. Gli indici *approssimati* rinunciano alla garanzia di trovare esattamente i k vicini più prossimi in cambio di un tempo di ricerca sublineare.

La struttura oggi prevalente è HNSW, *Hierarchical Navigable Small World* [25]. L'indice è un grafo multi-livello in cui ogni nodo è un vettore e gli archi collegano vettori vicini. I livelli superiori sono sparsi e servono a spostarsi rapidamente fra regioni distanti dello spazio; quelli inferiori sono densi e raffinano la ricerca localmente. Un'interrogazione entra dal livello più alto, scende avvicinandosi progressivamente al punto cercato, e conclude con una ricerca locale al livello base.

Due parametri governano il compromesso fra qualità e velocità: il numero di archi per nodo, fissato in costruzione, e l'ampiezza della lista di candidati mantenuta durante la visita, regolabile a tempo di interrogazione. Aumentarli migliora il richiamo dell'indice e allunga la ricerca.

2.3.5 Metriche di valutazione del recupero

Si assuma che per ciascuna interrogazione sia noto l'insieme R dei documenti rilevanti, e sia R_k l'insieme dei rilevanti recuperati fra i primi k risultati.

La **precisione** e il **richiamo** ai primi k risultati sono definiti come

$$\mathrm{P@}k = \frac{|R_k|}{k}, \qquad \mathrm{R@}k = \frac{|R_k|}{|R|}. \tag{2.6}$$

In un sistema RAG il richiamo ha un ruolo asimmetrico rispetto alla precisione: un chunk rilevante che non viene recuperato è definitivamente perduto, perché il generatore non potrà usarlo; un chunk irrilevante che viene recuperato costa spazio nel contesto e introduce rumore, che il modello può in linea di principio ignorare ma che, in pratica, tende a degradarne la risposta.

La letteratura fa uso anche di metriche sensibili alla posizione in graduatoria, come il *Mean Reciprocal Rank*, che premia la comparsa precoce del primo risultato rilevante, e il *Normalized Discounted Cumulative Gain*, che combina posizione e grado di rilevanza.

Una considerazione metodologica attraversa tutte queste metriche quando vengono applicate a un sistema RAG: sono definite rispetto a un numero di documenti recuperati, mentre il vincolo reale è lo spazio nel contesto, che si misura in token. Quando i chunk hanno dimensioni molto diverse — come accade confrontando strategie di suddivisione differenti — il confronto a parità di k è fuorviante, e va sostituito da un confronto a parità di token. È l'impostazione adottata nel Capitolo 6.

2.4 Riordinamento dei candidati

2.4.1 Bi-encoder e cross-encoder

La differenza fra bi-encoder e cross-encoder è strutturale e determina il loro ruolo nella pipeline.

Nel **bi-encoder** del recupero denso (sezione 2.3.2) domanda e documento non si incontrano mai prima del prodotto scalare finale. Il vantaggio del pre-calcolo si paga con una rappresentazione del documento che deve riassumerne il contenuto senza sapere quale domanda gli verrà posta.

In un **cross-encoder** la coppia domanda-documento viene concatenata e data in pasto al modello in un unico passaggio: ogni token della domanda può essere messo in relazione, tramite l'attenzione,

con ogni token del documento, attraverso tutti gli strati. L'interazione è completa e la stima di rilevanza sensibilmente più accurata [28]. Il prezzo è che nulla può essere pre-calcolato: ogni coppia richiede un passaggio in avanti, e il costo cresce linearmente con il numero di documenti da valutare, il che rende impraticabile applicarlo all'intero indice.

Fra i due estremi si collocano le architetture a *interazione tardiva* come ColBERT [18], che conservano una rappresentazione per token e calcolano l'interazione al momento della ricerca: più espressive di un bi-encoder, più economiche di un cross-encoder, ma con un costo di memoria dell'indice molto maggiore.

2.4.2 Il paradigma retrieve-then-rerank

La combinazione naturale delle due architetture è una pipeline a due stadi. Un primo stadio economico — bi-encoder, BM25 o la loro fusione — recupera dall'intero indice un insieme ampio di k_1 candidati, privilegiando il richiamo. Un secondo stadio accurato — il cross-encoder — li riordina e ne seleziona i $k_2 < k_1$ migliori, privilegiando la precisione.

Il dimensionamento dei due parametri risponde a criteri diversi. k_1 va scelto abbastanza grande da contenere con alta probabilità i chunk rilevanti, perché ciò che non entra qui è perduto; il suo limite è il tempo di riordinamento, lineare in k_1. k_2 è invece vincolato dallo spazio disponibile nel contesto del generatore, e va scelto sapendo che oltre una certa soglia il contesto aggiuntivo introduce più rumore che informazione.

2.5 Retrieval-Augmented Generation

2.5.1 Il paradigma

L'idea alla base del RAG, formalizzata da Lewis et al. [20] e anticipata da lavori come REALM [14] e oggetto di un'ampia letteratura successiva [13], è di affiancare alla *memoria parametrica* del modello — la conoscenza codificata nei pesi — una *memoria non parametrica* costituita da un corpus documentale consultabile al momento della richiesta.

Formalmente, nella variante che gli autori chiamano *RAG-Sequence*, la generazione della risposta y data la domanda x viene marginalizzata sui documenti z recuperati da un retriever $p_\eta(z \mid x)$:

$$p(y \mid x) \approx \sum_{z \in \mathcal{Z}(x)} p_\eta(z \mid x)\; p_\theta(y \mid x, z), \tag{2.7}$$

dove $\mathcal{Z}(x)$ è l'insieme dei documenti recuperati e p_θ il modello generativo. Nelle implementazioni pratiche contemporanee la marginalizzazione è sostituita da una semplice concatenazione: i documenti recuperati vengono inseriti nel prompt e il modello genera condizionatamente ad essi.

Questa architettura risponde puntualmente ai quattro limiti elencati nella sezione 2.1.3. La conoscenza diventa aggiornabile, perché per modificarla basta reindicizzare i documenti senza toccare il modello. Le allucinazioni si riducono, perché la risposta è ancorata a materiale fornito e il modello può essere istruito a non andare oltre. La tracciabilità diventa possibile, perché la risposta può essere accompagnata dall'indicazione delle fonti da cui è tratta. E il problema della finestra di contesto viene aggirato, perché non si inserisce l'intero corpus ma solo la porzione pertinente.

2.5.2 Anatomia di una pipeline

Una pipeline RAG si articola in due percorsi che si incontrano nell'indice. Il percorso *offline* comprende l'acquisizione e il parsing dei documenti, la loro suddivisione in chunk, il calcolo delle rappresentazioni e l'indicizzazione. Il percorso *online* comprende l'eventuale trasformazione della domanda, il recupero, il riordinamento, la costruzione del contesto e la generazione della risposta.

I due percorsi sono riassunti nella Figura 2.2: il primo popola l'indice a partire dai documenti, il secondo lo interroga a partire dalla domanda dell'utente e ne porta l'esito al modello generativo.

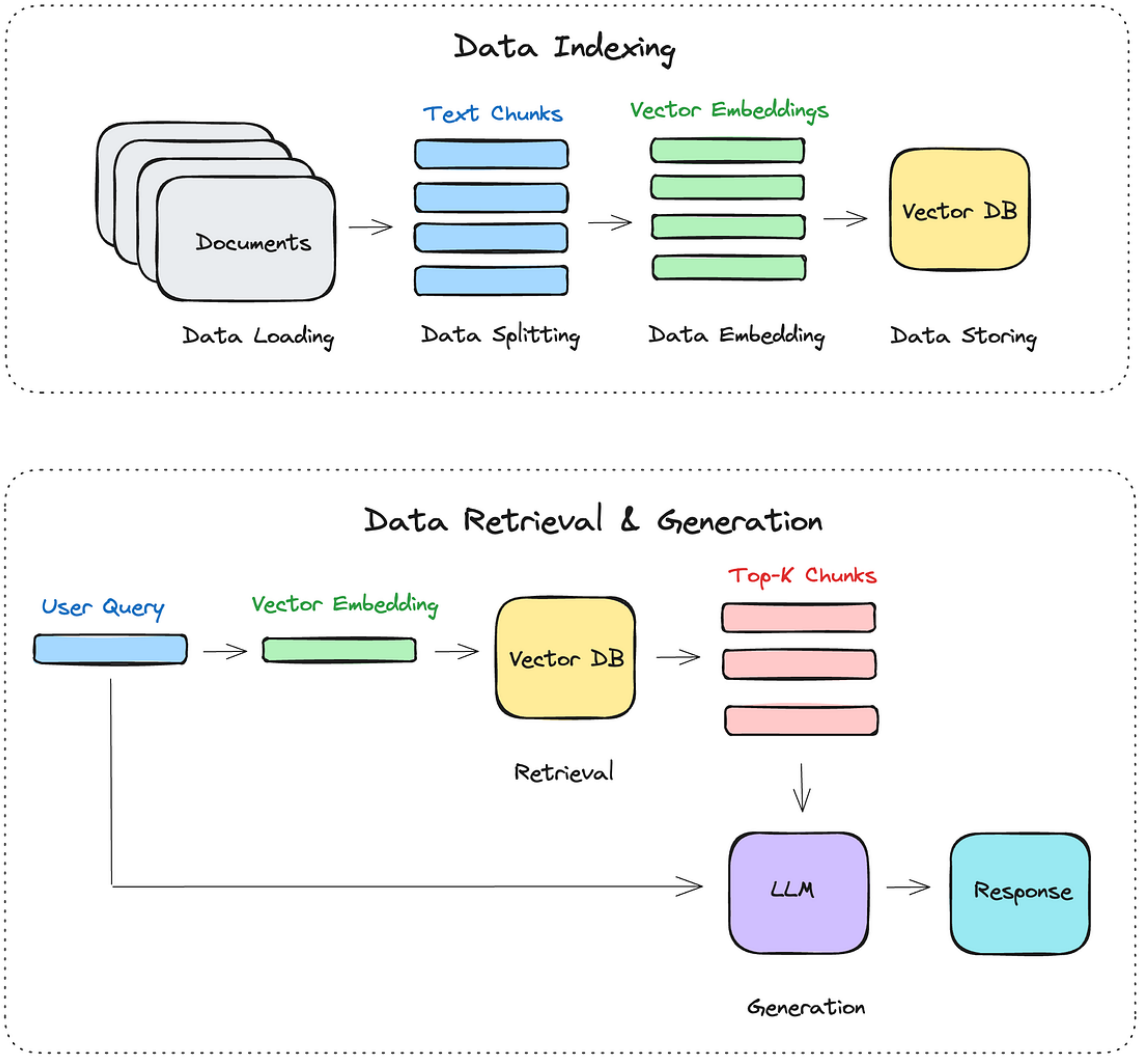


Figura 2.2: Schema concettuale di una pipeline RAG generica. In alto il percorso offline, che trasforma i documenti in chunk, li converte in rappresentazioni vettoriali e li deposita nel database vettoriale; in basso il percorso online, che rappresenta la domanda nello stesso spazio, recupera i top-k chunk più pertinenti e li passa al modello generativo insieme alla domanda originale.

2.5.3 Suddivisione dei documenti

La suddivisione in chunk (*chunking*) è probabilmente la scelta più sottovalutata dell'intera pipeline, e una di quelle che incidono di più sul risultato finale. Il chunk è infatti al tempo stesso l'unità di *indicizzazione* — ciò che viene rappresentato da un vettore — e l'unità di *contesto* — ciò che viene passato al generatore. Le due funzioni tirano in direzioni opposte.

Chunk **piccoli** producono rappresentazioni più focalizzate, perché un vettore che riassume poche righe ne coglie il contenuto meglio di un vettore che ne riassume molte, e permettono un recupero più selettivo. Ma un chunk piccolo, recuperato isolatamente, può risultare inutilizzabile: manca il contesto che lo rende interpretabile, e la risposta a una domanda operativa raramente sta in poche righe.

Chunk **grandi** conservano il contesto e sono auto-sufficienti, ma la loro rappresentazione vettoriale è una media di contenuti eterogenei, e occupano molto spazio nella finestra del generatore.

Le strategie principali sono:

Dimensione fissa con sovrapposizione.
Il testo viene tagliato a intervalli regolari, misurati in token, con una porzione di sovrapposizione fra chunk consecutivi. La sovrapposizione mitiga il rischio che un'informazione a cavallo di un confine venga spezzata. È la strategia più semplice e, proprio per questo, un termine di confronto spesso competitivo.

Suddivisione strutturale.
I confini seguono la struttura logica del documento riconosciuta in fase di parsing: sezioni, sotto-sezioni, paragrafi. I chunk sono semanticamente coerenti per costruzione, ma di dimensione molto variabile, il che rende necessario un vincolo di dimensione massima.

Suddivisione gerarchica.
Il documento viene suddiviso a più livelli di granularità contemporaneamente. Il recupero opera sul livello fine, più selettivo, mentre al generatore viene restituito il livello grossolano che contiene i chunk recuperati, così da conservarne il contesto.

Suddivisione semantica.
I confini sono posti dove la similarità fra frasi consecutive cala al di sotto di una soglia, nell'ipotesi che un cambio di argomento si manifesti come una discontinuità nello spazio degli embedding.

2.5.4 Trasformazione della query

Un insieme di tecniche interviene sulla domanda prima del recupero, nell'ipotesi che la formulazione dell'utente non sia quella ottimale per interrogare l'indice.

La **multi-query** genera diverse riformulazioni della stessa domanda e unisce i risultati delle rispettive ricerche, nella speranza che almeno una delle formulazioni intercetti il documento giusto.

HyDE, *Hypothetical Document Embeddings* [12], inverte l'impostazione: invece di migliorare la domanda, chiede al modello di *inventare* una risposta plausibile, e usa l'embedding di questo documento ipotetico per la ricerca. L'intuizione è che un documento fittizio somigli, nello spazio degli embedding, ai documenti reali che contengono la risposta più di quanto non faccia la domanda, che è un oggetto testuale di natura diversa.

Lo **step-back prompting** [41] astrae la domanda verso il concetto o il processo generale che la sottende, recuperando documentazione di contesto più ampia.

Un punto merita di essere sottolineato, perché torna nel Capitolo 6: tutte queste tecniche servono a colmare un disallineamento fra il linguaggio della domanda e quello del documento. Il beneficio presuppone però che il modello che riformula conosca il dominio: dove il lessico è specialistico e proprietario, l'espansione rischia di introdurre termini plausibili ma assenti dai documenti, mentre il costo — una chiamata aggiuntiva al modello generativo prima di ogni ricerca — resta intero.

2.5.5 Architetture iterative e agentiche

Una linea di ricerca più recente rende il recupero un processo iterativo governato dal modello stesso. Self-RAG [1] addestra il modello a decidere quando recuperare e a criticare il materiale recuperato prima di usarlo; FLARE [15] attiva il recupero dinamicamente durante la generazione, quando il modello sta per produrre contenuto di cui è poco sicuro.

Queste architetture migliorano i casi in cui una singola ricerca non basta, ma richiedono al modello generativo capacità di pianificazione e di autovalutazione che dipendono in misura rilevante dalla sua scala e, plausibilmente, dal grado di quantizzazione — una considerazione che ha un peso concreto quando il modello deve essere eseguito su hardware limitato.

2.5.6 Valutazione dei sistemi RAG

Valutare un sistema RAG significa valutare due cose distinte: la qualità del recupero e la qualità della risposta.

Per la seconda, la difficoltà è che non esiste una risposta corretta unica: la stessa informazione può essere espressa in molti modi, e le metriche di sovrapposizione testuale mutuate dalla traduzione e dalla sintesi automatica, come BLEU e ROUGE, correlano male con il giudizio umano. L'approccio che si è affermato è l'uso di un modello linguistico come valutatore (*LLM-as-a-judge*) [42], a cui si sottopongono la domanda, la risposta e un riferimento, chiedendo un giudizio motivato.

Framework come RAGAs [9] sistematizzano l'approccio articolandolo su più dimensioni: la *fedeltà*, cioè il grado in cui la risposta è sostenuta dal contesto recuperato; la *rilevanza della risposta* rispetto alla domanda; e la *rilevanza del contesto* rispetto a ciò che serviva per rispondere.

L'uso di un modello come giudice presenta bias documentati: preferenza per le risposte più lunghe, per quelle stilisticamente simili a quelle che il giudice stesso produrrebbe, e sensibilità all'ordine in cui le opzioni vengono presentate [42]. Le contromisure usuali consistono nel separare i ruoli — il modello che genera il materiale non è quello che lo valuta — nell'ancorare il giudizio a un riferimento costruito in precedenza anziché a un'opinione formulata al momento, e nel validare l'accordo con il giudizio umano su un campione.

2.6 Database vettoriali

2.6.1 Requisiti funzionali

Un database vettoriale deve fare molto più che cercare il vicino più prossimo. I requisiti che contano in un sistema in esercizio sono la persistenza dell'indice, l'aggiornamento incrementale dell'indice, senza ricostruirlo da zero, la memorizzazione di metadati associati a ciascun vettore, la possibilità di filtrare la ricerca su quei metadati, e la capacità di ospitare nella stessa struttura rappresentazioni di tipo diverso.

2.6.2 Filtraggio sui metadati

Il filtraggio merita una trattazione a sé, perché è il meccanismo su cui poggia il controllo degli accessi descritto nella sezione 4.5, e perché il modo in cui viene realizzato ha conseguenze sostanziali.

Nel **post-filtering** la ricerca viene eseguita sull'intero indice e il filtro è applicato ai risultati. L'implementazione è banale, ma il comportamento è insidioso: se i primi k risultati appartengono

tutti a documenti esclusi dal filtro, il sistema restituisce un insieme vuoto anche quando esistono documenti pertinenti e ammissibili più in basso nella graduatoria. Per ottenere k risultati ammissibili occorre recuperarne molti di più e sperare che bastino.

Nel **pre-filtering** la condizione è parte dell'interrogazione: la visita dell'indice considera soltanto i punti che la soddisfano. Il risultato è corretto per costruzione — si ottengono i k migliori *fra quelli ammissibili*, a meno dell'approssimazione dell'indice — ma l'implementazione è più delicata, perché un grafo HNSW costruito su tutti i punti può risultare poco navigabile quando gran parte dei nodi è esclusa: il cammino fra due punti ammissibili può passare per nodi filtrati, e la visita degenera. I motori che supportano il pre-filtering in modo efficiente adattano la strategia di visita alla selettività del filtro.

La distinzione ha anche una valenza di sicurezza: solo il pre-filtering garantisce che un contenuto non autorizzato non entri mai nella pipeline di recupero, nemmeno come candidato intermedio da scartare in seguito.

2.6.3 Qdrant

Qdrant [31] organizza i dati in *collection*, insiemi di *point* ciascuno composto da un identificativo, uno o più vettori e un *payload* di metadati in formato strutturato. La ricerca per similarità si basa su un indice HNSW, e le condizioni di filtro sul payload vengono valutate durante la visita del grafo.

Due caratteristiche sono particolarmente rilevanti per questa tesi. La prima è il supporto nativo a vettori densi e sparsi *nella stessa collection*: uno stesso point può portare entrambe le rappresentazioni, il che permette di realizzare il recupero ibrido senza mantenere due collection o due sistemi separati e, soprattutto, garantisce che un filtro si applichi in modo identico a entrambi i canali. La seconda è che può essere eseguito interamente in locale, senza alcuna dipendenza da servizi esterni.

2.6.4 Alternative

Fra le alternative, FAISS [16] è una libreria di ricerca per similarità molto efficiente ma priva di gestione dei metadati e di un'interfaccia di servizio: è un componente, non un database. Milvus e Weaviate offrono funzionalità paragonabili a quelle di Qdrant, mentre estensioni come `pgvector`

portano la ricerca vettoriale dentro PostgreSQL, combinandola con le condizioni relazionali al prezzo di prestazioni inferiori sui volumi elevati. In questo lavoro il criterio discriminante è stato il combinarsi di esecuzione interamente locale, supporto nativo all'ibrido e filtraggio efficiente sui metadati.

2.7 RAG multimodale

2.7.1 Il problema

La documentazione tecnica non è soltanto testo. Una procedura operativa è fatta di paragrafi, ma anche di schermate del gestionale che mostrano quale campo compilare, di diagrammi di flusso che descrivono la sequenza delle operazioni, di tabelle di codici. Un sistema che estragga il solo flusso testuale e scarti il resto rinuncia a una parte dell'informazione, e non è una parte ridondante: spesso il contenuto visuale è l'unico posto in cui un dettaglio operativo compare.

2.7.2 Modelli visione-linguaggio

I primi modelli a collegare immagini e testo si limitavano ad allinearne le rappresentazioni. CLIP [33] ha mostrato come ottenere tale allineamento con un addestramento contrastivo su coppie immagine-didascalia raccolte dal web, producendo uno spazio condiviso in cui un'immagine e il testo che la descrive sono vicini.

Una generazione successiva di modelli ha collegato un encoder visivo a un modello linguistico generativo, ottenendo sistemi capaci non solo di associare ma di *descrivere*: LLaVA [22] è rappresentativo di questa famiglia, addestrata con *visual instruction tuning* a seguire istruzioni su contenuti visivi. Sono i modelli che permettono di chiedere a un sistema di produrre una descrizione testuale di uno schema o di leggere il contenuto di una schermata. In questa tesi l'espressione *modello visione-linguaggio* (*Vision-Language Model*, VLM) indica questa seconda famiglia.

2.7.3 Parsing di documenti complessi

Prima ancora di descrivere le immagini, occorre individuarle. L'analisi del layout di un documento comprende il riconoscimento delle regioni (testo, titoli, tabelle, figure), la ricostruzione dell'ordine

di lettura e l'estrazione della struttura delle tabelle. Modelli come LayoutLM [40] hanno mostrato il vantaggio di trattare congiuntamente testo, posizione e aspetto visivo.

Docling [6] è una libreria che integra questi componenti in una pipeline: riconosce la struttura del documento, estrae le tabelle ricostruendone le celle, applica OCR alle porzioni che ne hanno bisogno e può invocare un modello visione-linguaggio per descrivere le immagini, producendo in uscita una rappresentazione strutturata del documento.

2.7.4 Strategie di indicizzazione dei contenuti visuali

Le strategie per rendere recuperabile il contenuto visuale appartengono a tre famiglie.

Conversione in testo.
Ogni immagine viene descritta da un modello visione-linguaggio, e la descrizione prende il posto dell'immagine nel documento. Il vantaggio è che l'intero corpus resta in un unico spazio di rappresentazione, con un solo modello di embedding, un solo indice e un comportamento uniforme di recupero ibrido e filtri. Lo svantaggio è che la descrizione è una compressione con perdita, e ciò che il modello non ha ritenuto rilevante non è più recuperabile.

Embedding multimodali.
Immagini e testo vengono rappresentati in uno spazio condiviso, e la ricerca opera direttamente su entrambi. Si evita la perdita dovuta alla descrizione testuale, ma servono un embedder e un modello generativo multimodali, in genere più pesanti a parità di resa sul testo.

Recupero sulla pagina renderizzata.
L'approccio più recente, rappresentato da ColPali [10], rinuncia del tutto all'estrazione: ogni pagina viene trattata come immagine e indicizzata con rappresentazioni multi-vettore a interazione tardiva ottenute adattando un modello visione-linguaggio. Elimina l'intera catena di parsing e ottiene risultati notevoli su documenti a layout complesso, al prezzo di un indice molto più pesante e di un modello visivo sempre attivo.

La scelta fra le tre non dipende solo dalla qualità attesa, ma dai vincoli di memoria e dalla necessità di mantenere uniformi funzionalità come il filtraggio sui metadati.

2.7.5 Le tabelle come caso particolare

Le tabelle si collocano scomodamente fra testo e immagine. Linearizzate come testo corrente perdono la corrispondenza fra intestazioni e valori, e una cella diventa un numero senza significato. Trattate come immagini, diventano invisibili al recupero lessicale — proprio quello che servirebbe per i codici che spesso contengono. La soluzione pratica prevalente è la conversione in un formato testuale che preserva la struttura, come Markdown o HTML: il contenuto resta ricercabile lessicalmente e la relazione riga-colonna resta leggibile dal modello generativo.

2.8 Sintesi

Gli elementi presentati in questo capitolo costituiscono il repertorio da cui il sistema descritto nel resto della tesi attinge: un modello generativo a pesi aperti, quantizzato e servito localmente; un modello di embedding denso multilingue, affiancato da una rappresentazione lessicale sparsa; un database vettoriale con filtraggio sui metadati; un cross-encoder per il riordinamento; una pipeline di parsing multimodale che riconduce immagini e tabelle a testo strutturato.

Due questioni restano però trattate solo marginalmente nella letteratura considerata, e sono le stesse su cui questo lavoro ha dovuto trovare una risposta propria.

La prima è l'integrazione del **controllo degli accessi** nella pipeline di recupero. La letteratura sul RAG assume quasi sempre un corpus uniformemente accessibile; il filtraggio sui metadati è trattato come funzionalità di prodotto dei database vettoriali, non come requisito architetturale di un sistema in cui utenti diversi devono ottenere risposte diverse dalla stessa domanda.

La seconda è il trattamento di **sorgenti prive di struttura documentale**. Le tecniche di suddivisione descritte sopra presuppongono un documento con un inizio, una fine e una struttura interna. Un flusso di conversazioni non ha nulla di tutto ciò: le unità informative non hanno confini espliciti e vanno ricostruite prima ancora di poter essere suddivise.

Su entrambe le questioni i Capitoli 4 e 5 propongono una soluzione operativa. Il Capitolo 6 misura la qualità dell'estrazione dalle conversazioni, mentre per il controllo degli accessi argomenta perché la correttezza discenda dalla costruzione stessa del filtro.

3. Contesto e problema

Prima di descrivere il sistema, questo capitolo ne fissa il punto di partenza: il contesto aziendale in cui è stato realizzato, il problema concreto che deve risolvere e le sorgenti da cui attinge. È il problema, più che la tecnologia, a giustificare le scelte dei capitoli successivi: che cosa includere nel corpus, quanto debba essere affidabile una risposta, perché ogni risposta debba citare le proprie fonti.

3.1 Il contesto aziendale

Il lavoro descritto in questa tesi è stato svolto presso il reparto logistico di un'azienda del settore ceramico. Il magazzino è gestito da un *Warehouse Management System* (WMS) commerciale, attorno al quale si è stratificata negli anni una documentazione fatta di manuali, descrizioni delle modifiche introdotte e appunti operativi.

L'impianto comprende due magazzini automatici verticali, in cui l'attività prevalente è il *picking*, cioè il prelievo dei singoli articoli necessari a comporre un ordine: è uno dei punti in cui la corretta esecuzione delle procedure incide più direttamente sulla continuità operativa.

Il funzionamento quotidiano del magazzino poggia quindi su un sistema gestionale complesso e su una quantità di conoscenza pratica che si è accumulata attorno ad esso negli anni: come si esegue una procedura, perché un pallet resta fermo in un punto dell'impianto, quale accorgimento ha risolto un problema ricorrente. Parte di questa conoscenza è scritta nei manuali del gestionale; una parte altrettanto rilevante non lo è, e vive nella memoria delle persone che hanno seguito il sistema nel tempo o nelle conversazioni in cui i problemi sono stati affrontati.

3.2 L'assistenza tecnica esterna

Il WMS è un prodotto commerciale, fornito e assistito da un'azienda esterna. Quando in magazzino si presenta un problema che il personale non sa risolvere — una stazione che resta in attesa, un pallet fermo, un errore del gestionale di cui non si conosce la causa — la via d'uscita è rivolgersi

all'assistenza tecnica del fornitore, con una segnalazione nel gruppo di messaggistica condiviso con i suoi tecnici e con la successiva apertura di un ticket.

L'assistenza è un servizio a pagamento. Ogni richiesta ha un costo diretto per l'azienda, a cui si aggiunge quello indiretto del tempo in cui l'operazione resta ferma in attesa della risposta.

Molte di queste richieste, però, non riguardano problemi nuovi. Lo mostra l'analisi delle conversazioni con l'assistenza descritta nella sezione 5.1: dei 482 problemi estratti dai messaggi dal 2021 al 2025, una volta accorpati i duplicati restano 109 problemi distinti. Poco meno di quattro problemi estratti su cinque ripetevano quindi un problema che si era già presentato in passato, e la cui soluzione era in genere già stata trovata e scritta — nello storico delle conversazioni o nella documentazione — ma che nel momento del bisogno non si riusciva a ritrovare. In questi casi l'azienda paga di nuovo per una conoscenza che possiede già.

3.3 Le sorgenti della conoscenza

La conoscenza che servirebbe per risolvere in autonomia buona parte di questi problemi esiste, ed è distribuita su tre sorgenti, differenti per formato, struttura e modo in cui l'informazione vi è depositata. Sono le sorgenti che costituiscono il corpus del sistema.

3.3.1 Documentazione del sistema di gestione del magazzino

La sorgente principale è la documentazione del WMS in uso in azienda. Comprende i manuali sul funzionamento del sistema e una serie di documenti che descrivono le migliorie introdotte negli anni, in particolare sui processi di *picking* e sui magazzini automatici verticali.

Si tratta di documentazione tecnica ricca di elementi non testuali: schemi, mappe concettuali, acquisizioni dell'interfaccia del gestionale e tabelle di codici, la cui rilevanza è discussa nella sezione 3.3.4.

Il corpus documentale comprende oltre un centinaio di documenti, di dimensioni molto eterogenee: si va da procedure operative di poche pagine — le istruzioni per una singola attività — fino a estratti di manuale di una cinquantina di pagine che descrivono interi moduli del gestionale. Questa eterogeneità pesa sulla scelta della strategia di suddivisione, discussa nella sezione 5.3.

Una caratteristica del corpus ha semplificato sensibilmente il lavoro: tutti i documenti sono nativamente digitali, con testo selezionabile. Non è stato quindi necessario affrontare il caso, ben più problematico, della documentazione scansionata, in cui la qualità del riconoscimento ottico dei caratteri diventa il fattore limitante dell'intera pipeline.

3.3.2 Conversazioni con l'assistenza

La seconda sorgente è costituita dalle conversazioni WhatsApp fra il personale e i tecnici dell'assistenza, le stesse descritte nella sezione 3.2, in cui nel corso degli anni sono state poste domande e ricevute risposte su problemi reali del magazzino. È una sorgente di conoscenza preziosa e al tempo stesso difficile da trattare: contiene la soluzione a problemi ricorrenti che non è documentata da nessun'altra parte — cioè proprio la conoscenza che, se ritrovata, eviterebbe una nuova richiesta di assistenza — ma la contiene in forma frammentata, dispersa fra messaggi di servizio, saluti e scambi non pertinenti.

A differenza di un documento, una conversazione non ha confini espliciti: un problema può iniziare in un messaggio, essere ripreso ore dopo e risolversi in un messaggio che non lo menziona più. Per questo la sorgente non viene indicizzata così com'è: una pipeline dedicata, descritta nella sezione 5.1, ne ricava 109 schede di problema autocontenute, che sono ciò che entra nell'indice.

Le conversazioni coprono l'arco 2021–2025. Il volume non è distribuito in modo uniforme: la Figura 3.1 mostra l'andamento settimanale, con una fase iniziale molto intensa — corrispondente all'avviamento del sistema — seguita da un regime più irregolare, punteggiato da picchi che coincidono con periodi di criticità operativa.

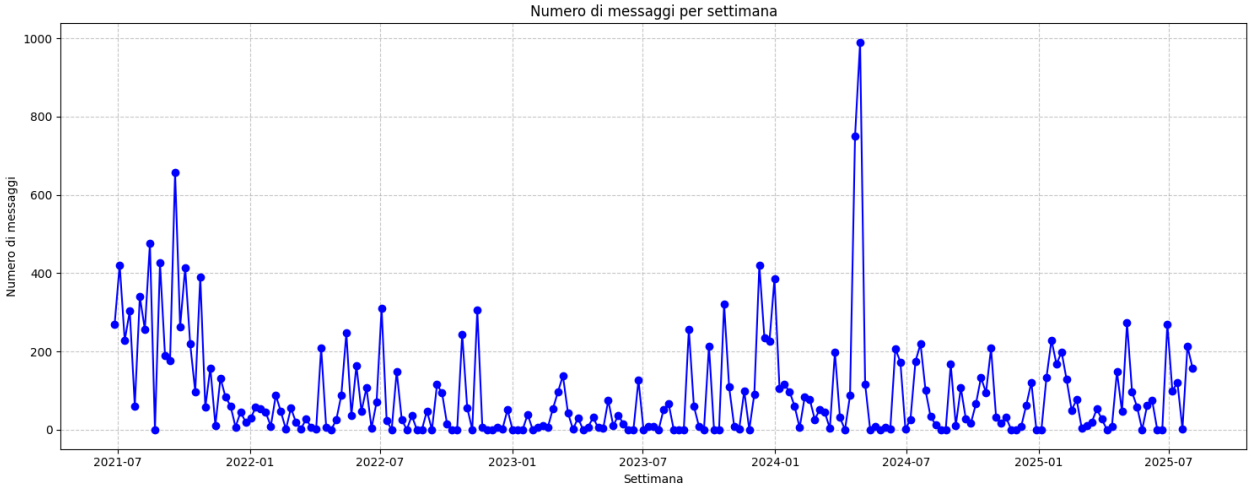


Figura 3.1: Distribuzione settimanale dei messaggi nell'arco 2021–2025. L'intensità iniziale corrisponde alla fase di avviamento; i picchi successivi a periodi di criticità operativa.

3.3.3 Procedure e fogli di lavoro interni

La terza sorgente è costituita dai fogli di lavoro interni. Si tratta in larga parte di *procedure*: documenti che descrivono passo per passo come eseguire una determinata operazione — dal cambio della pinza in una baia di picking alla creazione di un carico — redatti internamente e corredati di immagini. Sono la sorgente più direttamente operativa del corpus, e anche quella da cui sono tratti i documenti della campagna di valutazione del Capitolo 6. Dal punto di vista del sistema ricevono lo stesso trattamento della documentazione tecnica (sezione 4.3).

3.3.4 Natura multimodale del corpus

La componente non testuale è tutt'altro che marginale. In media il corpus presenta non meno di un'immagine ogni due pagine e una tabella ogni tre, con una distribuzione molto variabile fra documenti: alcuni ne sono densi, altri quasi privi. In molti casi il contenuto visuale non è ridondante rispetto al testo che lo circonda: è l'unico posto in cui l'informazione compare.

Le immagini sono in prevalenza acquisizioni dell'interfaccia del gestionale — schermate che mostrano quale maschera aprire, quale campo compilare, quale voce di menu selezionare. Una schermata non è un'illustrazione decorativa ma un contenuto che porta testo e struttura, e il modo in cui viene resa in forma testuale determina se quell'informazione resti recuperabile o vada perduta

(sezione 5.2.2). Le conversazioni contengono a loro volta fotografie e screenshot inviati per mostrare un errore; queste restano escluse dal trattamento, per le ragioni discusse nella sezione 5.1.6.

3.4 Definizione del problema

Le sorgenti descritte esistono e sono raggiungibili: il problema non è di archiviazione, ma di recupero. Occorre sapere che una certa informazione esiste, e sapere in quale delle sorgenti cercarla.

3.4.1 Limiti della ricerca tradizionale

Gli strumenti disponibili per cercare in questo materiale sono la ricerca per nome di file e la ricerca testuale all'interno dei documenti. Entrambe condividono lo stesso limite: operano sulle parole, non sul significato.

Chi cerca deve quindi indovinare il lessico esatto con cui l'informazione è stata scritta. Una procedura che parla di «disallineamento logico» non si trova cercando «il pallet risulta in una posizione sbagliata», anche se è esattamente quello che serve. Il problema si aggrava in un dominio in cui convivono il linguaggio comune degli operatori, la terminologia formale della documentazione e le sigle del gestionale: tre registri per la stessa cosa.

C'è poi una seconda limitazione, più radicale. Anche quando la ricerca trova il documento giusto, restituisce il documento — non la risposta. Chi cerca deve aprirlo, individuare la sezione pertinente e ricostruire da sé la procedura. Per una domanda puntuale, questo è gran parte del lavoro.

Infine, nessuno di questi strumenti raggiunge l'informazione che non è in un documento: ciò che è stato spiegato una volta in una conversazione con l'assistenza e mai trascritto altrove resta fuori portata.

3.4.2 Il costo del recupero

Le richieste che nascono da questa situazione sono molto variegate. A un estremo ci sono domande banali ma frequenti — dove si trova una certa locazione a partire dal suo codice — che costano

soltanto tempo. Nel mezzo si colloca la fascia più numerosa: le operazioni coperte da una procedura documentata — come il cambio della pinza citato sopra — che qualcuno ha scritto ma che non sempre si riesce a ritrovare nel momento in cui serve. All'altro estremo ci sono i problemi di funzionamento dell'impianto e del gestionale, per i quali, quando la soluzione non si trova, si ricorre all'assistenza esterna.

Quando la ricerca documentale fallisce, restano due canali, entrambi costosi. Il primo è la domanda a un collega: chi ha seguito il sistema negli anni sa dove guardare, o ricorda direttamente la risposta, e rivolgersi a lui costa meno che cercare. Il prezzo è un carico ricorrente sulle persone più esperte, interrotte da domande che in buona parte sono state poste in passato, e una conoscenza che continua a non depositarsi: ogni scambio risolve il problema contingente senza lasciare traccia recuperabile, e la dipendenza dalle stesse persone si rinnova. A questo si aggiunge il rischio di continuità: quando la conoscenza operativa risiede in poche persone, l'assenza o l'uscita di una di esse ha un impatto sproporzionato rispetto al suo ruolo formale.

Il secondo canale è l'assistenza del fornitore, che ha un costo economico diretto e che, come mostrato nella sezione 3.2, viene interpellata in larga parte per problemi già affrontati e risolti.

3.4.3 Perché un modello linguistico da solo non è sufficiente

Un modello linguistico, per quanto capace, non risolve il problema da solo: la documentazione proprietaria dell'azienda non fa parte della sua conoscenza, e interrogato su ciò che non conosce tende a produrre risposte plausibili ma non fondate, senza modo di risalire alla fonte (sezione 2.1.3). Serve un'architettura che ancori la generazione a documenti effettivamente recuperati dal corpus aziendale e renda verificabile il legame fra la risposta e le fonti: il paradigma RAG descritto nella sezione 2.5.

3.5 Obiettivo del sistema

L'obiettivo concreto del sistema è mettere il personale in condizione di trovare da sé la soluzione — nella documentazione, nelle procedure interne o nello storico dei problemi già affrontati con l'assistenza — prima di ricorrere al fornitore. Ogni problema risolto consultando il sistema è una richiesta di assistenza che non viene fatta, e quindi un costo evitato; allo stesso modo, ogni domanda a cui il sistema risponde è un'interruzione in meno per il personale più esperto.

Perché questo accada, la risposta deve essere affidabile. Una risposta sbagliata su una procedura operativa non fa risparmiare una richiesta di assistenza: nel migliore dei casi la rimanda, nel peggiore produce un errore in magazzino. Per questo il sistema cita sempre i documenti da cui la risposta è tratta, dichiara esplicitamente quando l'informazione non è disponibile, e la valutazione del Capitolo 6 misura anzitutto la correttezza delle risposte.

La riduzione effettiva delle richieste di assistenza è l'effetto atteso del sistema, non una grandezza misurata in questo lavoro: per stimarla occorre osservare il sistema in esercizio per un periodo sufficiente, ed è indicata fra gli sviluppi futuri (sezione 7.3.1). I requisiti che ne discendono sono formalizzati all'inizio del capitolo seguente.

4. Architettura del sistema

Descritti nel capitolo precedente il contesto, il problema e le sorgenti, questo capitolo descrive il sistema realizzato: da quali componenti è composto, come sono organizzati i dati e su quale infrastruttura viene eseguito. Risponde alle domande «che cosa è stato costruito» e «perché in questa forma»: per ogni componente dichiara il ruolo e le ragioni della scelta. Il *come* — la configurazione dei singoli stadi e il percorso che l'informazione compie dalle sorgenti all'indice e dalla domanda alla risposta — è oggetto del Capitolo 5, a cui il testo rimanda ogni volta che la descrizione scende a quel livello.

4.1 Requisiti e vincoli progettuali

4.1.1 Requisiti funzionali

I requisiti discendono dall'obiettivo descritto nella sezione 3.5: permettere al personale di trovare da sé, e più in fretta, la soluzione ai propri problemi, senza interrompere i colleghi più esperti e, nei casi più gravi, senza ricorrere all'assistenza esterna. Il sistema deve quindi rispondere in linguaggio naturale a domande poste dal personale interno sulla documentazione aziendale, citando per ogni risposta i documenti da cui l'informazione è stata tratta. Deve trattare in modo uniforme sorgenti di natura diversa — manualistica tecnica, conversazioni operative, procedure interne — e deve gestire i contenuti non testuali che queste sorgenti contengono, in particolare immagini e tabelle. Deve infine mantenere la cronologia della conversazione, così che l'utente possa porre domande di approfondimento senza ripetere il contesto, e distinguere gli utenti fra loro, perché non tutti hanno accesso agli stessi documenti.

In forma di elenco, il sistema deve:

- rispondere a domande poste in italiano, in linguaggio naturale, sulla documentazione aziendale;
- indicare per ogni risposta i documenti da cui l'informazione proviene;
- dichiarare esplicitamente quando l'informazione non è disponibile, invece di produrre una risposta non fondata;

- integrare documentazione tecnica, conversazioni operative e procedure interne in un unico spazio di ricerca;
- trattare i contenuti non testuali — immagini e tabelle — invece di scartarli;
- mantenere la cronologia della conversazione, per consentire domande di approfondimento;
- autenticare gli utenti e restituire a ciascuno risposte fondate soltanto sui documenti che è autorizzato a consultare;
- preprocessare e indicizzare i documenti tramite procedure che non richiedono interventi sul codice.

4.1.2 Requisiti non funzionali

Il sistema è destinato a un uso interattivo: la risposta deve arrivare in tempi compatibili con una conversazione, e la generazione in streaming serve proprio a rendere percepibile l'avanzamento invece di presentare un'attesa indifferenziata. Il numero di utenti concorrenti atteso è dell'ordine delle unità o delle poche decine, non delle migliaia: è un sistema interno a un reparto, non un servizio pubblico, e questo dimensionamento è ciò che rende praticabile ospitarlo su una sola GPU, senza replica né bilanciamento del carico.

Il preprocessing e l'indicizzazione dei documenti devono poter essere eseguiti da personale che non ha sviluppato il sistema. Analogamente, la creazione di nuovi utenti e la manutenzione ordinaria devono essere possibili tramite procedure documentate e non richiedere interventi sul codice.

Sul fronte dell'ingestion, l'elaborazione completa di un documento ha richiesto, nell'esperienza d'uso, al più un minuto, con il collo di bottiglia costituito dalla descrizione delle immagini da parte del modello visione-linguaggio: il parsing vero e proprio incide in misura molto minore.

In interrogazione i tempi di risposta sono risultati paragonabili a quelli di un assistente conversazionale commerciale: la generazione in streaming inizia dopo un'attesa breve e prosegue a un ritmo confrontabile, con 15–20 secondi per una risposta completa, quasi interamente spesi nella generazione (Tabella 5.2).

4.1.3 Il vincolo di esecuzione locale

Il vincolo più stringente dell'intero progetto, già introdotto nella sezione 1.3, è che il sistema deve essere eseguito *interamente su infrastruttura aziendale*: nessun documento, nessuna domanda e nessuna risposta possono essere inviati a servizi di terze parti. Sono quindi esclusi i modelli accessibili solo tramite API remote, i servizi gestiti di embedding e i database vettoriali offerti in modalità *managed*.

Il vincolo ha tre conseguenze dirette sulla progettazione, che ricorrono in tutto il capitolo.

Scelta dei modelli limitata ai pesi aperti.
Ogni componente che coinvolge un modello — embedding, reranking, descrizione delle immagini, generazione — deve poter essere scaricato, ospitato ed eseguito localmente. Questo esclude i modelli proprietari più capaci e impone di lavorare con modelli di dimensione contenuta, compensando con la qualità della pipeline ciò che non si può ottenere dalla scala del modello.

Memoria video come risorsa scarsa e condivisa.
Tutti i modelli attivi durante una richiesta insistono sulla stessa scheda grafica. La dimensione del modello generativo, il numero di modelli caricati contemporaneamente e la lunghezza massima del contesto non sono parametri indipendenti: competono per la stessa memoria. Il dimensionamento è discusso nella sezione 4.8.

Costo computazionale di ogni passaggio aggiuntivo.
Ogni stadio in più della pipeline — una riformulazione della query, una seconda chiamata al modello, un reranking su più candidati — si traduce in tempo di GPU sottratto alle altre richieste. Una tecnica viene adottata solo se il guadagno misurato sulla qualità giustifica il costo; è il criterio che guida le scelte discusse nella sezione 5.10 e verificate nel Capitolo 6.

Il vincolo riguarda il sistema in esercizio, ed è su di esso che vale senza eccezioni: nessuno stadio della pipeline che risponde alle domande degli utenti effettua chiamate verso l'esterno. Due attività svolte una tantum durante lo sviluppo, al di fuori di quel percorso, hanno invece fatto uso di modelli accessibili come servizio, in entrambi i casi su materiale che non ricade sotto il vincolo di riservatezza.

La prima è l'estrazione e la deduplicazione dei problemi dalle conversazioni operative: il materiale è stato anonimizzato in locale, in modo irreversibile, *prima* di qualunque elaborazione esterna (sezione 5.1.2). La seconda è la campagna di valutazione — la generazione delle domande, l'assegnazione dei chunk e il giudizio delle risposte — condotta su un sottoinsieme di documenti selezionati fra quelli il cui contenuto non è soggetto a riservatezza (sezione 6.1.1).

4.1.4 Vincoli sul trattamento dei dati

Oltre al vincolo di esecuzione locale, tre condizioni riguardano il modo in cui i dati possono essere trattati.

La **segregazione per gruppi**: non tutti i documenti sono visibili a tutti gli utenti, e la distinzione deve valere anche per le risposte generate, non solo per l'accesso diretto ai file. È il requisito che motiva la sezione 4.5.

La **tracciabilità**: ogni risposta prodotta dal sistema deve essere riconducibile ai documenti da cui è tratta, perché l'utente possa verificarla. Un sistema che risponda correttamente ma senza fonti non è utilizzabile in un contesto operativo, dove il costo di un errore non rilevato è alto.

L'**anonimizzazione delle conversazioni**: il materiale conversazionale contiene nomi di persone, riferimenti a clienti e dati commerciali che non devono entrare nell'indice. È il requisito che motiva la procedura descritta nella sezione 5.1.2.

4.2 Panoramica del sistema

Il sistema si articola su tre piani distinti, che si incontrano soltanto nel database vettoriale.

Il primo piano è l'**ingestion**, un processo *offline* che trasforma le sorgenti eterogenee in chunk di testo indicizzabili. È un processo batch, rieseguibile, avviato tramite uno script dedicato.

Il secondo piano è l'**indicizzazione**: i chunk prodotti dall'ingestion vengono trasformati in rappresentazioni vettoriali e memorizzati in Qdrant insieme ai metadati che ne descrivono la provenienza e la visibilità.

Il terzo piano è l'**interrogazione**, il percorso *online* che va dalla domanda dell'utente alla risposta citata. È il piano che coinvolge l'interfaccia conversazionale, il database vettoriale e i tre modelli serviti sulla GPU.

La Figura 4.1 mostra l'architettura complessiva e la collocazione dei componenti.

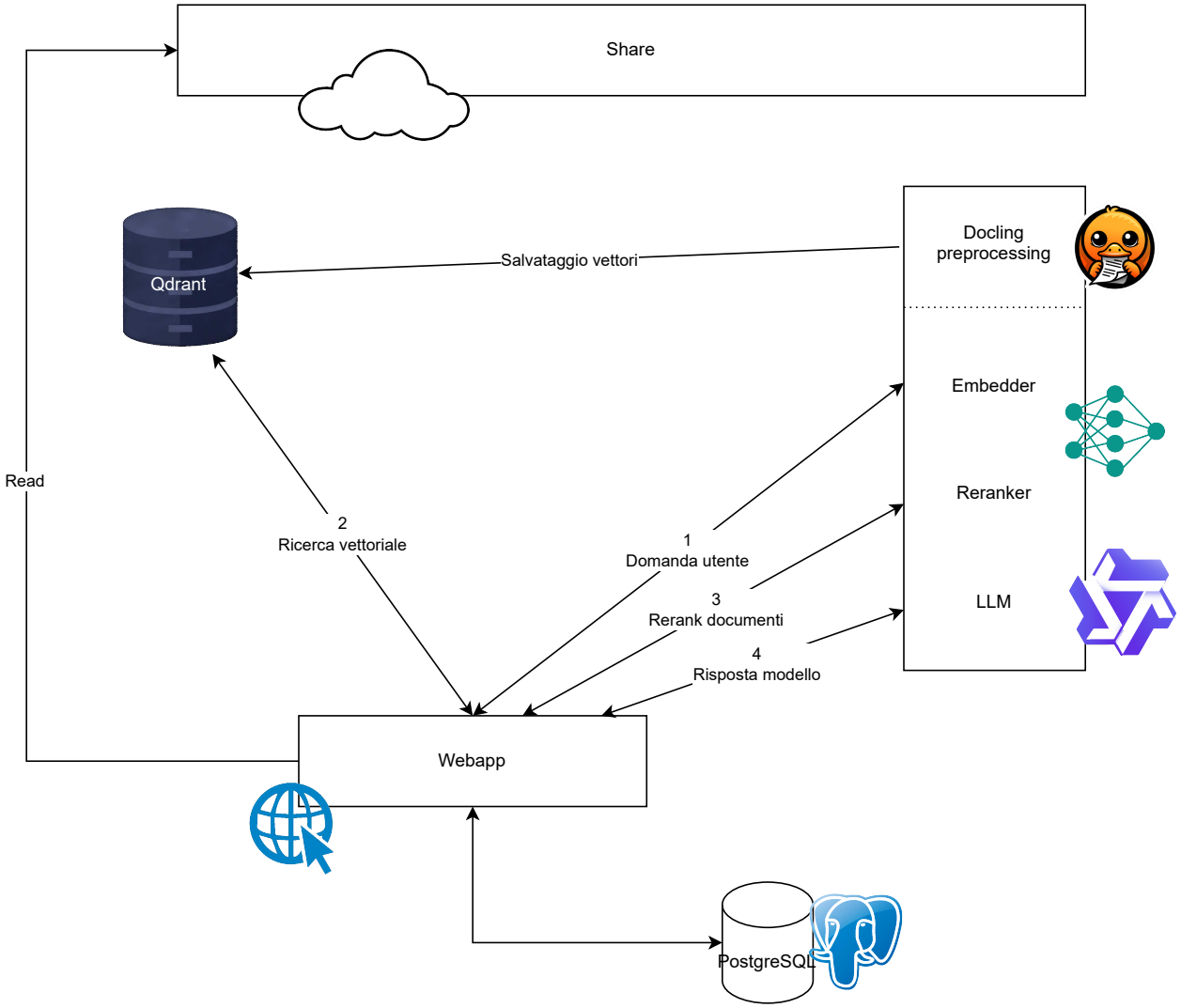


Figura 4.1: Schema generale dell'architettura. La webapp legge i documenti dalla cartella condivisa, interroga Qdrant per la ricerca vettoriale e i tre servizi di inferenza per embedding, reranking e generazione; PostgreSQL ospita utenti, gruppi e cronologia delle conversazioni.

La Tabella 4.1 riassume i componenti del sistema e la tecnologia adottata per ciascuno.

Tabella 4.1: Componenti del sistema e tecnologie adottate.

Componente	**Tecnologia**	**Funzione**
Parsing documenti	Docling	Estrazione di testo, tabelle e immagini
Descrizione immagini	`Qwen3-VL-8B-Instruct-FP8`	Conversione delle immagini in testo descrittivo
Chunking	LlamaIndex `SentenceSplitter`	Chunk fino a 1000 token, sovrapposizione fino a 200
Embedding denso	`BAAI/bge-m3`	Rappresentazioni dense (1024 dimensioni)
Canale lessicale	BM25	Rappresentazioni sparse per il recupero ibrido
Indice vettoriale	Qdrant	Memorizzazione, ricerca ibrida, filtri sui metadati
Reranking	`BAAI/bge-reranker-v2-m3`	Riordinamento dei candidati recuperati
Generazione	`Qwen/Qwen3-32B-AWQ`	Produzione della risposta citata
Servizio di inferenza	vLLM	Hosting dei modelli su GPU
Orchestrazione	LlamaIndex	Collegamento fra i componenti della pipeline
Interfaccia	Chainlit	Chat, autenticazione, cronologia
Base dati applicativa	PostgreSQL	Utenti, gruppi, conversazioni

4.3 Preprocessing e ingestion

4.3.1 Architettura della pipeline di ingestion

L'ingestion è un processo batch che porta le sorgenti a una rappresentazione comune prima dell'indicizzazione. La Figura 4.2 ne mostra i due rami. Il ramo dei documenti — manualistica e fogli di lavoro interni — passa per Docling, che scompone ciascun documento in testo, tabelle e immagini: le tabelle vengono convertite in Markdown e le immagini sostituite da una descrizione prodotta dal modello visione-linguaggio, così che il risultato sia un unico documento di testo strutturato. Il ramo delle conversazioni passa per una catena di elaborazioni con modelli linguistici — anonimizzazione in locale, estrazione dei problemi, deduplicazione — e produce schede di problema. Entrambi i rami confluiscono nella stessa collection di Qdrant.

La decisione architetturale più rilevante di questa fase è ricondurre anche il contenuto visuale a testo, invece di adottare embedding multimodali o il recupero diretto sulla pagina renderizzata discussi nella sezione 2.7. La scelta mantiene un unico spazio di rappresentazione per l'intero corpus, il che rende il recupero ibrido e il controllo degli accessi uniformi su tutte le sorgenti,

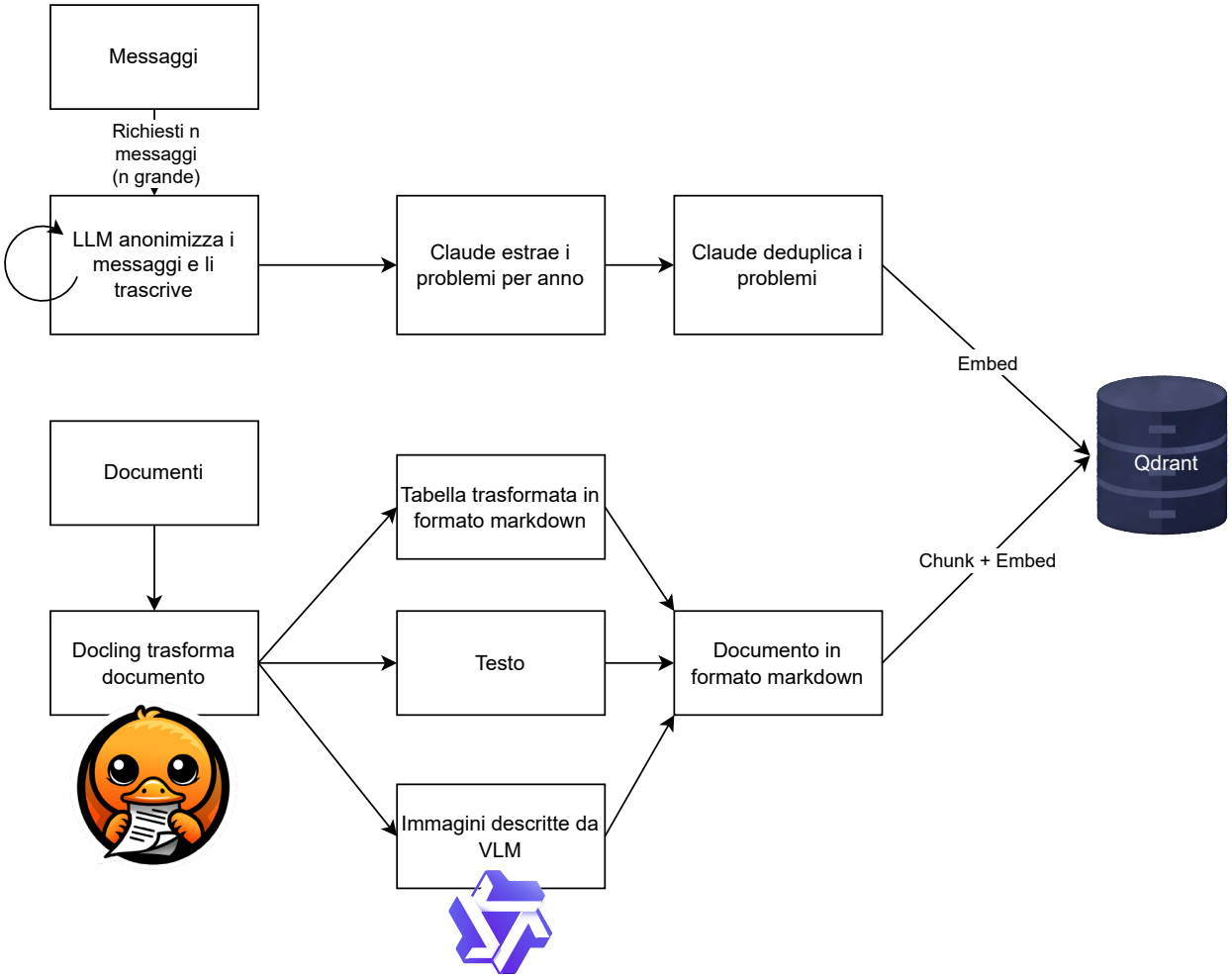


Figura 4.2: Pipeline di ingestion. I documenti vengono scomposti da Docling in testo, tabelle e immagini e ricomposti in un unico documento Markdown; le conversazioni vengono anonimizzate in locale e poi trasformate in schede di problema. Entrambi i rami confluiscono nell'indice vettoriale.

e non richiede in esercizio un embedder e un modello generativo multimodali, più pesanti. Il funzionamento dei singoli passaggi è descritto nel Capitolo 5: le conversazioni nella sezione 5.1, i documenti nella sezione 5.2.

4.3.2 Esecuzione dell'ingestion

I documenti hanno un proprio script, che li preprocessa secondo la pipeline appena descritta e ne indicizza successivamente i chunk, senza richiedere interventi sul codice. L'esecuzione è riservata al personale dell'ufficio tecnico logistico, che è anche la funzione con visibilità su tutti i gruppi.

La visibilità di ciascun documento è determinata dal suo nome, secondo la convenzione descritta nella sezione 4.5.2: la procedura presuppone quindi che i documenti siano nominati rispettando quella convenzione, ed è il punto discusso nella sezione 4.5.4.

4.4 Storage e indicizzazione

4.4.1 Il database vettoriale

I chunk vengono rappresentati con il modello di embedding BGE-M3 e memorizzati in Qdrant, eseguito in un container Docker [8]. La scelta di Qdrant risponde a tre esigenze contemporaneamente: può essere eseguito interamente in locale, come impone il vincolo della sezione 4.1.3; supporta nativamente vettori densi e sparsi nella stessa collection, il che rende naturale il recupero ibrido descritto nella sezione 5.5.3; e permette di filtrare la ricerca sui metadati in modo efficiente, che è il meccanismo su cui poggia il controllo degli accessi.

4.4.2 Organizzazione delle collection

Tutte le sorgenti confluiscono in un'unica collection. La scelta ha tre conseguenze desiderabili. La prima è che una domanda può essere risolta indifferentemente da un manuale, da un foglio interno o da una scheda di problema estratta dalle conversazioni, senza che il sistema debba decidere in anticipo dove cercare. La seconda è che il controllo degli accessi resta un unico meccanismo, applicato a un solo indice, invece di dover essere replicato e mantenuto coerente su più collection. La terza è che i due canali, denso e sparso, interrogano lo stesso insieme di punti, così che le due graduatorie da fondere si riferiscano alla stessa popolazione di chunk.

Il rovescio della medaglia è che sorgenti di natura molto diversa competono nello stesso spazio di ricerca: una scheda di problema e una sezione di manuale vengono confrontate con lo stesso criterio, pur avendo registro e struttura differenti. La distinzione resta comunque ricavabile dai metadati (nome e formato del file di origine), e potrebbe essere usata in futuro per strategie di recupero differenziate.

4.4.3 Schema dei metadati

A ogni chunk indicizzato sono associati tre campi di metadati applicativi, riportati nella Tabella 4.2. Sono pochi e hanno ciascuno una funzione precisa: il nome del file serve alle citazioni, il gruppo al controllo degli accessi, il formato a identificare il tipo di sorgente.

Il campo `groups` merita una precisazione, perché il suo nome al plurale può trarre in inganno: contiene un *singolo* valore testuale, il gruppo proprietario del documento. Non è quindi il campo a esprimere la visibilità estesa di un documento; è il filtro costruito al momento della ricerca a farlo, confrontando quel valore con l'insieme dei gruppi accessibili all'utente secondo la mappa descritta nella sezione 4.5.1; la traduzione in una condizione di ricerca è nella sezione 5.6. I tre campi sono valorizzati in fase di ingestion, come descritto nella sezione 5.2.4. Le schede di problema ricavate dalle conversazioni ricevono il gruppo `UTL`, e sono quindi visibili al solo ufficio tecnico logistico.

Tabella 4.2: Campi del payload associato a ciascun chunk in Qdrant.

Campo	**Tipo**	**Funzione**
`origin_filename`	testo	Nome del documento di origine; usato nelle citazioni
`groups`	testo	Gruppo proprietario del documento; usato per il filtro sui permessi
`origin_mimetype`	testo	Formato del documento di origine

Il payload è deliberatamente minimale. Una versione precedente prevedeva anche il numero di pagina di provenienza di ciascun chunk, che avrebbe permesso citazioni più precise, ma l'informazione non sopravviveva al passaggio fra la rappresentazione di Docling e l'indicizzazione con LlamaIndex: la provenienza è presente nel documento analizzato, ma non viene propagata ai nodi prodotti dal suddivisore. Il campo è stato quindi rimosso invece di essere mantenuto con valori inaffidabili, e la citazione si ferma al documento, come discusso nella sezione 5.9.2.

4.4.4 Configurazione dell'indice

La collection è configurata con due spazi vettoriali distinti sullo stesso insieme di punti: un vettore denso di 1024 dimensioni, prodotto da BGE-M3 e confrontato con la similarità del coseno, e un

vettore sparso BM25 (Codice 4.1). È la configurazione che rende possibile il recupero ibrido descritto nella sezione 5.5.3 senza mantenere due collection separate.

Codice 4.1: Creazione della collection con vettori densi e sparsi.

```
client.create_collection(
    collection_name=collectionname,
    vectors_config={
        "bge_m3": VectorParams(size=1024, distance=Distance.COSINE)
    },
    sparse_vectors_config={
        "bm25": SparseVectorParams()
    },
)
```

La ricerca per similarità si appoggia a un indice HNSW, che il motore costruisce e mantiene automaticamente sui vettori densi di ogni collection. I suoi parametri non sono stati impostati esplicitamente: la configurazione predefinita si è rivelata adeguata alle dimensioni del corpus, e non c'è stata evidenza che il recupero costituisse un collo di bottiglia. La scelta è coerente con il principio seguito in tutto il progetto: intervenire su un parametro solo quando una misura lo giustifica.

4.5 Controllo degli accessi

Non tutti gli utenti possono vedere tutti i documenti. Il principio adottato è che il sistema non deve limitarsi a nascondere all'utente un documento non autorizzato: deve fare in modo che quel documento non entri mai nel contesto fornito al modello generativo, e quindi non possa influenzare la risposta nemmeno in forma indiretta.

4.5.1 Modello di autorizzazione

Ogni utente è associato a un gruppo, e ogni gruppo ha accesso a un sottoinsieme dei documenti. La relazione è espressa da una mappa che associa a ciascun gruppo l'elenco dei gruppi di documenti a cui può accedere, riportata nella Tabella 4.4.

Tre dei quattro gruppi corrispondono ad altrettante funzioni aziendali; il quarto, PUB, raccoglie il materiale accessibile a tutti (Tabella 4.3).

Tabella 4.3: I gruppi definiti e le funzioni aziendali corrispondenti.

Sigla	Funzione	Tipo di documentazione
PUB	Documentazione pubblica	Materiale accessibile a chiunque in azienda
LOG	Logistica d'ufficio	Procedure gestionali e operative di reparto
MAN	Manutentori	Interventi tecnici su impianti e automazioni
UTL	Ufficio tecnico logistico	Configurazione e gestione complessiva del sistema

La visibilità fra i gruppi è annidata, e riflette il fatto che l'ufficio tecnico logistico è la funzione che sovrintende a tutte le altre e ha quindi bisogno di accedere a qualunque documento.

Tabella 4.4: Mappa delle visibilità: per ogni gruppo di utenti, i gruppi di documenti accessibili.

Gruppo utente	Documenti accessibili
UTL	UTL, MAN, LOG, PUB
MAN	MAN, LOG, PUB
LOG	LOG, PUB
PUB	PUB

Un utente del gruppo MAN, per esempio, vede i documenti dei manutentori, quelli della logistica d'ufficio e quelli pubblici, ma non quelli riservati all'ufficio tecnico.

La scelta di un modello a gruppi, invece di permessi assegnati per singolo documento o per singolo utente, risponde a un criterio di gestibilità: i profili di accesso in azienda sono pochi e stabili, mentre i documenti sono molti.

4.5.2 Associazione documenti–gruppi

L'associazione fra documento e gruppo è codificata nel nome del file: la porzione che precede il primo carattere di sottolineatura è la sigla del gruppo proprietario. Un documento denominato

`MAN_Arresto_baie_picking.pdf` appartiene quindi al gruppo dei manutentori, ed è di conseguenza accessibile agli utenti `MAN` e `UTL`; `PUB_Creazione_nuove_UDC.pdf` appartiene al gruppo pubblico ed è accessibile a tutti. Il nome dichiara così un solo gruppo, che in fase di ingestion diventa il valore del campo `groups` di ogni chunk del documento; chi altro possa vederlo lo stabilisce la Tabella 4.4.

La soluzione ha il pregio della semplicità operativa — la visibilità di un documento è stabilita dal suo nome, senza una base dati separata da mantenere — e il limite di appoggiare una proprietà di sicurezza a una convenzione di denominazione. Il punto è ripreso nella sezione 4.5.4.

4.5.3 Filtraggio in fase di recupero

Il punto qualificante dell'architettura è che il recupero è *condizionato dai permessi*: la ricerca per similarità non viene eseguita sull'intero indice per poi scartare i risultati non autorizzati, ma viene ristretta a priori al sottoinsieme di documenti visibili all'utente che ha formulato la domanda. È il database stesso ad applicare la restrizione, su entrambi i canali del recupero ibrido.

È quindi un *pre-filtering*, e non un filtro applicato a valle, per le ragioni discusse nella sezione 2.6.2: evita di restare senza contesto quando i primi risultati non sono autorizzati, e fa sì che un contenuto riservato non entri mai nella pipeline, nemmeno come candidato da scartare. Come la condizione viene costruita a partire dalla sessione dell'utente e applicata ai due canali è descritto nella sezione 5.6.

4.5.4 Garanzie e superficie di attacco residua

Il meccanismo garantisce che nessun chunk non autorizzato entri nel contesto passato al modello generativo, perché il filtro agisce sulla ricerca e non sui risultati. Due limiti vanno però dichiarati.

Il primo è che l'associazione documento–gruppo poggia su una convenzione di denominazione: un documento depositato con un prefisso sbagliato — o privo di prefisso — riceve la visibilità che il suo nome dichiara, non quella che dovrebbe avere. La correttezza del controllo dipende quindi dalla disciplina con cui i documenti vengono nominati.

Il secondo riguarda i canali indiretti: la visibilità è decisa per documento e non per contenuto, quindi un documento visibile che riporti al suo interno contenuti riservati li rende accessibili a

chi può leggerlo. Anche in questo caso il rischio dipende da come i documenti sono redatti ed etichettati, non dal meccanismo di filtro.

4.6 Backend e orchestrazione

La comunicazione con i modelli di embedding e di generazione è mediata da LlamaIndex [23], che fornisce le astrazioni per il collegamento fra i componenti della pipeline e per l'interazione con Qdrant.

Del framework vengono usate le astrazioni di più alto livello dove portano un vantaggio reale: il documento e il nodo come unità di testo con metadati, il *vector store* come interfaccia verso Qdrant, il *retriever* con i suoi parametri di modalità e di filtro, e i client compatibili con l'interfaccia OpenAI per dialogare con i modelli serviti da vLLM. Il reranking è invece implementato direttamente come chiamata al servizio dedicato, perché il passaggio attraverso l'astrazione del framework non aggiungeva nulla e rendeva meno esplicito il controllo sul numero di candidati.

Il codice è organizzato in tre repository, corrispondenti ai tre blocchi funzionali del sistema: il preprocessing dei documenti — che comprende anche il codice della campagna di valutazione — il trattamento delle conversazioni e l'applicazione conversazionale; l'Appendice A ne riporta la corrispondenza con le sezioni della tesi. Gli indirizzi dei servizi — il motore di inferenza, il database vettoriale, la base dati applicativa — e i segreti dell'applicazione sono esterni al codice e risolti da variabili d'ambiente, così che la stessa base di codice possa essere puntata su istanze diverse senza modifiche.

I parametri della pipeline, invece, non sono raccolti in un file di configurazione: la campagna del Capitolo 6 li fa variare direttamente nello script di valutazione, che itera sulla griglia dei valori di k e dei budget di token ed esegue la pipeline completa per ciascuna combinazione.

4.7 Frontend e autenticazione

4.7.1 Interfaccia conversazionale

L'interfaccia utente e la gestione dell'autenticazione sono realizzate con Chainlit [3], che offre un'esperienza d'uso simile a quella di ChatGPT. Ogni utente dispone di un proprio account, con

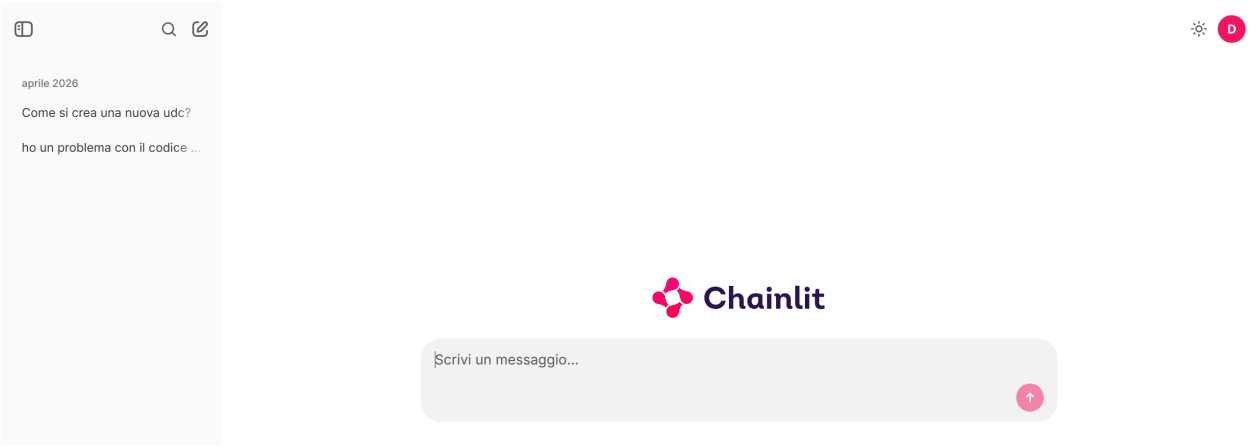


Figura 4.3: Schermata iniziale dell'interfaccia conversazionale. Sulla sinistra la cronologia delle conversazioni dell'utente autenticato, raggruppata per mese e ricercabile; al centro il campo di immissione della domanda. L'immagine riporta l'interfaccia nella configurazione predefinita del framework: nell'installazione aziendale i marchi sono sostituiti da quelli dell'azienda.

cui può interrogare il sistema e ritrovare la cronologia delle conversazioni precedenti.

La scelta di un'interfaccia già familiare non è secondaria. Gli utenti destinatari del sistema non hanno una formazione informatica specifica, e l'adozione di uno strumento interno dipende in misura non trascurabile da quanto poco esso richieda di imparare: un campo di testo, l'invio della domanda, la risposta che scorre. La Figura 4.3 mostra la schermata iniziale.

La barra laterale conserva le conversazioni passate dell'utente, che vengono persistite su PostgreSQL insieme al suo account: è il meccanismo che consente di riprendere una richiesta interrotta e di ritrovare una risposta già ottenuta senza doverla formulare di nuovo. Il modo in cui le risposte espongono i documenti da cui sono tratte è discusso nella sezione 5.9.2, insieme al formato delle citazioni.

4.7.2 Gestione degli utenti e dei gruppi

I nuovi account vengono creati tramite uno script dedicato, eseguito da chi gestisce il sistema; non esiste una registrazione autonoma. Ogni utente viene associato in quel momento a un gruppo, che ne determina la visibilità sui documenti secondo la mappa descritta nella sezione 4.5.1.

Per la memorizzazione delle credenziali e dei gruppi è stata aggiunta al database PostgreSQL di Chainlit — eseguito anch'esso, come Qdrant, in un container Docker — una tabella, accanto a

quelle che il framework usa per utenti, conversazioni e messaggi:

Codice 4.2: Tabella per l'autenticazione e l'associazione ai gruppi.

```
CREATE TABLE IF NOT EXISTS auth_user (
    identifier TEXT PRIMARY KEY,
    password   TEXT NOT NULL,
    gruppo     TEXT NOT NULL
);
```

Il campo `identifier` coincide con quello che Chainlit usa per identificare l'utente, il che permette di collegare l'autenticazione alla cronologia delle conversazioni senza duplicare l'anagrafica.

La password non viene mai memorizzata in chiaro. Prima dell'hashing le viene concatenato un segreto applicativo — un *pepper*, conservato nella configurazione del servizio e non nella base dati — e il risultato viene sottoposto a bcrypt [30] con fattore di costo 12. La scelta di bcrypt rispetto a una funzione di hash generica è motivata dalle proprietà che servono in questo contesto: il salt è incorporato e diverso per ogni utente, quindi due password uguali producono hash diversi; il fattore di costo è configurabile e può essere alzato nel tempo; e la funzione è deliberatamente lenta e poco parallelizzabile, il che la rende resistente agli attacchi a dizionario condotti con hardware dedicato. Il pepper aggiunge un'ulteriore difesa: un attaccante che ottenesse il solo dump della base dati non disporrebbe del segreto necessario a verificare le proprie ipotesi.

4.8 Infrastruttura hardware

4.8.1 La macchina di inferenza

Coerentemente con il vincolo di esecuzione locale discusso nella sezione 4.1.3, l'intero sistema è ospitato su infrastruttura interna all'azienda. Il carico computazionale è però concentrato quasi per intero sulla macchina di inferenza, che ospita i modelli ed è equipaggiata con una GPU NVIDIA GeForce RTX 5090 da 32 GB di memoria video. La scelta di una scheda consumer di fascia alta, invece di acceleratori da datacenter, è essa stessa una conseguenza del vincolo: offre un buon compromesso fra memoria disponibile, prestazioni di inferenza e costo sostenibile per un'installazione interna, senza ricorrere a capacità noleggiata in cloud.

I 32 GB di memoria video costituiscono il budget entro cui devono coesistere i modelli attivi durante una richiesta — il modello generativo quantizzato, il modello di embedding e il reranker — e la cache del contesto. È questo budget — e non la qualità astratta dei modelli disponibili — a determinare la dimensione del modello generativo e la lunghezza massima del contesto costruibile.

La macchina dispone inoltre di 32 GB di memoria centrale. In esercizio la memoria video è occupata quasi per intero dai tre modelli serviti in parallelo, secondo la ripartizione della Tabella 4.5.

Tabella 4.5: Ripartizione della memoria video fra i modelli in esercizio.

Modello	**Quota di memoria video**
Modello generativo (`Qwen3-32B-AWQ`)	85 %
Embedder (`bge-m3`)	5 %
Reranker (`bge-reranker-v2-m3`)	5 %

La lettura della tabella è istruttiva. Il modello generativo assorbe da solo la quasi totalità della scheda, mentre embedder e reranker — i due stadi su cui il lavoro ha investito di più in termini di progettazione — occupano insieme un decimo della memoria e sono computazionalmente trascurabili. La quota del modello generativo comprende sia i pesi sia la cache del contesto: in pratica, è la parte di quell'85 % che resta libera dopo il caricamento dei pesi a fissare il limite superiore alla finestra utilizzabile e al numero di sequenze concorrenti.

È anche la ragione per cui il modello visione-linguaggio non può restare residente e viene istanziato soltanto in fase di ingestion: non ci sarebbe spazio per ospitarlo accanto agli altri tre. Durante l'ingestion il modello generativo non è quindi in esecuzione, e il VLM viene caricato direttamente dal codice nella memoria lasciata libera dal modello generativo.

4.8.2 Servizio di inferenza

I modelli sono serviti con vLLM, organizzati in tre container Docker distinti: uno per il modello generativo, uno per l'embedder e uno per il reranker. Il modello visione-linguaggio, per la ragione appena detta, non ha un container dedicato.

La finestra di contesto del modello generativo è fissata a 12 288 token. È il parametro che, a valle, determina quanto spazio resta per i chunk recuperati e per la cronologia della conversazione, e con cui si confronta la costruzione del contesto descritta nella sezione 5.8.

I parametri di servizio del modello generativo sono riportati nel Codice 4.3.

Codice 4.3: Parametri di servizio del modello generativo.

```
gpu_memory_utilization: 0.85
max_model_len: 12288
max_num_seqs: 4
enable_prefix_caching: true
trust_remote_code: false
```

Tre parametri meritano un commento, perché traducono in configurazione le scelte discusse finora.

Il primo è `gpu_memory_utilization`, che riserva al modello generativo l'85 % della scheda: è il valore riportato nella Tabella 4.5, e lascia il margine necessario agli altri due servizi.

Il secondo è `max_num_seqs`, fissato a quattro sequenze concorrenti. È un numero basso, coerente con il profilo d'uso descritto nella sezione 4.1.2 — pochi utenti, le cui richieste di rado arrivano nello stesso istante — e con il fatto che ogni sequenza in più consuma cache del contesto, cioè proprio la risorsa più scarsa.

Il terzo è `enable_prefix_caching`, che riutilizza la cache dei prefissi comuni fra richieste diverse. In questo sistema il beneficio riguarda la parte fissa del prompt di sistema, che precede i chunk recuperati ed è identica in ogni richiesta: il motore non deve rielaborarla ogni volta.

5. La pipeline RAG

Questo capitolo segue il percorso completo dell'informazione: prima come le sorgenti vengono preparate e indicizzate, poi come una domanda dell'utente attraversa la pipeline fino a diventare una risposta citata. Dove il Capitolo 4 ha descritto i componenti e le ragioni della loro scelta, qui interessa come sono configurati e che cosa accade al dato in ciascuno stadio; i componenti già presentati vengono richiamati senza ripeterne la motivazione. La Figura 5.1 riassume il secondo percorso e costituisce il riferimento per le sezioni dalla 5.5 in poi.

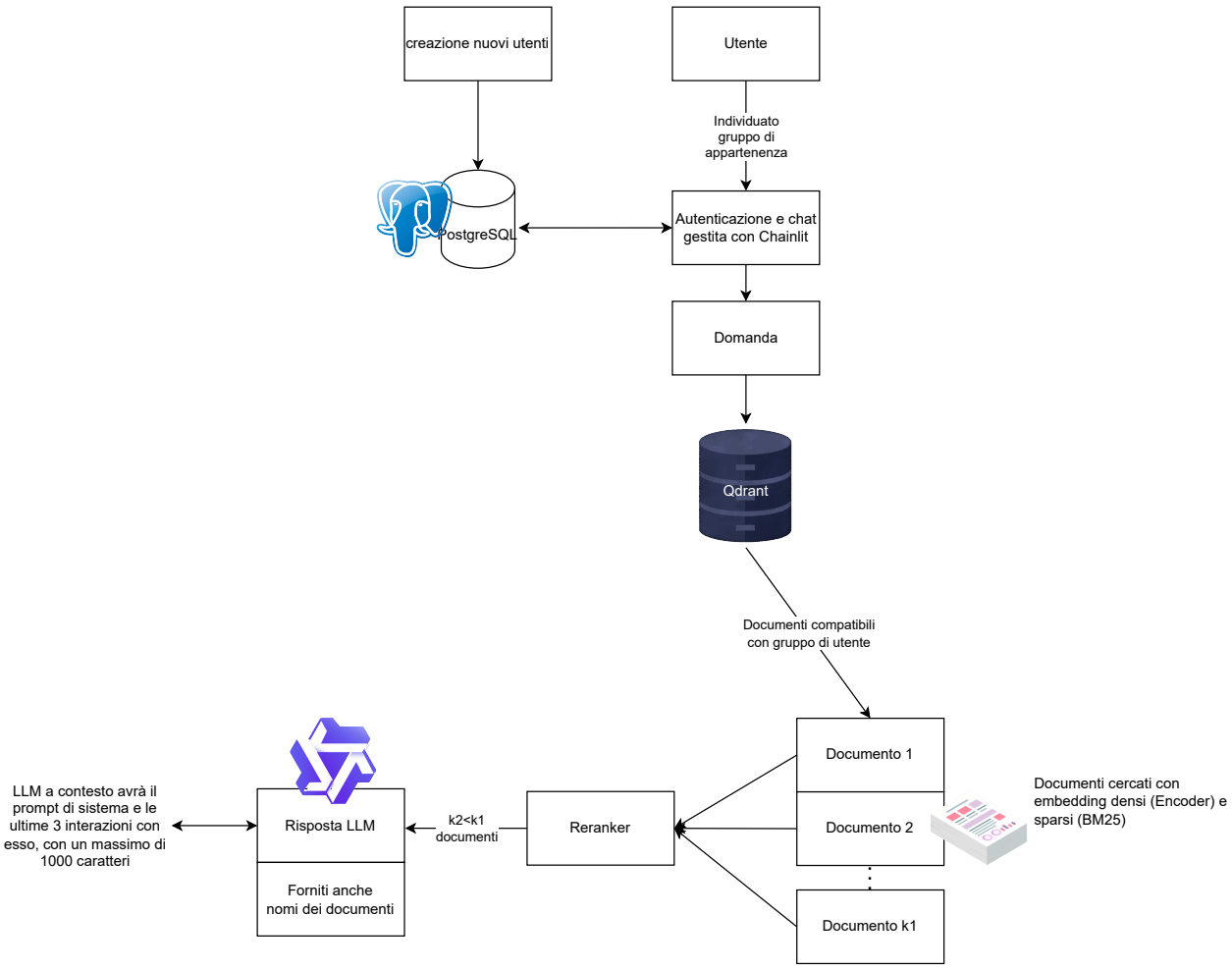


Figura 5.1: Percorso di una richiesta. L'utente autenticato tramite Chainlit pone una domanda; Qdrant restituisce soltanto i chunk compatibili con il suo gruppo, recuperati con il canale denso (BGE-M3) e con quello lessicale (BM25) e fusi con RRF; il reranker seleziona i k_2 chunk migliori, che vengono passati al modello generativo insieme al prompt di sistema e alla cronologia.

5.1 Preprocessing delle conversazioni

Le conversazioni operative, presentate nella sezione 3.3.2, sono la sorgente più difficile da trattare, perché non hanno la forma di un documento. Questa sezione descrive la catena di trasformazioni che le porta a una forma indicizzabile: anonimizzazione, estrazione dei problemi, deduplicazione.

5.1.1 Struttura del dato grezzo

Il materiale di partenza è l'esportazione testuale di una conversazione di gruppo, in cui ogni riga porta data, ora, autore e contenuto del messaggio. Il rumore è quello tipico di una chat di lavoro: saluti e convenevoli, messaggi di servizio, richieste rimaste senza risposta, riferimenti ad allegati non più disponibili. La difficoltà maggiore non è però il rumore, ma la *sovrapposizione*: mentre si discute di un problema ne viene segnalato un altro, e le due conversazioni procedono intrecciate per decine di messaggi.

5.1.2 Anonimizzazione

Il primo passo, e l'unico che precede qualunque altra elaborazione, è l'anonimizzazione. Il file di conversazione viene scorso a finestre di cinquanta righe, e ogni finestra viene data in input al modello generativo locale — Qwen3 32B in quantizzazione AWQ, lo stesso usato in esercizio — con l'istruzione di sostituire quattro classi di entità con altrettante etichette generiche: i nomi di persona con `Persona`, i nomi di azienda o cliente con `Azienda`, gli identificativi (codici numerici lunghi, identificativi di pallet e di unità di carico) con `COD` e gli importi monetari con `PREZZO`. La stessa istruzione chiede di sostituire il nome proprio del gestionale con il termine generico «wms».

Il prompt elenca anche ciò che *non* va toccato, perché è informazione necessaria alle fasi successive: i timestamp, che servono a ricostruire le occorrenze di un problema nel tempo; la terminologia tecnica del dominio; i nomi dei file allegati.

Due proprietà di questa fase meritano di essere sottolineate. La prima è che ogni finestra viene elaborata in isolamento e che non viene mantenuta alcuna corrispondenza fra entità originale ed etichetta: la stessa persona citata in due punti diversi diventa semplicemente `Persona` in entrambi, senza numerazione. L'anonimizzazione è quindi *irreversibile*, perché non esiste da nessuna parte una mappa che permetta di risalire al testo originale. La seconda è che avviene interamente in locale, ed è una precondizione e non un accorgimento successivo: i passaggi descritti nelle sottosezioni seguenti operano su dati già privi di riferimenti identificativi. È ciò che ha reso ammissibile, per i soli passaggi successivi, il ricorso a modelli accessibili come servizio, come discusso nella sezione 4.1.3.

Il prompt completo è riportato nell'Appendice B (Codice B.2). Il Codice 5.1 mostra un estratto del risultato.

Codice 5.1: Estratto di conversazione dopo l'anonimizzazione.

```
24/06/21, 14:15 - Persona: @Persona @Persona hi guys good afternoon
Consulting the troubleshooting guide, we have not fought a point
reguarding the error "wait for storage failed" on mf200; someone could
help us?
24/06/21, 14:17 - Persona: More precisly, someone could say to us how to
fix It? making in manner that we can be indipendent
24/06/21, 14:35 - Persona: @Persona The troubleshooting guide does not
contain this as this error has a very broad range that can vary from
case to case. If you are experiencing problems with a specific pallet,
that one needs to be investigated. @Persona for confirmation, in case of
things like this, should they do a ticket or post it here?
24/06/21, 14:52 - Persona: Blocked
24/06/21, 14:57 - Persona: Please use Azienda maintainance team to check,
if they can not resolved please open ticket
24/06/21, 15:03 - Persona: @Persona we are checking
```

L'estratto mostra anzitutto che cosa l'anonimizzazione conserva: la sigla della stazione resta leggibile, perché designa un impianto e non rientra fra gli identificativi da sostituire, e i timestamp sono intatti. Mostra poi ciò che rende difficile la fase successiva: il problema — un errore su una stazione — non trova qui una soluzione, ma solo un rimando ad altri interlocutori.

L'efficacia della sostituzione è stata verificata manualmente a campione sul materiale prodotto, senza che emergessero falsi negativi, cioè entità sfuggite all'anonimizzazione.

5.1.3 Estrazione dei problemi

Il file anonimizzato viene suddiviso per anno, e ciascuna porzione viene elaborata da un modello linguistico di grandi dimensioni con il compito di estrarre i singoli problemi dal flusso dei messaggi: individuare dove un problema comincia, quali messaggi lo riguardano e come si conclude, e riscriverlo in forma autocontenuta.

La scelta di procedere per anno, invece che sull'intero corpus o su finestre più piccole, risponde a due esigenze opposte: la porzione deve essere abbastanza ampia da permettere al modello di riconoscere che due segnalazioni distanti fra loro riguardano lo stesso problema, e abbastanza contenuta da rientrare nella finestra di contesto disponibile.

Per questa fase e per la successiva sono stati confrontati due modelli — Claude Opus 4.6 e Gemini in modalità *deep research* — sulle schede finali, e il confronto, riportato nella sezione 6.5, ha indicato il primo. Il prompt impiegato è riportato nell'Appendice B.2.2 (Codice B.3).

5.1.4 Deduplicazione

I file prodotti — uno per anno — vengono poi sottoposti, con lo stesso modello, a una seconda elaborazione, con il compito di deduplicare i problemi simili. Lo stesso problema ricorre infatti a distanza di mesi o di anni, formulato in modo diverso dalle persone coinvolte; indicizzarne più copie significherebbe occupare il contesto con chunk ridondanti e peggiorare la diversità dei risultati recuperati.

Il criterio di somiglianza è esplicitato nel prompt: due problemi vanno accorpati se condividono almeno due di tre elementi — lo stesso sistema o componente coinvolto, la stessa causa, la stessa soluzione. Nei casi dubbi le voci restano separate e vengono collegate da una nota di possibile correlazione, così da preferire una scheda in più a una fusione errata. Il prompt completo è riportato nell'Appendice B.2.2 (Codice B.4).

L'incidenza di questo passaggio è notevole: dai 482 problemi estratti dal passaggio precedente si scende a 109 schede distinte. Poco meno di quattro problemi estratti su cinque erano quindi il duplicato di un altro: è il dato da cui parte il problema descritto nella sezione 3.2.

Ciascuna delle 109 schede è strutturata secondo lo schema seguente:

- **titolo**, che identifica il problema;
- **occorrenze principali**, l'elenco delle date, raggruppate per anno, in cui il problema si è manifestato;
- **descrizione generale**, che riassume i sintomi osservati e l'impatto operativo;
- **causa ricorrente**, quando è stata individuata;
- **soluzione o workaround consolidato**;

- **frequenza**, una stima qualitativa e quantitativa delle occorrenze;
- **note**, con le osservazioni ricorrenti e i collegamenti ad altri problemi.

Sono queste schede, e non i messaggi originali, a essere indicizzate, ciascuna come un unico chunk (sezione 5.3.3). La trasformazione è sostanziale: da un flusso conversazionale in cui un problema è disperso fra decine di messaggi distribuiti su anni, si passa a una scheda autocontenuta che ne raccoglie storia, causa e rimedio.

Il Codice 5.2 riporta una scheda in forma abbreviata — sono omessi l'elenco delle occorrenze datate e la stima di frequenza — relativa a un problema diverso da quello dell'estratto riportato sopra: i due esempi sono indipendenti e servono a illustrare i due estremi della trasformazione, non a seguire un singolo caso lungo la pipeline.

Codice 5.2: Esempio di scheda di problema prodotta dalla pipeline.

```
Titolo: Pick Station 7 - pallet vuoto non arriva,
        "wait until work available"

Descrizione generale:
  Pick station 7 resta in stato "wait until the work will be available"
  anche dopo il riavvio. Impatto operativo: stazione bloccata.

Causa ricorrente:
  Richiesta di pallet vuoto non assegnata.

Soluzione / Workaround consolidato:
  Creazione di una nuova richiesta di pallet vuoto. In alcuni casi il
  problema si è risolto autonomamente.

Note: --
```

Accostare le due forme — il flusso di messaggi del Codice 5.1 e la scheda del Codice 5.2 — rende visibile il salto compiuto dalla pipeline. Da una parte una discussione in cui il problema è enunciato, ripreso a intervalli irregolari e intrecciato a messaggi di altro tenore; dall'altra un'unità informativa autocontenuta che dichiara sintomo, causa e rimedio, ed è quindi utilizzabile come risposta a una domanda.

5.1.5 Approcci alternativi valutati

La soluzione descritta non è stata la prima tentata. L'approccio iniziale, eseguito interamente in locale, faceva scorrere sul flusso dei messaggi una finestra di alcune decine di messaggi per volta, numerati in modo assoluto. A ogni chiamata il modello doveva restituire, in forma strutturata, tre informazioni: se nella finestra fosse presente un problema tecnico, la sua descrizione autocontenuta, e l'indice del primo messaggio che *non* apparteneva più a quel problema. Quest'ultimo valore serviva a riposizionare la finestra per la chiamata successiva, così da procedere problema per problema invece che a blocchi fissi.

Il metodo si è rivelato inadeguato: il modello non riusciva a cogliere la struttura dei problemi né a individuarne la fine. Il sintomo più evidente riguardava proprio l'indice di ripartenza, che il modello tendeva a collocare in punti arbitrari o a non far avanzare affatto — al punto che l'implementazione aveva dovuto prevedere un meccanismo di sicurezza che forzava l'avanzamento della finestra quando l'indice restituito non progrediva. Una revisione manuale di alcuni problemi estratti ha confermato che i confini individuati erano in larga parte arbitrari: l'esito non è stato registrato in forma quantitativa, ma il quadro era netto, e la quasi totalità dei problemi estratti risultava sbagliata o incompleta. Anche un clustering temporale eseguito in locale non ha risolto la questione: i problemi risultavano frammentati, perché alcuni si sovrapponevano nel tempo, altri uscivano dal cluster di appartenenza e altri ancora finivano per condividerne uno con problemi non correlati.

La differenza fra i due approcci è strutturale. Il primo chiede al modello di segmentare un flusso di messaggi con una finestra locale, senza mai vedere il quadro d'insieme; il secondo gli dà in input un intero anno di conversazioni e gli chiede di estrarne le unità informative, sfruttando la finestra di contesto ampia per ricostruire relazioni fra messaggi distanti. È stato sufficiente a scartare l'approccio senza misurarlo oltre.

5.1.6 Gestione delle immagini nelle conversazioni

Le immagini allegate ai messaggi — in larga parte fotografie di schermate di errore e di situazioni sul campo — sono state escluse dal trattamento. La ragione è la loro difficoltà intrinseca: a differenza delle illustrazioni della documentazione, che sono schemi o acquisizioni pulite dell'interfaccia, si

tratta di immagini scattate in condizioni non controllate, spesso parziali o poco leggibili, il cui significato dipende dal messaggio che le accompagna. Il modello visione-linguaggio eseguibile in locale non produceva descrizioni sufficientemente affidabili da giustificarne l'inserimento nell'indice, e una descrizione imprecisa inserita fra i contenuti recuperabili è peggio di un'assenza.

È una rinuncia consapevole e una perdita di informazione reale: in alcuni casi la fotografia è l'unico contenuto informativo del messaggio. La questione è ripresa fra i limiti del lavoro nella sezione 7.2.

5.2 Preprocessing dei documenti

5.2.1 Estrazione della struttura

I documenti vengono partizionati da Docling [6], che ne individua le tre componenti — testo, immagini e tabelle — e ne ricostruisce la struttura logica. Il risultato di questa fase non è ancora il testo da indicizzare, ma una rappresentazione intermedia in cui ogni elemento è tipizzato, porta con sé la propria provenienza ed è collocato nell'ordine di lettura corretto, anche quando il layout è a più colonne o intervallato da figure. La configurazione adottata è riportata nel Codice 5.3.

Codice 5.3: Configurazione della pipeline di parsing.

```
pipeline_options = PdfPipelineOptions()
pipeline_options.do_table_structure = True
pipeline_options.table_structure_options.do_cell_matching = True
pipeline_options.do_ocr = True
pipeline_options.ocr_options = RapidOcrOptions(lang=["ita", "eng"])
pipeline_options.do_picture_description = True
pipeline_options.picture_description_options = PictureDescriptionVlmOptions(
    repo_id="Qwen/Qwen3-VL-8B-Instruct-FP8",
    prompt=superprompt,
)
pipeline_options.images_scale = 2.0
pipeline_options.accelerator_options = AcceleratorOptions(
    num_threads=6, device=AcceleratorDevice.AUTO
)
```

Ogni opzione attivata risponde a una caratteristica del corpus. Il riconoscimento della struttura delle tabelle, con l'associazione delle celle riconosciute al testo estratto dal documento, serve perché la documentazione è ricca di tabelle di codici la cui relazione riga-colonna va preservata. L'OCR è configurato per italiano e inglese insieme, coerentemente con il bilinguismo della documentazione tecnica; poiché tutti i documenti del corpus sono nativamente digitali (sezione 3.3), interviene però in misura marginale, e la qualità dell'estrazione testuale non è stata un fattore limitante. La descrizione delle immagini è delegata al modello visione-linguaggio, con il prompt discusso di seguito, e il fattore di scala delle immagini è portato a 2, così che le schermate del gestionale siano rese al modello con risoluzione sufficiente a leggerne il testo.

L'uscita del parsing è duplice: una rappresentazione JSON completa, che conserva la struttura e la provenienza di ogni elemento, e una versione Markdown, che è la forma su cui opera la suddivisione in chunk.

5.2.2 Da immagine a descrizione testuale

Ogni immagine viene passata a Qwen3-VL 8B, che ne produce una descrizione testuale destinata a prenderne il posto nel documento, secondo la scelta di ricondurre il visuale al testo motivata nella sezione 4.3.1. È anche lo stadio più costoso dell'ingestion (sezione 4.1.2).

Il prompt fornito al modello è riportato nel Codice 5.4.

Codice 5.4: Prompt per la descrizione delle immagini.

```
Descrivi questa immagine in circa 3 frasi in modo diretto e concreto,
concentrandoti su elementi visibili, azioni, funzioni o informazioni
presenti. Evita frasi generiche come 'l'immagine mostra', 'si vede',
'c'è', ecc. Usa frasi informative che possano essere immediatamente
utili per un sistema di recupero basato su contenuti (RAG).
```

Due scelte nel prompt meritano di essere commentate, perché non sono stilistiche ma funzionali al recupero.

La prima è il divieto esplicito delle formule introduttive del tipo «l'immagine mostra». Non è una questione di eleganza: quelle formule, identiche in ogni descrizione del corpus, non portano

informazione e rendono più simili fra loro i vettori di descrizioni diverse. Eliminandole, il vettore della descrizione riflette il contenuto invece di una struttura ricorrente.

La seconda è la richiesta esplicita di frasi «immediatamente utili per un sistema di recupero»: il modello non deve produrre una didascalia per un lettore umano, ma un testo che verrà confrontato con domande operative. La lunghezza è vincolata a circa tre frasi, un compromesso fra il dettaglio necessario e il rischio che descrizioni prolisse diluiscano il contenuto del chunk in cui vengono inserite.

La Figura 5.2 riporta un esempio rappresentativo: una schermata del gestionale, che è la tipologia di immagine più frequente nel corpus.

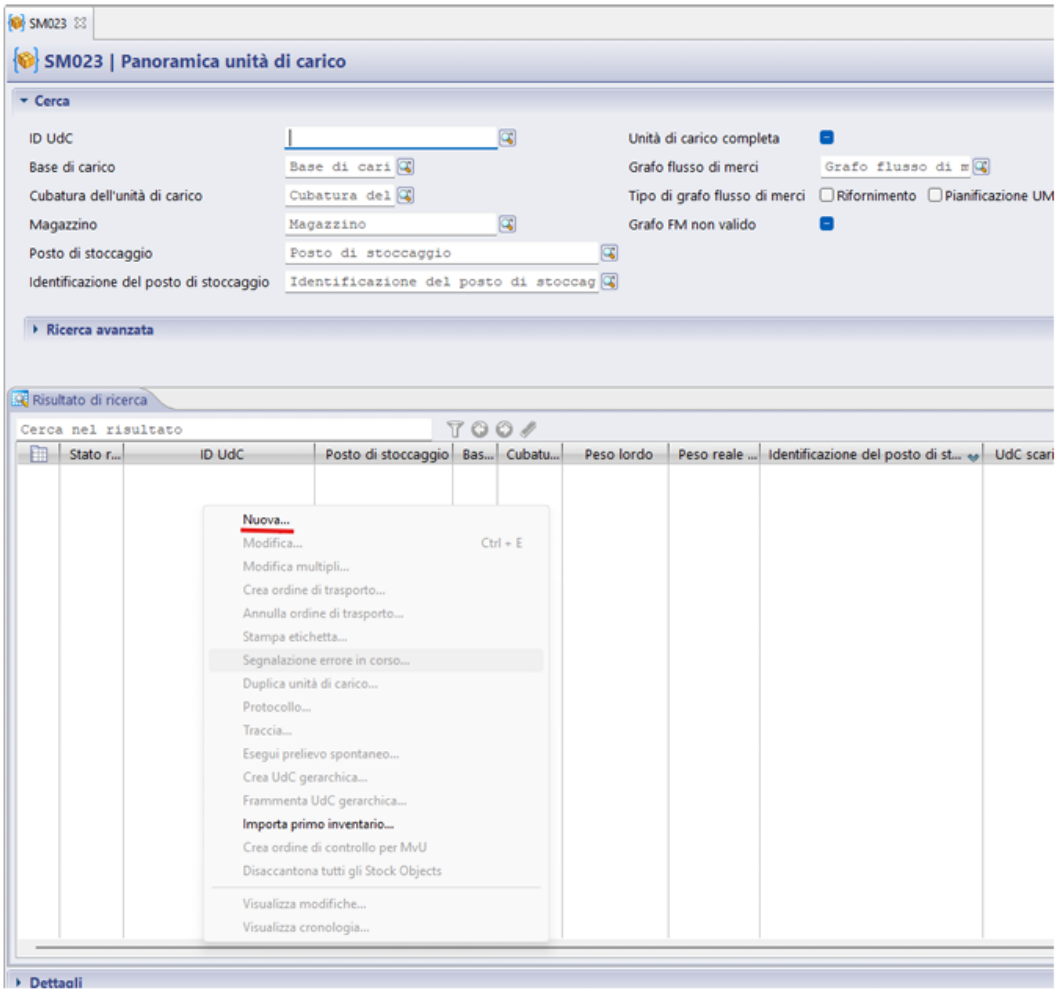


Figura 5.2: Acquisizione dell'interfaccia del gestionale: il modulo di panoramica delle unità di carico, con il menu contestuale aperto.

La descrizione prodotta dal modello per questa immagine è la seguente:

Codice 5.5: Descrizione generata per la Figura 5.2.

```
Interfaccia del modulo SM023 per la gestione delle unità di carico in un
sistema ERP, con un campo di ricerca e filtri per ID UDC, base di carico,
magazzino e posto di stoccaggio. È evidenziato il menu contestuale
"Nuova..." che offre opzioni per creare nuove unità di carico,
modificare, annullare ordini di trasporto, stampare etichette o eseguire
prelievi spontanei. La sezione "Risultato di ricerca" è vuota, indicando
che nessun record è stato trovato con i criteri attuali.
```

L'esempio mostra bene che cosa questa fase aggiunge e che cosa non può aggiungere. Il modello identifica il modulo per sigla, elenca i campi di ricerca e — soprattutto — riporta, parafrasandole, alcune voci del menu contestuale: sono proprio i termini su cui un utente formulerebbe una domanda («come creo una nuova unità di carico?»), e senza questa conversione resterebbero inaccessibili al recupero, perché presenti solo come pixel.

Il modello riporta però anche un dettaglio contingente e privo di valore informativo — che la sezione dei risultati è vuota — che finisce anch'esso nell'indice. È un rumore tollerabile, ma illustra il limite della conversione in testo: ciò che viene descritto è quello che il modello ritiene saliente, non necessariamente quello che sarà utile.

5.2.3 Da tabella a Markdown

Le tabelle vengono convertite in tabelle Markdown, evitando i due estremi discussi nella sezione 2.7.5: la corrispondenza fra intestazioni e valori resta intatta, e codici e sigle restano ricercabili lessicalmente, cosa particolarmente rilevante in questo dominio.

Il caso che ha dato i problemi maggiori è quello delle tabelle che attraversano più pagine. Il parser tratta ciascuna pagina come un'unità e interpreta la prima riga della continuazione come una nuova riga di intestazione, producendo due tabelle distinte di cui la seconda ha per intestazione quella che in realtà è una riga di dati. Il risultato è un chunk in cui i valori risultano associati a etichette sbagliate.

Nel corpus le tabelle in questa condizione sono poche, e l'impatto complessivo è quindi contenu-

to; resta però un limite noto della fase di parsing, e un candidato naturale a un controllo automatico in fase di ingestion.

5.2.4 Composizione e arricchimento

Le tre componenti vengono ricomposte in un unico documento Markdown. Prima della ricomposizione, le descrizioni prodotte dal modello visione-linguaggio vengono racchiuse fra due marcatori espliciti, `[IMAGE]` e `[/IMAGE]`. L'accorgimento ha due effetti: nel testo indicizzato la descrizione resta distinguibile dal testo originale del documento, e nel contesto passato al generatore è riconoscibile come resa testuale di un contenuto visivo, non come un'affermazione del documento. I marcatori di posizione lasciati dal parser al posto delle immagini vengono invece rimossi, perché non portano informazione.

Contestualmente vengono valorizzati i metadati della sezione 4.4.3: il nome e il formato del file sono letti dall'intestazione della rappresentazione prodotta da Docling, il gruppo dal prefisso del nome del file. Ogni chunk ricavato dal documento li eredita.

5.3 Suddivisione in chunk

5.3.1 Strategia adottata

Il documento Markdown viene suddiviso in chunk di al più 1000 token, con una sovrapposizione di al più 200 token fra chunk consecutivi:

Codice 5.6: Suddivisione del documento in chunk.

```
splitter = SentenceSplitter(chunk_size=1000, chunk_overlap=200)
nodes = splitter.get_nodes_from_documents([llama_doc])
```

È una dimensione ampia — vicina ai valori predefiniti del suddivisore e sensibilmente maggiore di quella dei chunk su cui operano le alternative considerate (512 token per Docling, 128 per il livello di ricerca della strategia gerarchica) — e la ragione per mantenerla sta nella natura del corpus: le unità informative della documentazione tecnica — una procedura, la descrizione di una maschera

del gestionale, una tabella di codici con il testo che la commenta — non sono brevi, e spezzarle produrrebbe chunk che, recuperati isolatamente, non bastano a rispondere.

I due valori vanno letti come limiti superiori, non come dimensioni effettive. Lo `SentenceSplitter` non taglia a 1000 token contati: cerca il confine di frase più vicino entro quella soglia e chiude lì il chunk, che risulta quindi spesso più corto. Lo stesso vale per la sovrapposizione, che raramente arriva ai 200 token dichiarati e in diversi casi non c'è affatto, perché il suddivisore la applica solo quando può farlo senza spezzare una frase. I chunk prodotti rispettano quindi la punteggiatura del documento, e la soglia in token agisce come vincolo di dimensione massima piuttosto che come criterio di taglio.

La sovrapposizione, dove è presente, attenua il problema residuo dei confini: un'informazione breve che cade a cavallo fra due chunk può comparire per intero in almeno uno dei due. Il costo è una ridondanza pari alla sovrapposizione nell'indice, che a queste dimensioni di corpus è del tutto sostenibile.

La scelta è stata verificata sperimentalmente contro le alternative descritte di seguito.

5.3.2 Alternative valutate

HierarchicalNodeParser di LlamaIndex.

Produce chunk a più livelli di granularità annidati l'uno nell'altro. I livelli configurati sono `chunk_sizes=[2048, 512, 128]`: ogni chunk da 2048 token contiene circa quattro chunk da 512, ciascuno dei quali ne contiene circa quattro da 128. L'idea è di effettuare il recupero sul livello fine, dove la corrispondenza semantica è più precisa perché il chunk è breve e poco diluito, e di risalire al blocco di livello superiore quando una parte sufficiente dei chunk fini che contiene viene recuperata, riportando il contesto che il chunk fine da solo non ha.

HybridChunker di Docling.

Suddivide seguendo la struttura del documento riconosciuta in fase di parsing, così che i confini dei chunk coincidano con confini logici invece che con una soglia di token. Il vincolo di dimensione massima è quello predefinito, 512 token, ma i chunk prodotti sono spesso più corti: il chunker preferisce chiudere il chunk alla fine di una frase o di un elemento strutturale piuttosto che riempire il budget disponibile. Il comportamento è notevolmente accurato sui confini — non spezza una frase

a metà, non separa l'intestazione di una tabella dal suo contenuto — ma è proprio questa fedeltà alla struttura a produrre il suo limite nel dominio in esame: una procedura operativa occupa quasi sempre più di 512 token, e viene quindi distribuita su chunk distinti, ciascuno dei quali contiene alcuni passi e non la procedura intera. Sulle domande che riguardano una procedura completa — frequenti fra le richieste degli utenti — il recupero restituisce un pezzo della risposta e il modello non ha modo di ricostruire il resto.

Le due alternative sono state confrontate con la strategia adottata sulla stessa griglia sperimentale; i risultati sono nella sezione 6.3. Va anticipato che le misure di richiamo relative alla strategia gerarchica vanno lette come una stima per difetto, per la ragione metodologica discussa nella sezione 6.2. Nel Capitolo 6 le tre strategie sono indicate per brevità come *Overlap* (quella adottata), *Hierarchical* e *Docling*.

5.3.3 Le schede di problema: un chunk per scheda

Le schede prodotte dalla pipeline delle conversazioni (sezione 5.1) non passano affatto per il suddivisore. La ragione è che non ne hanno bisogno: la struttura della scheda fornisce già i confini che la documentazione non ha, e ciascuna scheda viene indicizzata così com'è, come un unico chunk. Nessuna delle schede prodotte — nemmeno dopo la deduplicazione, che accorpa nella stessa scheda le diverse occorrenze di uno stesso problema — supera i 1000 token della strategia adottata per i documenti.

È una differenza che conviene sottolineare, perché mostra il senso di tutto il lavoro della sezione 5.1. Sulla documentazione la suddivisione funziona bene, ma resta un'operazione a posteriori su un testo che non è stato scritto per essere spezzato: i confini che il suddivisore trova sono confini linguistici, non le articolazioni del contenuto. Sulle conversazioni il problema è stato invece risolto a monte, trasformando un flusso senza struttura in unità che *sono già* la granularità giusta per il recupero. Il chunk coincide con il problema, e un problema è esattamente ciò di cui un utente va in cerca.

5.4 Generazione degli embedding

Le rappresentazioni dense dei chunk sono calcolate con BGE-M3. La scelta risponde a due caratteristiche del modello che corrispondono ad altrettante esigenze del progetto: è multilingue, e il corpus è in italiano con terminologia tecnica in inglese; e gestisce sequenze lunghe, coerentemente con la dimensione dei chunk adottata. La dimensione dei vettori, 1024, è quella nativa del modello e non una scelta di progetto. Accanto al vettore denso, a ogni chunk è associata una rappresentazione sparsa BM25, che alimenta il canale lessicale del recupero ibrido; le due rappresentazioni convivono nella stessa collection, configurata come descritto nella sezione 4.4.4.

L'occupazione dell'indice e i tempi di indicizzazione non sono stati misurati sistematicamente, perché sul corpus in esame non sono la grandezza che determina il costo dell'ingestion: il calcolo degli embedding e la scrittura su Qdrant sono operazioni rapide, e il tempo complessivo è dominato dalla descrizione delle immagini.

5.5 Recupero

Data la domanda dell'utente, la ricerca viene eseguita in modalità ibrida sui due spazi vettoriali della collection (sezione 4.4.4). Ciascuno dei due canali restituisce $k_1 = 17$ candidati; le due graduatorie vengono fuse, e la graduatoria risultante, troncata anch'essa a k_1, è quella passata al reranker.

5.5.1 Recupero denso

Il canale denso confronta il vettore della domanda con quelli dei chunk secondo la similarità del coseno, con la ricerca approssimata sull'indice HNSW e i parametri predefiniti di Qdrant (sezione 4.4.4).

È il canale su cui poggia la maggior parte del recupero, perché le domande degli utenti sono formulate in linguaggio naturale e spesso non riprendono alla lettera il lessico del manuale: chi segnala che «il pallet risulta in una posizione sbagliata» non usa il termine con cui la procedura descrive il problema, e la corrispondenza va trovata sul significato.

5.5.2 Recupero sparso

Il canale lessicale usa le rappresentazioni sparse BM25, anche qui con i parametri di ricerca predefiniti di Qdrant.

Il suo contributo è complementare a quello denso e si concentra sui casi in cui conta la corrispondenza esatta del termine più della vicinanza semantica: i codici delle maschere del gestionale, le sigle interne, i codici di errore. Sono stringhe per cui la nozione di «vicino nel significato» è poco informativa — due codici simili non indicano due procedure simili — e su cui la corrispondenza letterale è invece decisiva.

5.5.3 Recupero ibrido

Il recupero ibrido è realizzato con due retriever distinti, uno per il canale denso e uno per quello lessicale, le cui graduatorie vengono combinate con la *Reciprocal Rank Fusion* descritta dall'Equazione 2.5, come mostrato nel Codice 5.7. La fusione è quella implementata da LlamaIndex, che adotta la costante $k = 60$ del lavoro originale; la domanda non viene riformulata prima della ricerca, coerentemente con la scelta discussa nella sezione 5.10.1.

La scelta di una fusione basata sulle posizioni, e non sui punteggi, è quella naturale in questo contesto: i punteggi dei due canali non sono commensurabili — la similarità del coseno vive in un intervallo fisso, il punteggio lessicale no — e qualunque combinazione diretta richiederebbe una normalizzazione che dipende dalla distribuzione dei punteggi stessi. La RRF, lavorando sui soli ranghi, evita il problema e non introduce parametri da tarare sulla collezione.

Un secondo aspetto va sottolineato, perché riguarda una garanzia e non una preferenza: il filtro sui permessi descritto nella sezione 5.6 viene passato a entrambi i retriever e applicato dal database a ciascuna delle due interrogazioni nello stesso modo. La fusione lavora quindi su due liste già filtrate. Applicarlo a un solo canale produrrebbe graduatorie costruite su insiemi di documenti diversi, e reintrodurrebbe dal canale non filtrato esattamente i contenuti che si vogliono escludere; così com'è, invece, non esiste alcun percorso per cui un chunk non autorizzato possa entrare nella graduatoria finale.

5.5.4 Dimensionamento di k_1

La scelta di k_1 è il risultato del compromesso fra copertura e costo: un valore più alto aumenta la probabilità che il chunk corretto sia fra i candidati, ma allunga il tempo di riordinamento, che cresce linearmente con il numero di coppie query-documento valutate dal reranker.

Il valore $k_1 = 17$ è quello che, nella valutazione del Capitolo 6, massimizza, fra i valori provati, l'accuratezza end-to-end della strategia adottata, ed è stato portato in produzione invariato. Le misure della sezione 6.2 indicano che il richiamo si avvicinerebbe al massimo anche con un valore più basso; il valore più alto non ha però un costo apprezzabile in latenza, e offre un margine che assorbe in parte la differenza fra lo spazio di ricerca della valutazione e quello, più ampio, dell'indice in produzione — una cautela discussa nella sezione 6.8.

5.6 Recupero filtrato per permessi

È qui che il modello di autorizzazione descritto nella sezione 4.5.1 — gruppi, mappa delle visibilità, pre-filtering — diventa un'operazione concreta della pipeline.

5.6.1 Dalla sessione utente al filtro

L'utente si autentica tramite Chainlit; dalla sessione si risale al gruppo di appartenenza e, tramite la mappa gruppo–gruppi accessibili, all'insieme dei gruppi di documenti che può consultare. Da questo insieme viene costruita la condizione di filtro che accompagna la query.

Il filtro è derivato lato server dalla sessione autenticata e non da parametri controllabili dal client: la distinzione è essenziale, perché un filtro trasmesso dal client sarebbe manipolabile e il meccanismo non offrirebbe alcuna garanzia reale.

5.6.2 Applicazione del filtro nella ricerca

Il Codice 5.7 mostra la costruzione del filtro e la sua applicazione ai due retriever del recupero ibrido: la stessa condizione accompagna entrambe le interrogazioni, per la ragione discussa nella sezione 5.5.3.

Codice 5.7: Costruzione del filtro sui permessi e recupero ibrido filtrato con fusione RRF.

```
filters = MetadataFilters(
    filters=[
        MetadataFilter(
            key="groups",
            value=gruppi_accessibili,
            operator=FilterOperator.ANY,
        )
    ]
)

retriever_dense = index.as_retriever(
    vector_store_query_mode=VectorStoreQueryMode.DEFAULT,
    similarity_top_k=topk,
    filters=filters,
)
retriever_sparse = index.as_retriever(
    vector_store_query_mode=VectorStoreQueryMode.SPARSE,
    sparse_top_k=topk,
    filters=filters,
)

retriever = QueryFusionRetriever(
    retrievers=[retriever_dense, retriever_sparse],
    similarity_top_k=topk,
    num_queries=1,               # nessuna riformulazione della domanda
    mode="reciprocal_rerank",    # Reciprocal Rank Fusion
)
```

La condizione verifica che il campo `groups` del chunk — la sigla del gruppo proprietario del documento — sia compreso nella lista dei gruppi accessibili all'utente, ricavata dalla mappa della Tabella 4.4. Un utente `MAN`, per esempio, porta con sé la lista `[MAN, LOG, PUB]`, e il filtro lascia passare i soli chunk il cui gruppo proprietario è uno di quei tre. In Qdrant la condizione si traduce nel predicato `MatchAny`, soddisfatto quando il valore memorizzato compare fra quelli elencati.

La Tabella 5.1 illustra l'esito su alcuni casi.

Tabella 5.1: Esito del filtro per due profili di utente. La lista cercata è quella dei gruppi accessibili; il valore memorizzato è il gruppo proprietario del documento.

Gruppo del documento	**Utente `MAN`** `[MAN, LOG, PUB]`	**Utente `LOG`** `[LOG, PUB]`
PUB	✓	✓
LOG	✓	✓
MAN	✓	✗
UTL	✗	✗

5.6.3 Comportamento in assenza di documenti autorizzati

Resta da descrivere il caso in cui un utente ponga una domanda pertinente su documenti che non è autorizzato a vedere. È il caso che mette alla prova il meccanismo, e va detto con precisione che cosa succede.

Il filtro agisce prima del recupero, quindi i chunk che contengono la risposta non entrano nella graduatoria. La ricerca però non restituisce un insieme vuoto: restituisce i chunk più vicini alla domanda *fra quelli visibili*, che saranno quasi certamente non pertinenti — sono semplicemente i meno lontani di un insieme in cui la risposta non c'è. Il contesto costruito per il modello generativo è quindi popolato, ma non contiene ciò che servirebbe.

Da qui il comportamento si biforca. Nel caso desiderabile il modello riconosce che il contesto non risponde alla domanda e applica la seconda regola del prompt di sistema, dichiarando di non avere informazioni sufficienti (sezione 5.8.3). Nel caso peggiore il modello prova ugualmente a rispondere a partire da chunk che non c'entrano, e produce un'allucinazione. L'istruzione del prompt di sistema spinge verso l'astensione ma non la garantisce, e questo resta il caso in cui il rischio è più alto: a differenza di una domanda su materiale semplicemente assente dal corpus, qui la domanda è *ben posta* — la risposta esiste, da qualche parte nell'indice — e quindi il modello è sollecitato a produrla.

Vale però la pena osservare che sul piano della riservatezza, che è l'obiettivo del meccanismo, la garanzia regge in modo netto. Il modello non riceve alcuna traccia del documento riservato — non

il titolo, non un chunk, non il fatto che esista — perché il filtro ha agito a monte, nel database. Non c'è quindi alcun modo in cui la risposta possa rivelare l'esistenza di ciò che l'utente non può vedere: nemmeno la forma «esiste un documento ma non ti è accessibile», che sarebbe già di per sé una divulgazione. Il prezzo di questa garanzia è che il sistema non sa distinguere fra «l'informazione non esiste» e «l'informazione esiste ma non ti è visibile», e all'utente le due situazioni appaiono identiche.

5.6.4 Costo del filtraggio

Il costo del filtraggio non è stato misurato, perché non si è mai presentato come un problema da indagare: il recupero, filtro compreso, è di fatto istantaneo rispetto alla generazione (Tabella 5.2), e ridurlo non avrebbe alcun effetto percepibile sull'esperienza d'uso.

Resta vero, sul piano teorico, che il pre-filtering può degradare l'efficienza della visita di un indice approssimato quando la porzione visibile della collection è molto piccola, per le ragioni discusse nella sezione 2.6.2. È una considerazione che meriterebbe di essere ripresa su un indice di dimensioni sensibilmente maggiori; su quello in esame non è osservabile.

5.7 Riordinamento dei candidati

I 17 candidati recuperati vengono riordinati da BGE-Reranker-v2-M3, esposto come servizio HTTP dedicato, che ne seleziona i $k_2 = 7$ migliori. Il reranker è un cross-encoder: valuta la coppia query-chunk nel suo insieme invece di confrontare due rappresentazioni calcolate separatamente, ed è quindi più accurato del recupero ma più costoso, motivo per cui opera su pochi candidati e non sull'intero indice.

La scelta di un reranker multilingue della stessa famiglia del modello di embedding non è casuale: favorisce un comportamento omogeneo sulla terminologia mista italiano-inglese del corpus, e resta compatibile con il budget di memoria della sezione 4.8.

5.7.1 Dimensionamento di k_2

Il valore $k_2 = 7$ nasce dal vincolo sulla finestra di contesto. I chunk hanno una dimensione massima di 1000 token, ma come osservato nella sezione 5.3 sono spesso più corti, perché il suddivisore li chiude sul confine di frase: sette chunk occupano quindi tipicamente intorno ai 6000 token dei 12 288 disponibili, e il resto deve bastare per il prompt di sistema, la cronologia della conversazione e la risposta da generare.

In fase di valutazione il parametro è stato trattato non come un numero fisso di chunk ma come un *budget di token*: il reranker seleziona i chunk in ordine di punteggio finché il budget è esaurito. È la formulazione corretta quando i chunk hanno dimensioni diverse, ed è quella su cui si basano i risultati della sezione 6.3, che individuano nell'intervallo fra 4096 e 6144 token la regione in cui la qualità della risposta è massima. Il valore adottato in produzione — sette chunk, circa 6000 token — corrisponde al punto migliore della strategia adottata, il budget di 6144 token con $k_1 = 17$ (Tabella 6.3).

5.8 Costruzione del contesto

5.8.1 Selezione e ordinamento dei chunk

I sette chunk selezionati dal reranker vengono inseriti nel prompt di sistema, al posto del segnaposto `{context}` (Appendice B.1).

Ogni chunk viene preceduto da un'intestazione che ne dichiara il documento di provenienza. L'intestazione ha una funzione precisa: rende il contesto *attribuibile*, cioè permette al modello di citare la fonte di ciascuna informazione invece di fondere tutti i chunk in un testo indistinto.

L'ordine è quello stabilito dal reranker: dal chunk giudicato più rilevante al meno rilevante. Non viene applicata alcuna ricomposizione — i chunk provenienti dallo stesso documento non vengono riavvicinati fra loro, né si tenta di ricostruirne la sequenza originale.

La scelta discende dalla dimensione dei chunk. Con chunk fino a 1000 token la risposta si trova spesso per intero dentro uno di essi, invece di doversi ricomporre attraversandone più d'uno: rimetterli in ordine di documento aggiungerebbe poco, perché di norma non c'è una continuità da

ricostituire. È coerente con quanto emerge nel Capitolo 6, dove il vantaggio della strategia a chunk ampi sta proprio nella sufficienza del singolo chunk recuperato.

5.8.2 Cronologia della conversazione

Se la conversazione ha dei turni precedenti, vengono recuperate al massimo tre interazioni fra utente e modello, fino a un limite di 1000 caratteri, e passate al modello insieme alla domanda corrente.

La cronologia viene fornita senza il contesto documentale dei turni precedenti. Tutti e tre i limiti — il numero di interazioni, la lunghezza in caratteri e l'esclusione dei chunk già recuperati — rispondono allo stesso vincolo, la finestra di contesto. I sette chunk della domanda corrente ne occupano già circa la metà (sezione 5.7.1), e il resto deve bastare per il prompt di sistema e per la risposta: se la cronologia portasse con sé anche i chunk dei turni precedenti, basterebbe una sola domanda di approfondimento per sforare i 12 288 token disponibili.

5.8.3 Prompt di sistema

Il prompt di sistema incorpora i chunk selezionati, delimitati da un tag `<context>`, e detta cinque regole di comportamento:

1. usare soltanto il contesto fornito nei tag `<context>`;
2. quando l'informazione non è presente, rispondere con una formula fissa che lo dichiari, invece di tentare una risposta;
3. non attingere a conoscenze esterne al contesto;
4. mantenere un tono tecnico e asciutto;
5. citare il nome del documento e, quando il chunk lo riporta, la sezione o il capitolo.

La seconda regola è quella che incide di più sul comportamento del sistema: impone un'astensione esplicita al posto di una risposta inventata, e rende l'astensione riconoscibile anche in fase di valutazione, dove viene contata come categoria a sé (sezione 6.1.8).

Il prompt completo è riportato nell'Appendice B.1.

5.9 Generazione della risposta

5.9.1 Modello generativo

La risposta viene generata da Qwen3 32B [32] in quantizzazione AWQ, servito da vLLM con i parametri riportati nella sezione 4.8: è la quantizzazione (sezione 2.1.5) a farlo rientrare nel budget di memoria.

I parametri di decodifica riflettono la priorità del sistema, che è la fedeltà al contesto e non la varietà della formulazione: temperatura 0,2, lunghezza massima della risposta 1024 token, generazione in streaming così che l'utente veda la risposta formarsi invece di attendere il testo completo. La modalità di ragionamento esplicito non è né imposta né disattivata: è il modello a decidere se ragionare prima di rispondere. Quando lo fa, il ragionamento allunga i tempi di risposta e consuma token della finestra di contesto.

5.9.2 Citazioni e tracciabilità

Al termine della risposta il sistema elenca, senza ripetizioni, i nomi dei documenti da cui provengono i chunk passati al modello. È il meccanismo che rende la risposta verificabile: l'utente può risalire alla fonte e controllare, e ha a disposizione un riferimento compatto da consultare invece di una ripetizione delle intestazioni presenti nel contesto. La granularità della citazione è quindi il documento, non la posizione al suo interno: l'utente sa quale file aprire, ma deve poi cercarvi il passaggio. È un limite reale sui documenti più lunghi, e non è una scelta di progetto ma una conseguenza tecnica: il riferimento alla pagina era previsto, ma l'informazione di provenienza non sopravvive al passaggio fra il parsing e l'indicizzazione, per la ragione discussa nella sezione 4.4.3. Recuperarla è indicato fra gli sviluppi possibili nella sezione 7.3.2.

Quando il contesto non contiene la risposta, il modello restituisce invece la formula fissa prevista dalla seconda regola del prompt di sistema (sezione 5.8.3).

5.10 Scelte progettuali e alternative scartate

5.10.1 Assenza di trasformazione della query

La pipeline non effettua alcuna riformulazione della query: la domanda dell'utente viene usata così com'è per il recupero. È una scelta deliberata e sostenuta dalle prove condotte.

Le tecniche di trasformazione della query descritte nella sezione 2.5.4 — multi-query, HyDE, step-back prompting — sono state tutte provate: HyDE in modo sistematico, multi-query e step-back in via qualitativa. Il dominio su cui opera il sistema è troppo tecnico perché queste tecniche funzionino: il modello che riformula la domanda non conosce il lessico dell'azienda — nomi delle maschere, sigle, codici — e l'espansione introduce terminologia plausibile ma assente dai documenti, allontanando la domanda dal testo che dovrebbe trovare. La valutazione (sezione 6.4) mostra che per la configurazione adottata HyDE peggiora il richiamo, e che dove un miglioramento c'è è troppo contenuto per giustificare la chiamata aggiuntiva al modello che ogni tecnica richiede prima di ogni ricerca.

5.10.2 Architettura agentica

È stata sperimentata anche un'architettura agentica, in cui il modello decide autonomamente quando effettuare una ricerca, quali chunk trattenere e quando ha informazioni sufficienti per rispondere. L'esito è stato negativo, e la ragione è istruttiva.

Un modello da 32 miliardi di parametri quantizzato regge bene compiti circoscritti a un solo passo — generare varianti di una query, produrre un documento ipotetico — ma fatica quando gli strumenti da orchestrare diventano contemporaneamente la ricerca vettoriale, la selezione dei chunk rilevanti (obbligatoria, dato il contesto limitato) e la prosecuzione di una conversazione basata su chunk recuperati in precedenza. Lo sforzo è stato quindi concentrato dove il margine di miglioramento era reale — la qualità del preprocessing, dell'embedding e del riordinamento —, cioè nel fornire al modello i dati migliori possibili, invece di chiedere a un modello quantizzato di comportarsi da agente.

5.10.3 Indici basati su riassunti

Non sono state adottate strategie basate su riassunti o su alberi di sintesi del documento. Il motivo è la natura del compito: gli utenti non cercano una sintesi ma la procedura per filo e per segno, e un livello di riassunto tende a trascurare proprio i dettagli operativi che servono. Un chunk ampio recuperato integralmente contiene invece, di norma, le informazioni necessarie a rispondere. A questo si aggiunge il requisito della citazione delle fonti, che architetture di quel tipo complicano sensibilmente.

C'è poi una ragione empirica, emersa provando: il modello generativo locale non riassume in modo affidabile. Pur con 32 miliardi di parametri, sui testi procedurali del corpus produce sintesi che appiattiscono i passaggi operativi — proprio ciò che andrebbe conservato — e che spesso omettono i vincoli e le eccezioni, che in una procedura di magazzino sono la parte che conta. Un'architettura basata su riassunti avrebbe quindi fatto dipendere la qualità dell'indice dal punto debole del modello, invece che dal compito su cui è più affidabile, estrarre una risposta da un contesto già pertinente.

Va detto che la rinuncia è anche figlia del tempo disponibile: il tirocinio comprendeva la messa in produzione del sistema — autenticazione, interfaccia, gestione delle conversazioni — e le architetture più elaborate sono indicate fra gli sviluppi possibili nella sezione 7.3.5.

5.10.4 Coerenza con il vincolo di esecuzione locale

Le scelte progettuali adottate sono coerenti con il vincolo di esecuzione locale discusso nella sezione 4.1.3. In particolare, la rinuncia alla riformulazione della query, il dimensionamento di k_1 e k_2, la quantizzazione del modello generativo e l'istanziazione del modello visione-linguaggio soltanto in fase di ingestion non sono scelte di comodo: sono il modo in cui il budget di memoria e di tempo di GPU descritto nella sezione 4.8 è stato distribuito fra gli stadi che incidono di più sulla qualità della risposta.

La Tabella 5.2 riassume il costo in tempo dei singoli stadi, ed è il modo più diretto per vedere dove quel budget viene speso.

Tabella 5.2: Costo in tempo degli stadi della pipeline di risposta. I valori sono ordini di grandezza osservati in esercizio, non misure strumentate.

Stadio	**Tempo**
Embedding della domanda	istantaneo
Ricerca su Qdrant	istantaneo
Riordinamento (reranker)	istantaneo
Generazione della risposta	15–20 s, in streaming
Trasformazione della query (non adottata)	*5–10 s*

La lettura è netta: tutto ciò che precede la generazione è sostanzialmente gratuito, e il tempo di risposta coincide con il tempo del modello generativo. Lo streaming lo rende però accettabile, perché le prime parole compaiono molto prima della fine: l'utente non attende quindici secondi davanti a una schermata ferma, ma vede la risposta formarsi.

È anche la ragione per cui la trasformazione della query, oltre a non migliorare il recupero (sezione 6.4), sarebbe stata difficile da giustificare: cinque o dieci secondi — e sono quelli ottenibili disattivando il ragionamento del modello, altrimenti di più — sono un'attesa *prima* che cominci lo streaming, cioè nel punto esatto in cui l'utente non ha ancora nulla da leggere.

6. Risultati

Le scelte descritte nel Capitolo 5 — la strategia di suddivisione dei documenti, la modalità di recupero, la rinuncia alla trasformazione della query — non sono state prese per intuizione, ma sulla base di una campagna di misurazione costruita appositamente per questo dominio. Questo capitolo descrive come è stato costruito il dataset di valutazione, come sono stati misurati il recupero e la qualità delle risposte e che cosa i risultati dicono del sistema.

L'impostazione è quella consueta nella valutazione dei sistemi RAG: il recupero viene misurato a monte della generazione, e il sistema viene poi valutato end-to-end su una griglia di configurazioni. Conviene anticipare che alcuni risultati hanno portato a *semplificare* la pipeline invece che ad arricchirla.

6.1 Impostazione sperimentale

6.1.1 Il corpus di valutazione

La campagna è stata condotta su un sottoinsieme selezionato del corpus: quattordici documenti di procedura relativi al sistema di gestione del magazzino, scelti fra quelli il cui contenuto non è soggetto a riservatezza. Complessivamente il corpus di valutazione ammonta a circa 27 000 token.

La selezione ha un vantaggio metodologico e un limite, entrambi da dichiarare. Il vantaggio è che, trattandosi di documenti non riservati, la procedura di valutazione ha potuto avvalersi di modelli di grandi dimensioni accessibili come servizio: il vincolo discusso nella sezione 4.1.3 riguarda i contenuti riservati e il sistema in esercizio, e non è violato da una procedura svolta una tantum, fuori dall'esercizio, su documenti non riservati. Il limite è che le misure di richiamo sono relative a uno spazio di ricerca molto più piccolo di quello reale; la questione è ripresa nella sezione 6.8.

Da questi documenti è stato generato un insieme di 137 domande, circa una decina per documento. Le domande sono le stesse per tutte le strategie di suddivisione confrontate, per la ragione metodologica discussa più avanti in questa sezione.

6.1.2 Generazione delle domande

Per un corpus aziendale non esiste alcun dataset di valutazione: vanno costruite sia le domande sia i riferimenti alle risposte corrette. La generazione è stata affidata a Gemini 2.5 Pro, con un prompt che chiede esplicitamente di coprire tre profili di utente — l'operatore esperto, che vuole risolvere un problema puntuale; l'utente in formazione, che non conosce il sistema e chiede come funziona; l'amministratore, che deve impostare parametri o gestire eccezioni — e tre categorie di domanda: *single-hop* su un singolo punto del documento, *multi-hop* che collegano parti diverse, e domande di *onboarding* sui flussi generali.

Il prompt contiene anche vincoli di stile pensati per rendere le domande realistiche: nessuna formulazione da quiz scolastico, un'intenzione operativa dietro ogni domanda, un registro linguistico coerente con il profilo (l'esperto usa termini precisi, il novizio termini generici) e il divieto di suggerire nella domanda in quale parte del documento si trovi la risposta.

Il prompt completo è riportato nell'Appendice B (Codice B.9).

6.1.3 Supervisione con modello giudice

Ogni insieme di domande generato viene sottoposto a un modello giudice, che riceve il documento completo e la lista delle domande e valuta le domande nel loro insieme, non una per una. I criteri sono la rispondibilità a partire dal solo documento, l'assenza di ambiguità e di ridondanze, la copertura di parti diverse del documento e l'utilità per valutare un retriever.

Il giudizio non è un semplice filtro: quando il set viene respinto, il giudice produce un feedback che viene reimmesso nel prompt di generazione per l'iterazione successiva. Il criterio di accettazione è deliberatamente severo — basta una domanda mal formulata per invalidare l'intero set — e punta alla precisione più che alla numerosità. Il prompt del giudice è riportato nell'Appendice B (Codice B.10).

Al controllo automatico se ne aggiunge uno umano: prima di adottarle come dataset di valutazione, un campione delle domande accettate dal giudice è stato esaminato e valutato da colleghi del reparto logistico, le cui indicazioni hanno contribuito anche alla stesura del prompt di generazione.

6.1.4 Assegnazione dei chunk di riferimento

La costruzione della ground truth è separata dalla generazione delle domande, ed è una scelta metodologica che merita di essere motivata.

Nella prima versione della procedura, il modello riceveva il documento *già partizionato* e generava in un solo passaggio le domande e l'indicazione dei chunk che contenevano la risposta. L'approccio si è rivelato inadeguato al confronto fra strategie di suddivisione: poiché le domande venivano generate a partire dai chunk, ogni metodo di chunking produceva domande diverse, e i tre metodi finivano per essere valutati su dataset differenti.

La procedura è stata quindi spezzata in due fasi. Le domande vengono generate una sola volta a partire dal documento *intero*, e sono perciò identiche per tutti i metodi. In un secondo momento, per ciascuna strategia di suddivisione, a Gemini 2.5 Pro vengono sottoposti il documento partizionato e le domande già fissate, con il compito di indicare in quale chunk — o in quali, quando la risposta è distribuita — si trovi la risposta. Il prompt è riportato nell'Appendice B (Codice B.11).

A differenza della generazione, questa fase non passa per il modello giudice, e la scelta dipende dalla diversa natura dei due compiti. I criteri su cui il giudice lavora — ambiguità, ridondanza, copertura, utilità della domanda — riguardano un compito aperto, in cui il modello deve inventare qualcosa; l'assegnazione è invece un compito chiuso di localizzazione: le domande sono già state validate come rispondibili a partire dal documento, e il modello deve soltanto indicare, fra segmenti numerati, quelli che contengono la risposta, secondo regole esplicite (un solo chunk per le domande single-hop, l'insieme minimo e sufficiente per le multi-hop, una lista vuota se la domanda non risulta rispondibile), motivando la scelta prima di indicare gli identificativi. La parte verificabile senza un secondo modello è controllata in modo deterministico: gli identificativi restituiti vengono confrontati con quelli dei chunk effettivamente presenti, e quelli inesistenti scartati. L'assegnazione va inoltre ripetuta per ciascuna delle tre strategie, e un ciclo di supervisione ne avrebbe moltiplicato il costo. Il rischio residuo — un chunk di riferimento indicato in modo impreciso — è discusso fra i limiti della valutazione (sezione 6.8).

Il risultato è che il confronto fra le tre strategie avviene sulle stesse domande, e ciò che cambia è soltanto la mappatura fra domanda e chunk di riferimento. È questa separazione che rende i numeri delle sezioni seguenti direttamente confrontabili.

6.1.5 Output strutturato

Tutte le chiamate ai modelli della campagna — generazione delle domande, supervisione, assegnazione dei chunk e giudizio delle risposte — usano la funzione di output strutturato di LlamaIndex [23]: al modello viene fornito, insieme al prompt, lo schema dei campi attesi, e la risposta viene restituita come oggetto conforme a quello schema invece che come testo libero da interpretare. Il generatore restituisce così la lista delle domande, ciascuna preceduta da un breve scenario d'uso che simula il contesto dell'operatore; il giudice delle domande un ragionamento, un feedback e un verdetto booleano di validità, in quest'ordine; l'assegnazione, per ciascuna domanda, un ragionamento seguito dalla lista degli identificativi dei chunk che contengono la risposta; il giudizio delle risposte il valore 1, 0 o -1, preceduto da una lista di passaggi logici che analizzano la risposta.

In tutti e quattro i casi, quindi, lo schema obbliga il modello a produrre qualcosa di simile a un ragionamento prima dell'esito: il generatore si immedesima nell'utente prima di formulare la domanda, l'assegnazione motiva la scelta dei chunk prima di indicarli, e i due giudici esplicitano l'analisi prima del verdetto.

È ciò che rende la procedura interamente automatica: il ciclo fra generazione e supervisione si chiude sul campo di validità, e il conteggio dei giudizi non richiede di estrarre il verdetto da un testo scritto dal modello.

6.1.6 Organizzazione dei dati

Per rendere ripetibile la procedura, in fase di suddivisione è stata costruita una base dati con una tabella per ciascun metodo, contenente il nome del file, l'indice del chunk e il testo del chunk. A valle della generazione, un'ulteriore tabella per ciascun metodo raccoglie l'esito: identificativo, nome del file, indice o indici dei chunk rilevanti, testo della domanda e documento ipotetico generato per la valutazione di HyDE.

6.1.7 Metriche

Il recupero è misurato con il **richiamo cumulato**: scorrendo la graduatoria dei chunk restituiti, si registra a ogni posizione la frazione di chunk di riferimento già recuperati. Il richiamo viene

riportato in funzione dei **token cumulati** invece che del numero di chunk, perché sono i token la risorsa realmente scarsa data la finestra di contesto del modello generativo.

La misura in token è ciò che rende equo il confronto fra strategie che producono chunk di dimensione molto diversa: diciassette chunk da al più 1000 token occupano nel prompt molto più spazio di diciassette chunk da al più 128, e confrontare le strategie a parità di numero di chunk sarebbe quindi privo di significato.

La qualità complessiva è misurata con l'**accuratezza**: la frazione di risposte giudicate corrette sul totale delle domande.

6.1.8 Protocollo end-to-end

La valutazione complessiva esercita la pipeline nella sua interezza. Data una domanda, si ottiene la risposta del sistema; questa viene poi sottoposta a Gemini 2.5 Pro insieme alla domanda e alla ground truth ricostruita dai chunk di riferimento, con il compito di giudicarla.

Il giudizio è su tre livelli: 1 se la risposta è corretta, pertinente e coerente con i chunk di riferimento; 0 se è errata, fuorviante o incompleta rispetto alla ground truth; -1 se il modello dichiara esplicitamente di non avere informazioni sufficienti per rispondere. La distinzione fra 0 e -1 separa l'errore dall'astensione, ma l'accuratezza riportata nelle sezioni seguenti conta come corrette le sole risposte giudicate 1: un'astensione, dal punto di vista dell'utente, è una domanda a cui il sistema non ha risposto.

Il prompt non chiede un ragionamento, ma lo schema di output prevede, prima del verdetto, una lista di passaggi logici che analizzano la risposta. Va notato inoltre che il giudice non riceve il contesto recuperato dal sistema, ma soltanto la domanda, la ground truth e la risposta: valuta quindi la risposta rispetto ai chunk di riferimento, non rispetto a ciò che la pipeline ha trovato. Il prompt è riportato nell'Appendice B (Codice B.12).

Ogni valutazione viene registrata insieme ai parametri della configurazione che l'ha prodotta — domanda, tecnica di recupero, metodo di suddivisione, numero di candidati e budget di token — così da poter ricostruire a posteriori qualunque confronto.

6.1.9 Griglia sperimentale

La valutazione end-to-end è stata condotta come ricerca esaustiva su due parametri: il numero di candidati restituiti dal recupero e il budget di token che il reranker può riempire selezionando i chunk da passare al generatore. I valori del budget sono 1024, 4096 e 6144 token.

I valori di k non sono gli stessi per le tre strategie, ed è deliberato: sono stati scelti in modo che a ciascun livello corrisponda un volume di testo confrontabile fra i metodi, visto che i chunk hanno dimensioni molto diverse. La Tabella 6.1 riporta la corrispondenza.

Tabella 6.1: Griglia dei valori di k esplorati per ciascuna strategia di suddivisione. I tre livelli corrispondono a volumi di testo confrontabili, non allo stesso numero di chunk.

Strategia	**Livello basso**	**Livello medio**	**Livello alto**
Overlap (1000 token)	3	8	17
Hierarchical	14	52	90
Docling HybridChunker	10	30	50

6.1.10 Riproducibilità

Ogni confronto fa variare un solo fattore — strategia, k o budget — a parità degli altri due, e la griglia completa li incrocia tutti; le altre componenti della pipeline restano fisse. La pipeline esercitata è quella di produzione, con gli stessi modelli e la stessa macchina di inferenza descritta nella sezione 4.8.

Le temperature di campionamento dei modelli esterni sono state scelte in base al compito. Generazione e supervisione delle domande condividono la stessa configurazione, con temperatura 0,8: un valore alto favorisce la varietà delle domande e fa sì che, dopo un rifiuto del giudice, il nuovo insieme non riproponga le stesse formulazioni. L'assegnazione dei chunk usa invece una temperatura di 0,2 e il giudizio delle risposte di 0,1: sono compiti di localizzazione e di valutazione, in cui la variabilità non è un pregio e l'esito deve dipendere il meno possibile dal campionamento.

L'intera griglia — tre strategie di suddivisione, tre livelli di k, tre budget di token — comporta l'esecuzione della pipeline completa su ciascuna delle 137 domande per ognuna delle ventisette

configurazioni, seguita da altrettante chiamate al modello giudice: circa 3700 risposte generate e altrettanti giudizi.

Ogni configurazione è stata eseguita una volta sola, e le ragioni sono diverse per i due blocchi di misure.

Le misure di recupero della sezione 6.2 si fermano a monte della generazione: coinvolgono il modello di embedding e il reranker, entrambi eseguiti in locale, ed entrambi deterministici a parità di indice e di domanda. Ripeterle restituirebbe gli stessi valori, e la ripetizione non aggiungerebbe quindi informazione.

Le misure end-to-end della sezione 6.3 sono un caso diverso, perché ogni risposta generata richiede un giudizio, e il modello giudice è accessibile come servizio: ogni giudizio è una chiamata a pagamento, e replicare la griglia avrebbe moltiplicato una spesa già consistente, pari a circa 3700 chiamate. La conseguenza va dichiarata: i valori di accuratezza riportati nel seguito sono singole osservazioni, non medie su più esecuzioni, e non è quindi possibile associarvi una misura di dispersione. La lettura dei risultati ne tiene conto: il capitolo argomenta sui divari ampi e sistematici — quelli che si ripresentano nello stesso verso su tutti i livelli di k e su tutti i budget di token — e non su differenze di pochi centesimi fra configurazioni vicine, che una singola esecuzione non permette di distinguere dal rumore.

6.2 Valutazione del recupero

Il primo blocco di misure riguarda il solo recupero, a monte della generazione: quanto testo bisogna passare al modello perché i chunk giusti siano stati recuperati.

6.2.1 Confronto fra le configurazioni di recupero

La Figura 6.1 riporta il richiamo cumulato in funzione dei token, per la strategia adottata in produzione, al variare della modalità di recupero e dell'eventuale trasformazione della query. Le misure si riferiscono al solo recupero, prima del riordinamento.

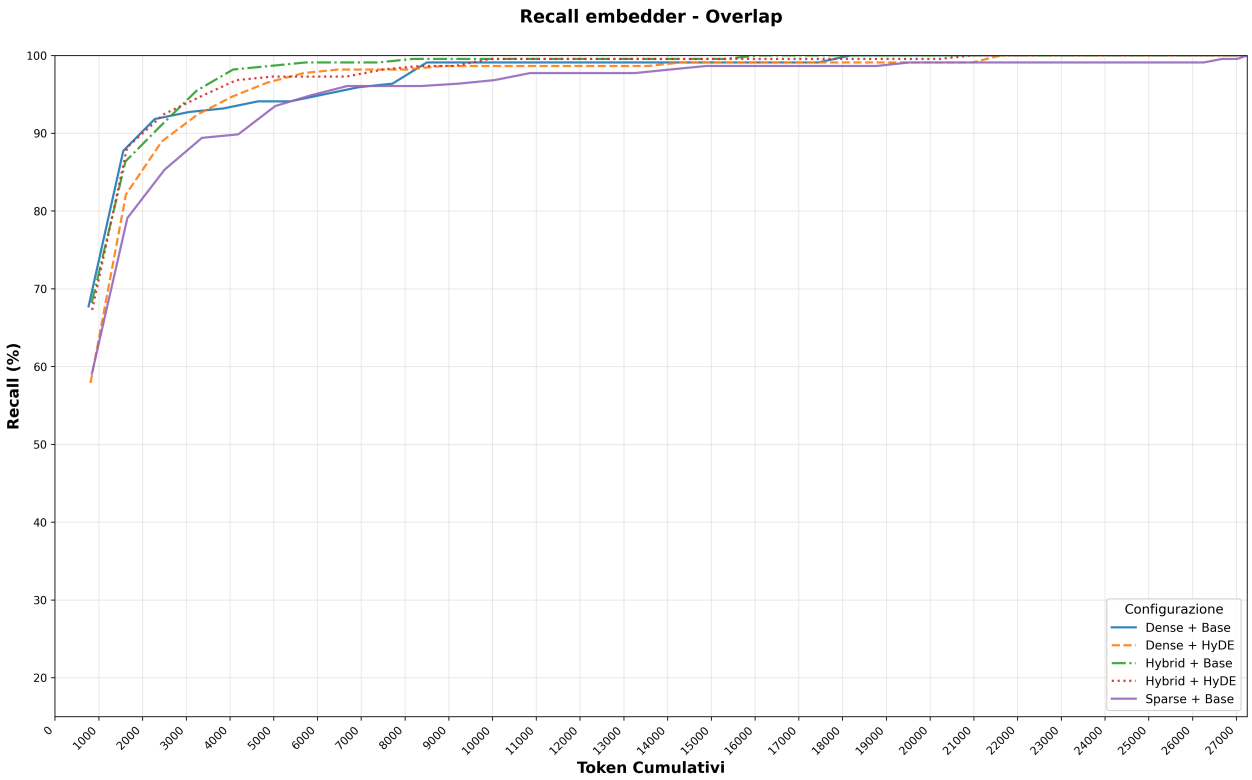


Figura 6.1: Richiamo cumulato in funzione dei token recuperati, prima del riordinamento, strategia Overlap. In legenda *Dense*, *Sparse* e *Hybrid* indicano il recupero denso, lessicale e ibrido; *Base* indica l'assenza di trasformazione della query. Oltre i 3000 token circa la configurazione ibrida senza trasformazione domina le altre; al di sotto le curve sono ravvicinate.

Qualche valore puntuale aiuta a leggere il grafico. Ai due budget più ampi della griglia sperimentale, 4096 e 6144 token, il richiamo delle cinque configurazioni è quello riportato nella Tabella 6.2.

Tabella 6.2: Richiamo delle cinque configurazioni di recupero, prima del riordinamento, a due budget di token, strategia Overlap. Valori letti dalle curve della Figura 6.1 e arrotondati al mezzo punto.

Configurazione	**4096 token**	**6144 token**
Ibrido, senza trasformazione	**98,0 %**	**99,0 %**
Ibrido + HyDE	96,5 %	97,5 %
Denso + HyDE	95,0 %	98,0 %
Denso, senza trasformazione	93,5 %	95,0 %
Lessicale	90,0 %	95,5 %

Tre osservazioni emergono. La prima è che, a partire da circa 3000 token, il recupero ibrido senza trasformazione della query domina tutte le altre configurazioni, e con un margine non trascurabile: a 4096 token recupera il 98 % dei chunk rilevanti contro il 93,5 % del solo canale denso, il che significa ridurre di circa due terzi i chunk rilevanti mancati. Sotto quella soglia le curve sono ravvicinate, e il recupero denso e l'ibrido con HyDE risultano per un breve tratto leggermente superiori; ma è una regione che i budget di 4096 e 6144 token superano ampiamente.

La seconda è che HyDE non aiuta dove conta. Sul canale denso isolato un effetto c'è — a 6144 token porta il richiamo dal 95 % al 98 % — ma sulla configurazione ibrida, che è quella adottata, la trasformazione peggiora il risultato a 4096 come a 6144 token. L'interpretazione è che il documento ipotetico generato da HyDE compensa un recupero denso debole introducendo terminologia che la domanda non conteneva, ma quella stessa terminologia inventata disturba il canale lessicale, che sulle sigle e sui codici del dominio lavora per corrispondenza esatta.

Dietro a questo c'è probabilmente una ragione più generale, che riguarda tutte le tecniche di trasformazione della query e non il solo HyDE: il dominio è troppo specifico perché un modello possa espandere la domanda a ragion veduta. Riformulare una domanda, generare un documento ipotetico o astrarla verso un concetto più generale sono operazioni che funzionano quando il modello conosce l'argomento e può quindi arricchire la domanda con termini che appartengono davvero a quel campo. Qui non è il caso: è improbabile che i nomi delle maschere del gestionale, i codici delle stazioni, le sigle interne e il modo in cui l'azienda chiama le proprie procedure compaiano

nei dati di pre-addestramento di un modello generalista. Ciò che l'espansione produce non è quindi terminologia del dominio, ma terminologia *plausibile* di magazzino in generale — corretta in astratto e assente dai documenti da recuperare. Il risultato è una domanda allontanata dal testo che dovrebbe trovare, ed è la spiegazione più probabile sia del risultato di HyDE sia dell'esito negativo delle prime prove con multi-query e step-back (sezione 6.4).

La terza è che il canale lessicale da solo è il peggiore su quasi tutto l'intervallo: resta nettamente indietro sotto i 5000 token, raggiunge e supera di poco il canale denso isolato in un breve tratto intorno ai 6000, e oltre gli 8000 torna a essere il peggiore. Non è quindi una riabilitazione del recupero lessicale, ma conferma che i due canali trovano chunk in parte diversi, ed è esattamente la ragione per cui la loro combinazione funziona meglio di entrambi.

6.2.2 Confronto fra le strategie di suddivisione

Le Figure 6.2 e 6.3 confrontano le tre strategie, ciascuna nella propria configurazione di recupero migliore — Overlap con il recupero ibrido, Hierarchical con il recupero denso, Docling con il recupero denso e HyDE — prima e dopo il riordinamento.

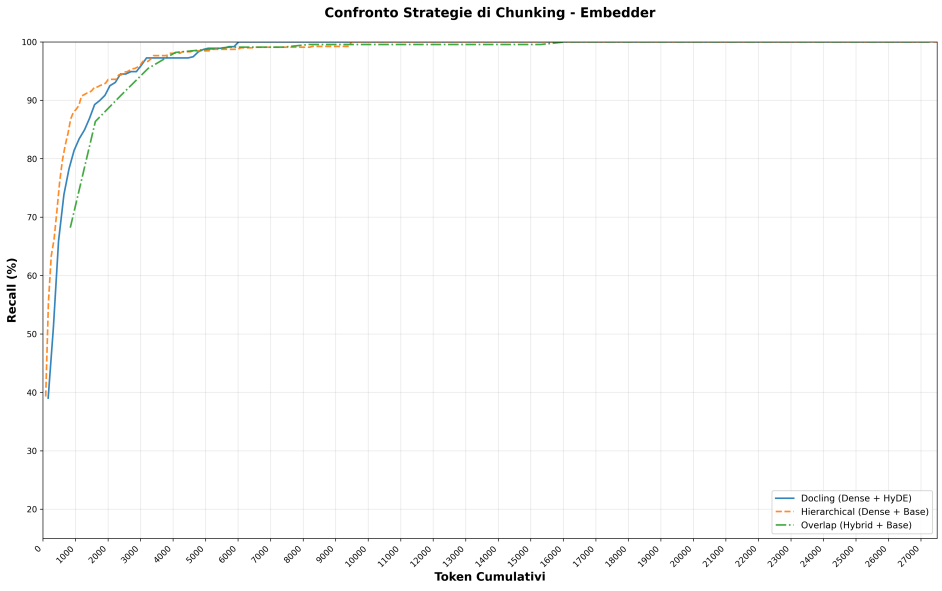


Figura 6.2: Richiamo cumulato dopo il solo recupero, prima del riordinamento, per le tre strategie di suddivisione nella rispettiva configurazione migliore.

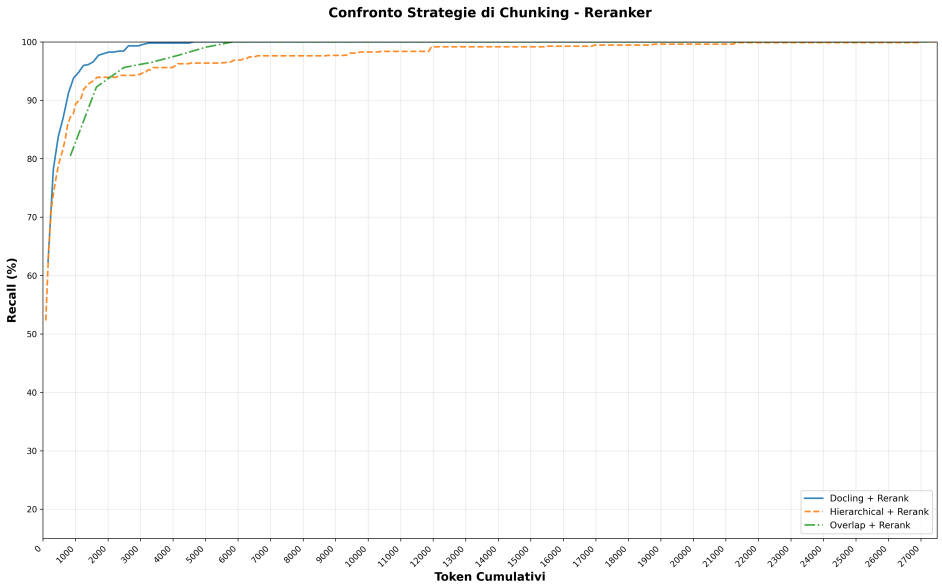


Figura 6.3: Richiamo cumulato dopo il riordinamento. Per Overlap e Docling il reranker sposta i chunk rilevanti verso le prime posizioni, e il richiamo pieno si raggiunge con un budget inferiore; per la strategia gerarchica la curva resta, oltre i 2500 token circa, al di sotto di quella della Figura 6.2.

Un'avvertenza sulla curva della strategia gerarchica. Le due figure non sono, per quella strategia, del tutto omogenee alle altre due, e la ragione sta in come la ground truth è stata assegnata. La strategia gerarchica produce lo stesso testo a tre livelli di granularità annidati — 2048, 512 e 128 token — e stabilire se il chunk giusto sia stato recuperato richiederebbe che la ground truth dicesse dove si trova la risposta *a ciascuno dei tre livelli*: il modello che ha costruito il dataset avrebbe dovuto indicare il blocco da 128 token, quello da 512 e quello da 2048 che contengono la risposta. La ground truth disponibile è invece assegnata al solo livello da 128 token, e il richiamo è stato conseguentemente calcolato contando come successo il recupero di uno di quei blocchi.

La conseguenza è che i valori riportati per la strategia gerarchica sono un *lower bound*: un caso in cui il sistema ha recuperato il chunk da 512 o da 2048 token contenente la risposta — e ha quindi fatto il suo lavoro — non viene contato come successo se il blocco da 128 associato non compare fra i risultati. Il richiamo reale di quella strategia è quindi almeno quello mostrato, e plausibilmente superiore.

La limitazione riguarda il solo richiamo. L'accuratezza end-to-end della sezione 6.3 non ne è affetta, perché lì al modello generativo sono stati passati i chunk effettivamente recuperati, quali

che fossero il loro livello e la loro dimensione, e il giudizio è stato espresso sulla risposta prodotta: non c'è alcun conteggio di chunk da cui possa derivare una sottostima.

Il confronto fra le due figure mostra l'effetto del riordinamento, che non è però uniforme. Per la strategia adottata il guadagno è netto ai budget piccoli — con il primo chunk il richiamo passa da circa il 68 % a circa l'80 % — e soprattutto il richiamo pieno si raggiunge intorno ai 6000 token invece che intorno ai 16 000, cioè entro il budget di esercizio; nella fascia intorno ai 4000 token le due curve sono invece equivalenti. Per Docling il vantaggio è netto su tutto l'intervallo, e il richiamo pieno arriva intorno ai 4700 token. Per la strategia gerarchica, infine, oltre i 2500 token circa il richiamo dopo il riordinamento è più basso che prima: è un effetto su cui pesa plausibilmente la cautela di misura appena discussa. Per la strategia adottata è comunque questo l'argomento che giustifica il costo del cross-encoder nella pipeline: porta l'intero richiamo dentro la finestra che il modello generativo può effettivamente ricevere.

Il limite, del resto, non indebolisce la conclusione del capitolo, ma la rafforza. Prima del riordinamento, nella regione dei budget piccoli — sotto i tremila token, dove le curve non sono ancora sature — la strategia gerarchica è quella che sale più rapidamente, e per quanto appena detto la sua curva reale è almeno quella mostrata. Eppure la sua accuratezza end-to-end si ferma a 0,759 contro lo 0,883 della strategia adottata. Lo stesso vale, senza alcuna cautela di misura, per Docling: dopo il riordinamento è la strategia più rapida di tutte e raggiunge il richiamo pieno intorno ai 4700 token, ben dentro il budget di esercizio, ma la sua accuratezza si ferma a 0,745.

Sul solo richiamo, in altre parole, le tre strategie si avvicinano piuttosto presto: prima del riordinamento tutte superano il 98 % intorno ai 5000 token. La differenza sostanziale emerge non qui, ma nella qualità della risposta finale, discussa nella sezione 6.3. È un risultato interessante di per sé: in questo esperimento il richiamo del recupero non si è rivelato un buon predittore della qualità end-to-end, perché non dice nulla su quanto il chunk recuperato sia *utilizzabile* dal generatore. È l'osservazione che la sezione 6.9 riprende.

6.3 Valutazione end-to-end

6.3.1 Risultati per strategia di suddivisione

La Figura 6.4 riporta l'accuratezza della pipeline completa per ciascuna strategia, al variare del numero di candidati e del budget di token.

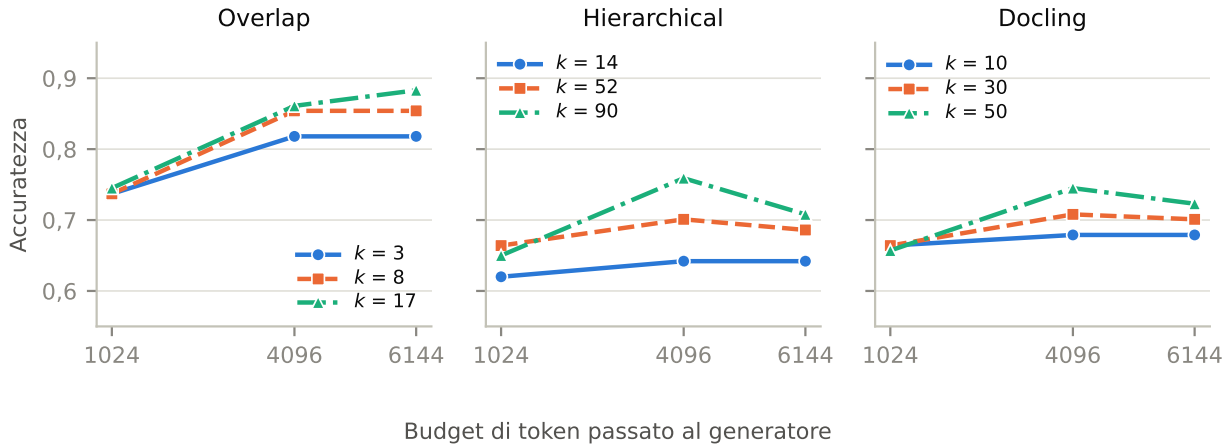


Figura 6.4: Accuratezza della pipeline completa al variare del budget di token passato al generatore, per i tre livelli di k della Tabella 6.1. Si noti la scala verticale comune ai tre pannelli ma ristretta all'intervallo 0,55–0,95.

I valori sono riportati per esteso nella Tabella 6.3.

Tabella 6.3: Accuratezza della pipeline completa. In grassetto la configurazione migliore per ciascuna strategia.

Strategia	**top-k**	**1024 token**	**4096 token**	**6144 token**
Overlap (1000 token)	3	0,737	0,818	0,818
	8	0,737	0,854	0,854
	17	0,745	0,861	**0,883**
Hierarchical	14	0,620	0,642	0,642
	52	0,664	0,701	0,686
	90	0,650	**0,759**	0,708
Docling HybridChunker	10	0,664	0,679	0,679
	30	0,664	0,708	0,701
	50	0,657	**0,745**	0,723

6.3.2 Discussione dei risultati end-to-end

Il confronto fra le configurazioni migliori, riportato nella Figura 6.5, mostra un distacco netto.

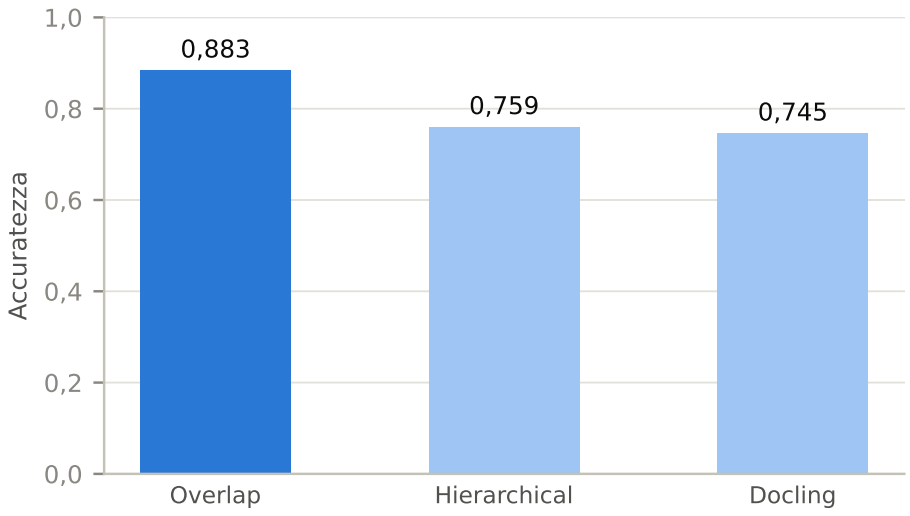


Figura 6.5: Accuratezza della configurazione migliore per ciascuna strategia di suddivisione. La strategia a chunk ampi con sovrapposizione, adottata in produzione, è evidenziata.

La strategia Overlap, adottata in produzione, raggiunge un'accuratezza di 0,883, contro 0,759 della strategia Hierarchical e 0,745 della strategia Docling: un distacco di oltre 0,12 sul secondo classificato. È il risultato che ha determinato la scelta descritta nella sezione 5.3.

Tre osservazioni meritano di essere sviluppate.

Il vantaggio è sistematico. La strategia adottata prevale in tutte e nove le celle della griglia, con margini compresi fra 0,073 e 0,160 sulla migliore delle alternative allo stesso livello di k e di budget. È un divario ben superiore a quello che una singola esecuzione potrebbe attribuire al rumore (sezione 6.1.10). Fra le due alternative, invece, la differenza delle configurazioni migliori — 0,014, cioè due domande su 137 — non permette di stabilire un ordine.

Il richiamo non basta. Come osservato nella sezione 6.2.2, a 4096 e 6144 token le tre strategie raggiungono un richiamo comparabile, ma divergono nettamente sulla qualità della risposta. La spiegazione plausibile è che un chunk ampio porta con sé il contesto necessario a interpretarlo — l'intera procedura, non il singolo passaggio — mentre chunk piccoli, per quanto correttamente recuperati, costringono il generatore a ricostruire una procedura da pezzi slegati.

Budget e numero di candidati interagiscono. In tutte e tre le strategie il salto maggiore avviene fra 1024 e 4096 token; oltre, il budget conta solo se k è abbastanza alto da riempirlo: con i valori di k più bassi le accuratezze a 4096 e 6144 token coincidono, plausibilmente perché i candidati non bastano a occupare lo spazio disponibile. Per la strategia adottata il budget pesa più di k — +0,116 da 1024 a 4096 token con $k = 17$, contro +0,043 da $k = 3$ a $k = 17$ a 4096 token —, mentre per la strategia gerarchica i due effetti sono comparabili. Oltre i 4096 token, per Hierarchical e Docling l'accuratezza cala ai livelli medio e alto di k: passare più testo al generatore oltre una certa soglia introduce rumore invece che informazione. La strategia adottata è l'unica a non peggiorare, e con $k = 17$ passa anzi da 0,861 a 0,883: una differenza di tre domande, che da sola non basterebbe a fondare una conclusione ma è coerente con il quadro. Con chunk ampi il testo aggiuntivo è ancora informazione utile; con chunk piccoli è per lo più materiale marginale.

6.4 Valutazione delle tecniche di trasformazione della query

Sono state valutate tre tecniche: multi-query augmentation, HyDE e step-back prompting. Tutte e tre sono state implementate con generazione strutturata a partire da prompt calati sul dominio del

WMS, riportati nell'Appendice B.3.

Multi-query e step-back hanno dato fin dalle prime prove risultati lontani dalle attese e sono state abbandonate senza una misura sistematica: il giudizio su queste due tecniche è quindi qualitativo. HyDE, che appariva più promettente, è stato analizzato in modo sistematico: per questo il documento ipotetico è stato generato e conservato insieme a ciascuna domanda del dataset di valutazione, così da poter ripetere le misure senza rigenerarlo.

L'esito è visibile nella Figura 6.1: per la strategia e la modalità di recupero adottate, HyDE peggiora il richiamo a 4096 e 6144 token rispetto alla stessa configurazione senza trasformazione, mentre sul solo canale denso lo migliora. Il quadro non è uniforme — per la strategia Docling la configurazione migliore risulta essere proprio quella densa con HyDE (Figura 6.2) — ma anche dove aiuta il margine è troppo contenuto per giustificare una chiamata aggiuntiva al modello generativo prima di ogni ricerca, che in esercizio costerebbe dai 5 ai 10 secondi.

La spiegazione è quella già discussa nella sezione 6.2.1: il modello non conosce il lessico del dominio, e i termini che introduce, plausibili ma assenti dalla documentazione, aggiungono rumore invece di avvicinare la domanda al documento.

6.5 Valutazione dell'estrazione dei problemi

La pipeline di trattamento delle conversazioni descritta nella sezione 5.1 non si presta alla stessa valutazione automatica: non esiste un riferimento rispetto a cui misurare quali problemi «avrebbero dovuto» essere estratti da un flusso di messaggi. L'intera catena, estrazione e deduplicazione, è stata quindi valutata manualmente sulle schede finali ottenute con due modelli diversi dallo stesso materiale, anonimizzato in locale prima dell'invio: Claude Opus 4.6 e Gemini in modalità *deep research*. Per ciascuno sono state esaminate quaranta schede, verificandone la correttezza rispetto ai messaggi da cui erano state ricavate. La misura è quindi una precisione: dice quante delle schede prodotte sono corrette, non quanti problemi presenti nelle conversazioni siano sfuggiti all'estrazione: questo secondo aspetto non è stato misurato.

La verifica di ciascuna scheda segue un percorso fisso, reso possibile dal fatto che l'anonimizzazione conserva i timestamp (sezione 5.1.2). A partire dalle date riportate nella scheda si risale ai punti corrispondenti della conversazione e si controllano tre condizioni: che il problema

descritto sia effettivamente presente in quei punti; che la descrizione sia conforme a quanto emerge dai messaggi; e che la causa e la soluzione riportate siano quelle effettivamente individuate nello scambio. Una scheda è giudicata corretta solo se tutte e tre le condizioni sono soddisfatte.

I risultati sono riportati nella Tabella 6.4.

Tabella 6.4: Valutazione manuale dell'estrazione dei problemi: schede finali corrette su quaranta esaminate.

Modello	**Corrette**	**Accuratezza**
Claude Opus 4.6	39 / 40	**0,975**
Gemini (*deep research*)	31 / 40	0,775

Il divario è netto — 0,975 contro 0,775 — su un compito che non ammette risposte parzialmente corrette: una scheda che attribuisce a un problema una causa o una soluzione che nei messaggi non compare è semplicemente sbagliata, e inserirla nell'indice significa introdurre un'informazione falsa nel sistema.

Sulla base di questo confronto la catena è stata affidata a Claude Opus 4.6: le schede indicizzate sono le 109 da esso prodotte, descritte nella sezione 5.1.4.

6.6 Controllo degli accessi

Il controllo degli accessi non è stato oggetto di una campagna di misura, e la ragione è che la sua correttezza non è una proprietà da stimare statisticamente, ma una proprietà della costruzione. Il filtro è applicato dal database a ciascuna interrogazione, prima del recupero (sezione 5.6): un chunk il cui gruppo proprietario non compare fra quelli accessibili all'utente non viene restituito dalla ricerca, e non esiste alcun percorso successivo per cui possa entrare nel contesto del modello. Un test su un insieme di domande servirebbe a verificare l'implementazione, non a stimare una proprietà che il meccanismo garantisce per ogni domanda.

Il rischio residuo non sta quindi nel filtro, ma a monte di esso: nell'assegnazione del gruppo a ciascun documento, che dipende dalla convenzione sul nome del file e quindi da chi lo nomina. Un documento etichettato con il gruppo sbagliato viene filtrato in modo perfettamente coerente con

un'etichetta errata. È un errore umano che nessuna valutazione del sistema potrebbe intercettare, ed è discusso con le altre garanzie del meccanismo nella sezione 4.5.4.

6.7 Analisi degli errori

Un'analisi sistematica degli errori, con una tassonomia costruita su un campione di risposte sbagliate, non è stata condotta. Due elementi permettono però di collocare con buona confidenza la fonte della maggior parte degli errori.

Il primo viene dai numeri. Nella configurazione adottata il richiamo dopo il riordinamento raggiunge il 100 % intorno ai 6000 token, cioè proprio al budget di esercizio (Figura 6.3), mentre l'accuratezza end-to-end si ferma a 0,883. Nella grande maggioranza dei casi in cui la risposta è sbagliata, dunque, il chunk che la conteneva era nel contesto: l'errore non è avvenuto nel recupero, ma a valle, nella generazione.

Il secondo viene dall'osservazione delle risposte, ed è coerente con il primo: il limite principale è il modello generativo, che fatica a comprendere un lessico tecnico come quello aziendale. Va considerato che le domande del profilo esperto — e, a maggior ragione, quelle reali — non sono formulate in modo pulito. Un operatore non chiede «come si esegue la procedura di riallineamento di un'unità di carico»; chiede qualcosa di molto più specifico e contratto, con il codice della stazione, la sigla della maschera e il messaggio di errore, dando per scontato un contesto che il modello non ha. Quando il chunk giusto è nel contesto, il modello deve ancora riconoscere che quei termini si riferiscono a ciò che il chunk descrive — e su terminologia che non ha mai visto in addestramento, non sempre ci riesce.

È la stessa ragione che, come argomentato nella sezione 6.9, aiuta a spiegare lo scarto fra le strategie di suddivisione osservato nella sezione 6.3.

6.8 Limiti della valutazione

Sei limiti vanno dichiarati esplicitamente, perché condizionano la portata dei risultati.

Lo spazio di ricerca è ridotto. Le misure di richiamo sono state ottenute su un corpus di circa 27 000 token, molto più piccolo dell'indice reale. Il richiamo a 4096 token su 27 000 non è

lo stesso che su un corpus di un ordine di grandezza superiore, e non c'è motivo di assumere che i valori si trasferiscano invariati. Due considerazioni attenuano il problema senza eliminarlo. La prima è che i documenti selezionati appartengono allo stesso dominio e hanno la stessa struttura di quelli riservati. La seconda è che il numero di candidati offre un margine rispetto a quanto il solo richiamo richiederebbe: dopo il riordinamento il richiamo satura intorno ai 6000 token, cioè sei o sette chunk, mentre il riordinamento lavora su $k_1 = 17$ candidati. Resta però un vincolo che nessuna delle due considerazioni rimuove: il budget di circa 6000 token per il contesto documentale è già vicino al massimo praticabile, perché la finestra del modello deve contenere anche il prompt di sistema, la domanda, l'eventuale cronologia e la risposta da generare, e superarlo stabilmente è difficile. Su un indice più ampio non ci sarebbe quindi margine per compensare con più testo.

La ground truth è generata automaticamente. Le domande sono supervisionate da un modello giudice con criteri severi e iterazione sul feedback, mentre l'assegnazione dei chunk di riferimento non passa per alcuna supervisione oltre al controllo di validità degli identificativi; in entrambi i casi il risultato resta costruito da un modello linguistico, e un chunk di riferimento indicato in modo impreciso altera la misura del richiamo della domanda corrispondente. Il rischio è attenuato da un controllo umano: prima di essere adottate come dataset di valutazione, le domande generate sono state esaminate a campione e valutate da colleghi del reparto (sezione 6.1.3). Il controllo resta però campionario e non ha prodotto una misura formale di accordo fra giudizio umano e automatico.

Il valutatore è un modello linguistico. L'uso di un modello come giudice introduce bias noti — preferenza per le risposte lunghe o prodotte dallo stesso modello, sensibilità alla formulazione. Il protocollo li limita affidando la generazione delle risposte e il loro giudizio a modelli diversi e ancorando il giudizio a una ground truth costruita in precedenza, ma non li elimina. Va inoltre notato che lo stesso modello, Gemini 2.5 Pro, ha generato le domande, assegnato i chunk di riferimento e giudicato le risposte: in tre chiamate distinte e indipendenti, ma con le eventuali tendenze sistematiche di un unico modello.

Il richiamo della strategia gerarchica è una stima per difetto. Per la ragione discussa nella sezione 6.2.2, la ground truth di quella strategia è assegnata al solo livello di granularità più fine, e il richiamo misurato ne è quindi un limite inferiore. Il confronto sul richiamo fra le tre strategie va letto con questa cautela; l'accuratezza end-to-end, invece, non è interessata dal problema.

Ogni configurazione è stata eseguita una sola volta. Come discusso nella sezione 6.1.10, le accuratezze sono singole osservazioni, senza una misura di dispersione: le differenze di pochi centesimi — fra cui lo 0,022 fra 4096 e 6144 token nella configurazione adottata e lo 0,014 fra le due alternative — non sono distinguibili dal rumore. Le conclusioni del capitolo poggiano per questo sui divari ampi e sistematici.

La valutazione è condotta su domande sintetiche. Il dataset è costruito da un modello a partire dai documenti, non raccolto fra le domande reali degli operatori, che sono spesso più contratte e più specifiche (sezione 6.7). I valori di accuratezza vanno quindi letti come una misura comparativa fra configurazioni, più che come una stima assoluta delle prestazioni in esercizio. Il limite è però attenuato da due fattori: le domande sono state controllate a campione da esperti del reparto prima di essere adottate, e il prompt che le genera è stato costruito tenendo conto delle loro indicazioni, così da avvicinare le domande al modo in cui gli operatori le pongono realmente.

6.9 Discussione

I risultati del capitolo convergono su un'unica lettura: il limite più grande del sistema è il modello generativo, messo di fronte a una terminologia tecnica che non conosce.

È questa la lettura che meglio spiega perché le tre strategie di suddivisione, che a 4096 e 6144 token recuperano i chunk giusti con efficacia paragonabile, producano risposte di qualità così diversa. Un modello che fatica già a collegare il lessico dell'azienda al contenuto dei documenti si confonde quando il contesto gli arriva spezzettato in molti chunk brevi, ciascuno con una parte della risposta; se invece riceve pochi chunk ampi, che contengono la procedura per intero, il lavoro di ricomposizione non è più a suo carico, e la risposta migliora. La preparazione dei dati compensa, in questo senso, i limiti del modello: non rende il modello più capace, ma gli chiede meno.

Ne segue anche un'indicazione sulla trasferibilità dei risultati. La superiorità netta dei chunk ampi è plausibilmente legata al modello impiegato: un modello di dimensioni maggiori, capace di ricostruire una procedura a partire da pezzi distinti e di orientarsi meglio su un lessico specialistico, avrebbe con buona probabilità ridotto lo scarto fra le strategie, forse fino ad annullarlo: è un'ipotesi, che i vincoli di memoria non hanno permesso di verificare. Il risultato non va quindi letto come una regola generale sul chunking, ma come la scelta giusta per un sistema che deve funzionare entro i vincoli della sezione 4.1.3.

7. Conclusioni

Il problema da cui questo lavoro è partito non era l'assenza di documentazione, ma la sua irreperibilità. La conoscenza operativa necessaria a far funzionare un magazzino gestito da un sistema complesso esiste — nei manuali, nei fogli di lavoro interni, nelle conversazioni in cui i problemi sono stati posti e risolti — ma è distribuita su sorgenti che non si parlano, espressa in registri linguistici diversi e, in parte, depositata in forma che nessuno strumento di ricerca raggiunge. Il suo costo più concreto era il ricorso ripetuto all'assistenza esterna, a pagamento, per problemi già affrontati e risolti in passato.

La risposta costruita è un sistema che risponde a domande in linguaggio naturale su quelle sorgenti, citando i documenti da cui la risposta proviene, rispettando i permessi di chi interroga e funzionando, in esercizio, interamente su infrastruttura interna all'azienda. Questo capitolo ne riassume i contributi, ne dichiara i limiti e indica le direzioni in cui potrebbe essere esteso.

7.1 Contributi

I contributi riprendono i quattro enunciati nella sezione 1.4; del primo si discutono separatamente due aspetti trasversali: il trattamento della componente multimodale e la fattibilità dell'esecuzione interamente locale.

7.1.1 Un sistema RAG completo in un contesto reale

Il primo contributo è il sistema stesso, nella sua interezza: dalla raccolta e preparazione dei dati fino all'interfaccia conversazionale con autenticazione e cronologia. Non si tratta di un prototipo dimostrativo ma di un sistema installato e utilizzabile, il che ha imposto di affrontare una serie di questioni che una prova di concetto può ignorare: la preparazione e l'indicizzazione dei documenti, la creazione e l'autenticazione degli utenti, il comportamento del sistema quando non sa rispondere.

Le scelte principali della pipeline — la strategia di suddivisione, la modalità di recupero, la rinuncia alla trasformazione della query, il modello di estrazione dei problemi — sono documentate insieme alla misura o alle prove che le sostengono, e non alla sola intuizione progettuale. È una

differenza di metodo che incide sul risultato: la strategia di suddivisione adottata è stata scelta perché ha vinto il confronto sperimentale, con 0,883 di accuratezza contro 0,759 e 0,745 delle alternative.

7.1.2 Estrazione di conoscenza da sorgenti non strutturate

La pipeline che trasforma anni di conversazioni operative in schede di problema strutturate — con descrizione, causa, soluzione e note — è probabilmente il contributo meno visibile e più trasferibile del lavoro. Affronta un caso che la letteratura sul RAG tratta raramente: una sorgente priva di struttura documentale, in cui le unità informative non hanno confini espliciti e vanno ricostruite.

Il percorso che ha portato alle 109 schede di problema è istruttivo proprio per il tentativo che lo ha preceduto. L'approccio a finestra scorrevole, in cui il modello segmenta il flusso decidendo di volta in volta dove finisce un problema, fallisce perché chiede una decisione globale a partire da una vista locale: il modello non ha mai sotto gli occhi il quadro d'insieme che servirebbe per stabilire quei confini. L'approccio adottato rovescia l'impostazione — un anno intero di conversazioni in input, estrazione delle unità informative, deduplicazione successiva — e sfrutta la finestra di contesto ampia per ricostruire relazioni fra messaggi distanti nel tempo.

La valutazione manuale riportata nella sezione 6.5 quantifica l'esito: l'accuratezza è di 0,975 sulle quaranta schede finali esaminate, contro 0,775 del modello alternativo confrontato. È un risultato che dipende in misura determinante dalla scelta del modello impiegato in questa fase, e che va quindi letto come proprietà della coppia metodo-modello più che del solo metodo.

7.1.3 Trattamento della multimodalità

Il trattamento esplicito di immagini e tabelle è stato affrontato con la strategia della conversione in testo: le tabelle in Markdown, le immagini in descrizioni prodotte da un modello visione-linguaggio, reintegrate nel documento, con le descrizioni delle immagini racchiuse fra marcatori espliciti che le rendono distinguibili. La scelta è stata dettata dal vincolo di memoria — mantenere un solo spazio di rappresentazione e nessun modello multimodale residente — e ha il pregio di rendere uniformi su tutto il corpus il recupero ibrido e il filtraggio sui permessi. L'effetto di questa scelta sulla qualità del recupero non è stato misurato separatamente: il contributo è progettuale più che sperimentale.

7.1.4 Controllo degli accessi integrato nel recupero

Il controllo degli accessi è realizzato come condizione della ricerca e non come filtro sui risultati: Qdrant riceve, insieme ai vettori denso e sparso della domanda, l'insieme dei gruppi accessibili all'utente, e la visita dell'indice considera soltanto i chunk compatibili. La garanzia che ne deriva è di natura architetturale — un contenuto non autorizzato non entra nel contesto del modello generativo, e quindi non può influenzare la risposta nemmeno indirettamente — e non dipende dalla capacità del modello di rispettare un'istruzione.

È un aspetto a cui la letteratura sul RAG dedica poco spazio, perché assume quasi sempre un corpus uniformemente accessibile; in un sistema aziendale, dove utenti diversi devono ottenere risposte diverse alla stessa domanda, è invece un requisito di partenza.

7.1.5 Una metodologia di valutazione riutilizzabile

La procedura di valutazione costruita per questo lavoro è riutilizzabile su qualunque corpus documentale privo di dataset di riferimento. Il suo elemento distintivo è la separazione fra generazione delle domande e assegnazione dei chunk di riferimento: le domande nascono dal documento intero e restano identiche per tutte le strategie di suddivisione, mentre la mappatura verso i chunk viene ricostruita per ciascuna. Senza questa separazione — come emerso nel primo tentativo, in cui domande e riferimenti erano generati insieme dal documento già partizionato — ogni strategia verrebbe valutata su domande proprie, e il confronto sarebbe privo di significato.

7.1.6 Fattibilità dell'esecuzione interamente locale

Infine, il lavoro mostra che un sistema di questo tipo è realizzabile su infrastruttura interna, con una sola GPU di fascia consumer a sostenere tutta l'inferenza in esercizio. Il prezzo da pagare si vede su due piani. Il primo è la qualità della risposta: come mostrato nella sezione 6.9, il limite principale del sistema è proprio il modello generativo, che fatica con il lessico tecnico dell'azienda, e la preparazione dei dati ne compensa i limiti senza eliminarli. Il secondo è il *margine di manovra*: ogni stadio aggiuntivo della pipeline compete per la stessa memoria e lo stesso tempo di GPU, e le architetture più elaborate restano fuori portata.

Questo ha orientato l'intero progetto verso una direzione precisa: investire nella qualità dei dati forniti al modello — preprocessing, suddivisione, recupero, riordinamento — invece che nella sofisticazione di ciò che il modello deve fare con essi. I risultati del Capitolo 6 suggeriscono che, in questo dominio, sia anche la direzione più redditizia.

7.2 Limiti del lavoro

I limiti del lavoro si collocano su quattro piani — il modello generativo, la valutazione, il controllo degli accessi e l'infrastruttura — a cui si aggiunge l'assenza di una valutazione sul campo.

Limiti del modello generativo. La valutazione indica nel modello generativo il limite principale del sistema (sezione 6.7): dopo il riordinamento il richiamo della configurazione adottata arriva al 100 % entro il budget di esercizio, ma l'accuratezza si ferma a 0,883, e l'esame delle risposte errate indica come causa principale un modello che non riconosce il lessico tecnico dell'azienda. Un modello più capace non è compatibile con il budget di memoria attuale.

Limiti della valutazione. La campagna è stata condotta su un sottoinsieme del corpus — quattordici documenti, circa 27 000 token — selezionato per poter usare modelli esterni senza violare il vincolo di riservatezza. Lo spazio di ricerca su cui il richiamo è stato misurato è quindi sensibilmente più piccolo di quello reale, e i valori non si trasferiscono invariati; un numero di candidati superiore a quello che il solo richiamo richiederebbe attenua il problema, ma resta una contromisura e non una verifica. La ground truth, inoltre, è generata automaticamente: la supervisione del modello giudice e il controllo a campione da parte dei colleghi ne alzano la qualità, ma non equivalgono a una validazione umana sistematica. Lo stesso vale per la valutazione delle risposte, affidata a un modello linguistico con i bias noti discussi nella sezione 6.8. Ogni configurazione, infine, è stata eseguita una sola volta: le accuratezze sono singole osservazioni, e le differenze di pochi centesimi fra configurazioni vicine non sono distinguibili dal rumore.

Limiti del controllo degli accessi. La granularità è quella del documento: non è possibile rendere visibile una parte di un documento e non un'altra. L'associazione fra documento e gruppi poggia su una convenzione di denominazione dei file, il che rende la correttezza del meccanismo

dipendente dalla disciplina con cui i documenti vengono nominati. Resta inoltre aperto il canale indiretto descritto nella sezione 4.5.4: un documento autorizzato che citi contenuti riservati li rende comunque accessibili a chiunque possa leggerlo.

Limiti infrastrutturali. La dimensione dei modelli, il grado di quantizzazione e la lunghezza del contesto sono determinati dal budget di memoria di una singola scheda, occupata pressoché per intero dai tre modelli in esercizio. Alcune direzioni — il recupero multimodale nativo, le architetture agentiche — sono state escluse anzitutto perché non compatibili con quel budget, o perché il modello che vi rientra non era in grado di sostenerle. Al vincolo di esecuzione locale si deve anche l'esclusione delle immagini allegate ai messaggi (sezione 5.1.6): il modello visione-linguaggio eseguibile in locale non era in grado di descriverle in modo affidabile, e una parte di informazione operativa resta perciò fuori dall'indice.

Assenza di una valutazione sul campo. Tutte le misure riportate sono ottenute su un dataset costruito ad hoc. Manca una valutazione con gli utenti reali su un periodo prolungato, che è l'unica in grado di dire se il sistema riduca effettivamente le richieste all'assistenza e il carico sulle persone di riferimento — che era il problema di partenza.

7.3 Sviluppi futuri

7.3.1 Misura dell'effetto sulle richieste di assistenza

La valutazione di questo lavoro misura la correttezza delle risposte, che è la condizione perché il sistema possa sostituire una richiesta di assistenza, ma non la sostituzione stessa. Il passo successivo naturale è misurarla: confrontare il numero di richieste al fornitore prima e dopo l'introduzione del sistema, distinguendo quelle relative a problemi già presenti fra le schede ricavate dalle conversazioni, su un periodo abbastanza lungo da assorbire la variabilità dell'attività di magazzino. È la misura che traduce la qualità delle risposte nel risparmio che ha motivato il progetto (sezione 3.5).

7.3.2 Citazioni a grana di pagina

La citazione si ferma oggi al nome del documento, e su un manuale di alcune decine di pagine questo lascia all'utente una ricerca da fare a mano. Il riferimento alla pagina era previsto ed è stato abbandonato per una ragione tecnica e non di progetto: l'informazione di provenienza, presente nella rappresentazione prodotta dal parser, non viene propagata ai chunk in fase di indicizzazione (sezione 4.4.3). Recuperarla richiede di intervenire sul passaggio fra parsing e suddivisione, trasportando la provenienza nei metadati di ciascun nodo. È l'estensione con il miglior rapporto fra costo e beneficio percepito fra quelle elencate qui: non cambia nulla dell'architettura e migliora sensibilmente la verificabilità delle risposte, che è uno degli obiettivi dichiarati del sistema.

7.3.3 Autorizzazione a grana più fine

Due estensioni sono naturali. La prima è spostare l'associazione fra documento e gruppi dalla convenzione sul nome del file a una base dati dedicata, eliminando la dipendenza dalla disciplina di denominazione e permettendo di modificare la visibilità senza rinominare i file né ricalcolare gli embedding. La seconda è portare la granularità dal documento alla sezione o al chunk, così da gestire documenti a visibilità mista — il caso, oggi irrisolto, di una procedura in gran parte pubblica con una sezione riservata. Il costo è un modello di autorizzazione più complesso da amministrare, e va valutato se i profili di accesso aziendali lo giustifichino.

7.3.4 Recupero multimodale nativo

La conversione delle immagini in testo è una compressione con perdita: ciò che il modello visione-linguaggio non ha ritenuto rilevante non è più recuperabile. Due direzioni eliminerebbero questa perdita. ColPali [10] indicizza direttamente la pagina renderizzata e rinuncia al parsing, un vantaggio soprattutto dove il testo è difficile da estrarre, ma fissa la granularità alla pagina e richiede un indice multi-vettore molto più pesante. Più adatta a un corpus nativamente digitale come questo è la seconda: affiancare ai chunk testuali le immagini estratte, in uno spazio multimodale e con un modello generativo capace di leggerle, conservando chunk ampi, canale lessicale e schede di problema, al costo di modelli più pesanti, praticabili con una seconda scheda o con modelli più compatti.

7.3.5 Recupero adattivo

La rinuncia alle tecniche di trasformazione della query è stata una decisione binaria: sulla configurazione adottata il beneficio osservato era nullo o negativo, e si è scelto di non applicarle mai. Un'alternativa più raffinata è attivarle selettivamente, solo quando la domanda lo richiede — per esempio quando il recupero restituisce punteggi bassi o chunk fra loro poco coerenti. Sarebbe il modo naturale di superare il compromesso discusso nella sezione 5.10: il costo verrebbe pagato solo sui casi difficili, dove il beneficio atteso è maggiore. Lo stesso principio si applicherebbe al recupero iterativo.

Nella stessa direzione vanno le architetture più elaborate scartate nella sezione 5.10 — quella agentica e quella basata su indici riassuntivi. Entrambe sono state messe da parte anzitutto perché il modello generativo locale, l'unico compatibile con il budget di GPU, mostra i propri limiti proprio nei compiti che quelle architetture gli chiederebbero, l'orchestrazione di più strumenti e la sintesi. Un modello più capace a parità di memoria, o una seconda scheda, cambierebbero i termini della questione.

7.3.6 Apprendimento dal feedback degli utenti

L'infrastruttura per raccogliere i giudizi degli utenti sulle risposte è già presente: Chainlit affianca a ogni risposta i pulsanti di valutazione positiva e negativa, e i giudizi vengono memorizzati nella base dati PostgreSQL dell'applicazione, in una tabella dedicata del framework. Il materiale così raccolto, però, non è ancora sfruttato. Un insieme di giudizi raccolti sul campo avrebbe due usi distinti: fornire finalmente una validazione umana delle metriche automatiche del Capitolo 6, e costituire il materiale per un affinamento del modello di embedding o del reranker sul dominio specifico.

7.3.7 Estensione ad altre sorgenti aziendali

Il sistema oggi risponde sulla documentazione, non sui dati. Una direzione naturale è l'integrazione con i dati strutturati del gestionale, così da poter rispondere non solo a «come si fa» ma anche a «qual è lo stato di». È un'estensione che cambia però la natura del sistema, perché richiede la

traduzione della domanda in interrogazione strutturata e un modello di autorizzazione applicato ai dati oltre che ai documenti.

7.4 Considerazioni finali

Costruire un sistema RAG che funzioni in una dimostrazione è oggi relativamente semplice: gli strumenti sono maturi e le pipeline di base ben documentate. Portarlo all'uso quotidiano in azienda è un problema diverso, e questo lavoro suggerisce che la distanza fra i due stati sia fatta soprattutto di questioni che la letteratura tratta poco.

La prima è che la qualità dipende dai dati più che dall'architettura. Lo scarto di oltre 0,12 in accuratezza fra la strategia di suddivisione migliore e le altre è il più ampio fra quelli misurati sulla pipeline di risposta, e nasce da come il testo viene preparato, non da stadi aggiuntivi della pipeline: HyDE, l'unica tecnica di trasformazione della query misurata in modo sistematico, non ha migliorato nemmeno il richiamo della configurazione adottata; e la sorgente che ha richiesto più lavoro — le conversazioni — è anche quella che nessuna pipeline standard avrebbe saputo trattare.

La seconda è che i vincoli non sono un contorno ma parte del progetto. L'obbligo di eseguire tutto in locale non ha solo ristretto la scelta dei modelli: ha determinato quanti stadi la pipeline potesse permettersi, e quindi dove valesse la pena investire.

La terza è che in un contesto reale gli utenti non sono intercambiabili. Un sistema che risponde a tutti allo stesso modo è più semplice da costruire e inadatto a essere usato; il controllo degli accessi non è una funzionalità aggiuntiva, ma una condizione perché il sistema possa esistere in azienda.

Resta, infine, la misura che più interessa all'azienda: quante richieste all'assistenza il sistema permette di evitare. Questo lavoro ne ha costruito le premesse — un sistema che in valutazione risponde correttamente a quasi nove domande su dieci, cita le proprie fonti ed è istruito a dichiarare quando non sa — ma il suo valore economico si misurerà nell'uso quotidiano, ogni volta che un problema già risolto in passato trova la sua soluzione senza una nuova chiamata al fornitore.

A. Organizzazione del codice

Il codice sviluppato nel corso del lavoro è organizzato in tre repository, corrispondenti ai blocchi funzionali descritti nel Capitolo 4, come riportato nella Tabella A.1.

Tabella A.1: Repository del progetto e sezioni della tesi corrispondenti.

Repository	**Contenuto**	**Riferimento in tesi**
`preprocessing`	Ingestion dei documenti: parsing con Docling, descrizione delle immagini, chunking, indicizzazione. Comprende la campagna di valutazione	Sez. 4.3, 5.2, 5.3; Cap. 6
`messages`	Trattamento delle conversazioni: anonimizzazione, estrazione e deduplicazione dei problemi	Sez. 5.1
`Chainlit`	Applicazione conversazionale: interfaccia, autenticazione, gruppi, pipeline di risposta	Sez. 4.7, 5.6

Le repository non sono pubblicamente accessibili: contengono configurazioni, prompt e riferimenti ai dati aziendali che ricadono sotto il vincolo di riservatezza discusso nella sezione 1.3.

B. Prompt

Questa appendice raccoglie per esteso i principali prompt usati nelle diverse fasi del sistema e della campagna di valutazione. Sono riportati nel testo effettivamente impiegato, salvo dove diversamente indicato; il nome commerciale del sistema di gestione del magazzino è sostituito con la sigla generica WMS.

B.1 Prompt di sistema della pipeline

È il prompt che accompagna i chunk selezionati dal reranker nella richiesta al modello generativo (sezione 5.8.3). Il segnaposto `{context}` viene sostituito dai chunk, ciascuno preceduto dall'intestazione che ne dichiara il documento di provenienza.

Codice B.1: Prompt di sistema per la generazione della risposta.

```
Sei un assistente tecnico esperto del sistema WMS. Il tuo obiettivo
è fornire risposte precise, professionali e basate esclusivamente sui
dati forniti.

### REGOLE DI COMPORTAMENTO:
1. Usa SOLTANTO il contesto fornito nei tag <context> per rispondere.
2. Se le informazioni necessarie per rispondere non sono presenti nel
   contesto, rispondi testualmente: 'Mi dispiace, ma non ho informazioni
   sufficienti nel manuale WMS per rispondere a questa domanda.'
3. Non utilizzare conoscenze esterne al di fuori del contesto fornito.
4. Mantieni un tono tecnico, asciutto e professionale.
5. Se possibile, cita la sezione o il capitolo specifico se presente nel
   testo, cita anche il nome del documento.

<context>
{context}
</context>

### ISTRUZIONE FINALE:
Rispondi alla domanda dell'utente in modo strutturato (usa elenchi
puntati se la procedura è complessa).
```

B.2 Prompt di preprocessing

B.2.1 Anonimizzazione delle conversazioni

Usato nella fase descritta nella sezione 5.1.2, applicato a finestre successive di cinquanta righe del file di conversazione.

Codice B.2: Prompt di anonimizzazione.

```
Sei un assistente che anonimizza trascrizioni di chat di supporto tecnico
per sistemi warehouse.

TESTO DA ANONIMIZZARE:
{chunk}

ISTRUZIONI:
1. Sostituisci:
   - Nomi reali di persone (nome, cognome o entrambi) -> Persona
   - Nomi di aziende o clienti reali -> Azienda
   - Codici identificativi: ID numerici lunghi, pallet ID, LU-ID -> COD
   - Prezzi e importi monetari -> PREZZO
   - Il nome proprio del WMS aziendale -> wms
2. Non serve mantenere coerenza tra un'occorrenza e l'altra: ogni entità
   va sostituita con l'etichetta generica corrispondente, senza
   numerazione.
3. NON sostituire:
   - Timestamp (date e ore)
   - Termini tecnici: AGV, WMS, OIL, JIRA, ITS, TPO, QC, AF, ecc.
   - Nomi di file allegati (es. IMG-20210624-WA0000.jpg)
   - Emoji, simboli speciali
4. Mantieni la struttura esatta del testo (righe, spaziatura,
   punteggiatura).
5. Restituisci ESCLUSIVAMENTE il testo anonimizzato, senza commenti,
   spiegazioni o blocchi di codice attorno.
```

B.2.2 Estrazione e deduplicazione dei problemi

Sono i due prompt della catena descritta nella sezione 5.1: il primo è applicato separatamente alla conversazione anonimizzata di ciascun anno, il secondo riceve i cinque elenchi annuali così ottenuti e li consolida in un unico documento.

Codice B.3: Estrazione dei problemi dalle conversazioni.

```
Sei un analista tecnico specializzato in sistemi di warehouse
management. Ti fornisco un file contenente conversazioni WhatsApp
esportate, relative a comunicazioni operative e di supporto tecnico su
un sistema WMS.
Il tuo compito è analizzare il file per produrre un elenco cronologico
dei problemi tecnici e operativi emersi nelle conversazioni.
Per ogni problema trovato, restituisci una voce strutturata così:
• [DATA e ORA] — [TITOLO BREVE DEL PROBLEMA]

Descrizione: cosa è successo, quali sistemi/funzionalità erano coinvolti
e quale era l'impatto operativo
Causa identificata: se emersa dalla conversazione
Soluzione / Workaround: cosa è stato fatto per risolvere, anche
parzialmente. Se non è stata trovata una soluzione definitiva, indicalo
esplicitamente.
Note: eventuali azioni di follow-up richieste, ticket aperti, migliorie
segnalate
Regole
Includi solo problemi concreti (tecnici, operativi, di configurazione,
di performance). Ignora conversazioni di routine senza problemi.
Se un problema si sviluppa in più momenti della stessa giornata,
trattalo come una singola voce con la data/ora di prima segnalazione.
Usa un linguaggio tecnico ma chiaro, sintetico.
Ordina tutto in ordine cronologico crescente.
Per eseguire questa ricerca basati ESCLUSIVAMENTE sul file che ti ho
dato in input
```

Codice B.4: Consolidamento e deduplicazione dei problemi.

```
Ti fornisco 5 file, uno per ogni anno, contenenti ciascuno un elenco
cronologico di problemi tecnici e operativi relativi al sistema WMS.
Il tuo compito è produrre un unico documento consolidato, seguendo
queste istruzioni:

Fase 1 — Unificazione
Unisci i problemi dei 5 file in un unico elenco cronologico, mantenendo
per ogni voce la struttura originale:

[DATA e ORA] — [TITOLO BREVE DEL PROBLEMA]

Descrizione
Causa identificata
Soluzione / Workaround
Note

Fase 2 — Deduplicazione e raggruppamento
Identifica i problemi ricorrenti o molto simili (stessa causa, stesso
sistema coinvolto, stessa tipologia di soluzione) e raggruppali in una
voce unificata, strutturata così:

[TITOLO DEL PROBLEMA RICORRENTE]

Occorrenze: elenco delle date in cui il problema si è verificato
Descrizione generale: sintesi comune del problema e del suo impatto
operativo
Causa: causa ricorrente identificata
Soluzione / Workaround consolidato: cosa veniva fatto ogni volta per
risolvere
Frequenza: quante volte si è verificato e in quali anni
Note: eventuali evoluzioni del problema nel tempo, se la soluzione è
cambiata tra un'occorrenza e l'altra, se è mai stata trovata una
soluzione definitiva

Fase 3 — Struttura dell'output
```

```
Organizza il documento finale in due sezioni:
Sezione A — Problemi ricorrenti (raggruppati)
I problemi che si sono ripetuti nel tempo, ordinati per frequenza
decrescente (dal più frequente al meno frequente).
Sezione B — Problemi singoli (cronologico)
I problemi che si sono verificati una sola volta, in ordine cronologico
crescente, raggruppati per anno.

Regole generali

Due problemi sono "simili" se condividono almeno due di questi tre
elementi: stesso sistema/componente coinvolto, stessa causa, stessa
soluzione.
Se hai dubbi sul raggruppamento, mantieni le voci separate e aggiungi una
nota "Possibile correlazione con: [titolo altro problema]".
Usa un linguaggio tecnico ma chiaro e sintetico.
Non perdere nessuna informazione rilevante presente nei file originali.
```

B.2.3 Descrizione delle immagini

Riportato anche nel Codice 5.4 (sezione 5.2.2), è passato al modello visione-linguaggio tramite la configurazione di Docling.

Codice B.5: Prompt per la descrizione delle immagini (ripetuto per comodità).

```
Descrivi questa immagine in circa 3 frasi in modo diretto e concreto,
concentrandoti su elementi visibili, azioni, funzioni o informazioni
presenti. Evita frasi generiche come 'l'immagine mostra', 'si vede',
'c'è', ecc. Usa frasi informative che possano essere immediatamente
utili per un sistema di recupero basato su contenuti (RAG).
```

B.3 Prompt delle tecniche di trasformazione della query

Sono riportati qui i prompt delle tre tecniche valutate nella sezione 6.4 e non adottate in produzione.

B.3.1 HyDE

Codice B.6: Generazione del documento ipotetico (HyDE).

```
Sei un tecnico esperto del sistema WMS.
Genera una soluzione procedurale ipotetica per il seguente problema
utente. Scrivi un paragrafo tecnico tratto da un manuale di gestione di
magazzino (WMS) o procedure ERP che spieghi come risolvere il problema.
Usa terminologia specifica di magazzino, nomi di maschere o processi
standard.

Domanda utente: {query}
```

B.3.2 Step-back prompting

Codice B.7: Generazione della domanda astratta (step-back).

```
Sei un assistente esperto del sistema WMS (ERP di magazzino).
Data una domanda utente molto specifica, riformulala in una versione PIÙ
GENERALE e astratta: una 'step-back question' che catturi il concetto, il
processo o la categoria di riferimento sottostante, così da recuperare
documentazione di contesto più ampia utile a rispondere alla domanda
originale.

Regole:
- Astrai dai dettagli operativi (valori specifici, codici, messaggi di
  errore, nomi di pulsanti) verso il processo/concetto generale.
- La step-back question deve essere sensata da sola e rispondibile da
  documentazione procedurale.
- NON deve essere identica o quasi identica alla domanda originale.
- Scrivila in linguaggio naturale, come la farebbe un operatore che vuole
```

```
  capire il principio generale.

Esempio:
Domanda specifica: "Perché il tasto 'Conferma' nella maschera di uscita
merce è grigio e non cliccabile?"
Step-back: "In quali condizioni un'operazione di uscita merce nel WMS
risulta bloccata o non confermabile?"

Domanda utente: {query}
```

B.3.3 Multi-query

Codice B.8: Generazione di riformulazioni alternative (multi-query).

```
Sei un assistente esperto del sistema WMS (ERP di magazzino).
Data una domanda utente, genera 4 riformulazioni alternative della stessa
domanda, utili a migliorare il retrieval in un sistema RAG su
documentazione logistica.

Regole:
- Mantieni intatto il significato e l'intento della domanda originale.
- Varia il lessico: sinonimi, termini tecnici specifici di magazzino
  (UDC, picking, stoccaggio, carico, ecc.), parafrasi naturali.
- Alterna formulazioni colloquiali e tecniche.
- NON aggiungere informazioni non presenti né inventare vincoli non
  impliciti nella domanda.
- NON ripetere la domanda originale tra le riformulazioni.

Domanda utente: {query}
```

B.4 Prompt della campagna di valutazione

B.4.1 Generazione delle domande

È il prompt descritto nella sezione 6.1.2. Il segnaposto `{previous_feedback}` è vuoto alla prima iterazione e contiene il giudizio del modello valutatore nelle successive. Per motivi di spazio il prompt è riportato in forma abbreviata: le regole e le categorie di domande sono tutte presenti, ma sono stati omessi gli esempi che le accompagnano nel testo originale.

Codice B.9: Generazione del dataset di domande.

```
Sei un Architetto di Dataset per la valutazione di sistemi RAG su
documentazione logistica (WMS).

OBIETTIVO: genera domande che coprano tre profili utente:
  - L'Operatore Esperto: vuole risolvere un problema specifico o
    compilare un campo.
  - Il Novizio/Utente in Training: non conosce il sistema, vuole capire i
    concetti base o il flusso generale.
  - L'Amministratore: deve impostare parametri o gestire eccezioni
    complesse. Cerca la logica dietro l'azione.

REGOLE CRITICHE DI STILE:
1. NO "QUIZ MODE" ACCADEMICO
   - EVITA: "Definisci cos'è una UDC." (troppo scolastico)
   - PREFERISCI: "Come faccio a creare un carico se il camion non è
     ancora arrivato fisicamente?"
2. OBIETTIVO OPERATIVO
   La domanda deve sottintendere un'intenzione: anche se teorica, deve
   servire a fare qualcosa.
3. NATURALITÀ E LIVELLI DI CONOSCENZA
   Esperto: termini precisi, omette i passaggi ovvi.
   Novizio: termini generici, confuso sui processi.
4. REALISMO
   NON suggerire esplicitamente nella domanda in quale parte del
   documento si trova la risposta.
```

```
COMPITI DI GENERAZIONE:
  - DOMANDE SINGLE-HOP: basate su un singolo punto del documento; focus
    su errori, campi specifici, bottoni.
  - DOMANDE MULTI-HOP: collegano informazioni di più parti del
    documento.
  - DOMANDE ONBOARDING E FLUSSI MACRO: come iniziare un'attività o
    creare nuove entità da zero.

DOCUMENTO COMPLETO:
{full_document_text}

{previous_feedback}
```

B.4.2 Supervisione delle domande

È il prompt del modello giudice che valuta l'insieme delle domande generate e produce il feedback per l'iterazione successiva (sezione 6.1.3).

Codice B.10: Supervisione del set di domande.

```
Sei un valutatore esperto di dataset di domande utilizzati per valutare
un retriever in un sistema RAG (Retrieval-Augmented Generation).

Ti vengono forniti:
1) Un documento markdown completo
2) Una lista di domande generate a partire dal documento.

Il tuo compito è valutare l'INSIEME delle domande come set, non le
singole domande in modo isolato.

Documento:
{full_document_text}

Domande generate:
{lista_domande}
```

```
Istruzioni di valutazione:

- Analizza attentamente l'insieme delle domande, ragionando step-by-step.
- Valuta in particolare i seguenti aspetti:
  • Tutte le domande sono effettivamente rispondibili usando SOLO il
    contenuto del documento?
  • Ci sono domande ambigue, poco specifiche o fuorvianti?
  • Ci sono ridondanze o domande che coprono lo stesso concetto?
  • Le domande coprono parti diverse e rilevanti del documento?
  • Le domande sono utili per valutare un retriever RAG (non banali, non
    puramente lessicali)?

- Scrivi un unico ragionamento coerente e dettagliato che giustifichi la
  tua valutazione complessiva.
- Il ragionamento deve spiegare come sei arrivato al verdetto finale,
  senza essere inutilmente prolisso.

- Dopo il ragionamento, scrivi un feedback finale in cui spieghi
  chiaramente:
  • cosa funziona bene nell'insieme delle domande
  • quali problemi sono presenti (domande, copertura, utilità)
  • come il modello generatore dovrebbe migliorare le domande nella
    prossima iterazione

- **Sii cattivo**, punta alla massima precision, basta anche solo una
  domanda mal formulata per invalidare l'intero set.

- SOLO ALLA FINE, dopo aver completato ragionamento e feedback, fornisci
  il verdetto finale.

Regola di validità:
- valid DEVE essere false se esistono problemi significativi di
  rispondibilità, copertura o utilità per la valutazione di un retriever.
- valid può essere true SOLO se l'insieme delle domande è già adatto,
  così com'è, a valutare un retriever in un sistema RAG.
```

B.4.3 Assegnazione dei chunk di riferimento

È il prompt della fase descritta nella sezione 6.1.4, eseguita separatamente per ciascuna strategia di suddivisione. Il documento viene fornito già partizionato, con ogni chunk preceduto dal proprio identificativo.

Codice B.11: Assegnazione dei chunk di riferimento.

```
Sei un esperto di retrieval per sistemi RAG su documentazione logistica
(WMS).

INPUT:
1) Un documento diviso in segmenti numerati, nel formato
   "CHUNK X: [testo]".
2) Una lista di domande generate a partire da questo documento. Ogni
   domanda ha un "question_id" numerico.

COMPITO:
Per ciascuna domanda, identifica l'insieme MINIMO e SUFFICIENTE di chunk
che contengono l'informazione necessaria per rispondere.

REGOLE:
- Single-hop -> un solo chunk_id (es. [2]).
- Multi-hop -> più chunk_ids (es. [1, 4]).
- Se la domanda NON è chiaramente rispondibile usando il documento,
  restituisci una lista vuota [].
- NON includere chunk che non contribuiscono direttamente alla risposta.
- Gli id dei chunk devono corrispondere ESATTAMENTE a quelli indicati nel
  formato "CHUNK X: ...".
- Restituisci UNA entry per ogni question_id presente nella lista di
  domande.

DOCUMENTO:
{full_document_text}

DOMANDE:
{questions_list}
```

B.4.4 Giudizio delle risposte

È il prompt della valutazione end-to-end descritta nella sezione 6.1.8. Nelle esecuzioni relative alla strategia gerarchica il prompt conteneva una nota aggiuntiva, posta prima della domanda, che avvertiva il giudice che i chunk della ground truth sono chunk figli, mentre al modello valutato ne viene passata la versione già fusa nel chunk padre.

Codice B.12: Giudizio della risposta generata.

```
Sei un esperto valutatore di risposte generate da modelli di linguaggio
(LLM) in ambito tecnico-logistico.
Il tuo compito è valutare la qualità e l'accuratezza della risposta
fornita dall'LLM a una domanda, basandoti esclusivamente sul ground truth
fornito.
L'obiettivo è valutare la qualità della risposta.
Il ground truth è un insieme di chunk di testo estratti da manuali
tecnici, procedure operative e documentazione ufficiale del sistema WMS,
che contengono le informazioni corrette per rispondere alla domanda.

La risposta dell'LLM deve essere valutata come segue:
- '1' se la risposta è corretta, pertinente e basata sui dati forniti
- '0' se la risposta è errata, fuorviante o incompleta rispetto al
  ground truth
- '-1' se la risposta indica esplicitamente che il modello non sa la
  risposta o non ha abbastanza informazioni per rispondere

DOMANDA UTENTE:
{query}
----------
GROUND TRUTH:
{ground_truth}
----------
RISPOSTA FORNITA DALL'LLM:
{answer}
----------
```

Bibliografia

[1] Akari Asai et al. «Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection». In: *Proceedings of ICLR*. 2024.

[2] Tom B. Brown et al. «Language Models are Few-Shot Learners». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[3] Chainlit Team. *Chainlit: Build Conversational AI Applications*. URL: `https://chainlit.io` (visitato il giorno 22/09/2026).

[4] Jianlv Chen et al. «BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation». In: *Findings of the Association for Computational Linguistics (ACL)*. 2024.

[5] Gordon V. Cormack, Charles L. A. Clarke e Stefan Büttcher. «Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods». In: *Proceedings of ACM SIGIR*. 2009.

[6] Deep Search Team, IBM Research. *Docling Technical Report*. 2024. arXiv: `2408.09869`.

[7] Jacob Devlin et al. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». In: *Proceedings of NAACL-HLT*. 2019.

[8] Docker Inc. *Docker Documentation*. URL: `https://docs.docker.com` (visitato il giorno 22/09/2026).

[9] Shahul Es et al. «RAGAs: Automated Evaluation of Retrieval Augmented Generation». In: *Proceedings of EACL: System Demonstrations*. 2024.

[10] Manuel Faysse et al. «ColPali: Efficient Document Retrieval with Vision Language Models». In: *Proceedings of ICLR*. 2025.

[11] Elias Frantar et al. «GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers». In: *Proceedings of ICLR*. 2023.

[12] Luyu Gao et al. «Precise Zero-Shot Dense Retrieval without Relevance Labels». In: *Proceedings of ACL*. 2023.

[13] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2023. arXiv: 2312.10997.

[14] Kelvin Guu et al. «REALM: Retrieval-Augmented Language Model Pre-Training». In: *Proceedings of ICML*. 2020.

[15] Zhengbao Jiang et al. «Active Retrieval Augmented Generation». In: *Proceedings of EMNLP*. 2023.

[16] Jeff Johnson, Matthijs Douze e Hervé Jégou. «Billion-Scale Similarity Search with GPUs». In: *IEEE Transactions on Big Data* 7.3 (2021), pp. 535–547.

[17] Vladimir Karpukhin et al. «Dense Passage Retrieval for Open-Domain Question Answering». In: *Proceedings of EMNLP*. 2020.

[18] Omar Khattab e Matei Zaharia. «ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT». In: *Proceedings of ACM SIGIR*. 2020.

[19] Woosuk Kwon et al. «Efficient Memory Management for Large Language Model Serving with PagedAttention». In: *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*. 2023.

[20] Patrick Lewis et al. «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[21] Ji Lin et al. «AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration». In: *Proceedings of Machine Learning and Systems (MLSys)*. 2024.

[22] Haotian Liu et al. «Visual Instruction Tuning». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2023.

[23] Jerry Liu. *LlamaIndex: Data Framework for LLM Applications*. URL: https://www.llamaindex.ai (visitato il giorno 22/09/2026).

[24] Nelson F. Liu et al. «Lost in the Middle: How Language Models Use Long Contexts». In: *Transactions of the Association for Computational Linguistics* 12 (2024), pp. 157–173.

[25] Yu A. Malkov e D. A. Yashunin. «Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.4 (2020), pp. 824–836.

[26] Tomas Mikolov et al. «Efficient Estimation of Word Representations in Vector Space». In: *Proceedings of ICLR (Workshop Track)*. 2013.

[27] Niklas Muennighoff et al. «MTEB: Massive Text Embedding Benchmark». In: *Proceedings of EACL*. 2023.

[28] Rodrigo Nogueira e Kyunghyun Cho. *Passage Re-ranking with BERT*. 2019. arXiv: 1901.04085.

[29] Long Ouyang et al. «Training Language Models to Follow Instructions with Human Feedback». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.

[30] Niels Provos e David Mazières. «A Future-Adaptable Password Scheme». In: *Proceedings of the USENIX Annual Technical Conference*. 1999.

[31] Qdrant Team. *Qdrant: Vector Database and Vector Similarity Search Engine*. URL: https://qdrant.tech (visitato il giorno 22/09/2026).

[32] Qwen Team. *Qwen3 Technical Report*. 2025. arXiv: 2505.09388.

[33] Alec Radford et al. «Learning Transferable Visual Models From Natural Language Supervision». In: *Proceedings of ICML*. 2021.

[34] Nils Reimers e Iryna Gurevych. «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks». In: *Proceedings of EMNLP-IJCNLP*. 2019.

[35] Stephen Robertson e Hugo Zaragoza. «The Probabilistic Relevance Framework: BM25 and Beyond». In: *Foundations and Trends in Information Retrieval* 3.4 (2009), pp. 333–389.

[36] Gerard Salton, Anita Wong e Chung-Shu Yang. «A Vector Space Model for Automatic Indexing». In: *Communications of the ACM* 18.11 (1975), pp. 613–620.

[37] Nandan Thakur et al. «BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models». In: *NeurIPS Datasets and Benchmarks Track*. 2021.

[38] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971.

[39] Ashish Vaswani et al. «Attention Is All You Need». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[40] Yiheng Xu et al. «LayoutLM: Pre-training of Text and Layout for Document Image Understanding». In: *Proceedings of ACM SIGKDD*. 2020.

[41] Huaixiu Steven Zheng et al. «Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models». In: *Proceedings of ICLR*. 2024.

[42] Lianmin Zheng et al. «Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena». In: *NeurIPS Datasets and Benchmarks Track*. 2023.

Nota sull'utilizzo dell'intelligenza artificiale

Mi sono avvalso di strumenti di intelligenza artificiale generativa per migliorare la sintassi e la chiarezza del testo.