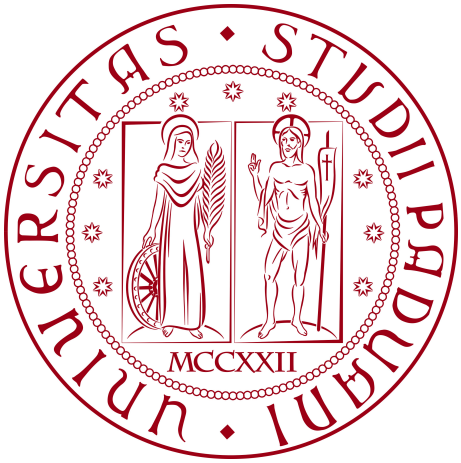
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Confronto tra Large Language Models nella Retrieval-Augmented Generation

Tesi di Laurea Triennale

Relatore
Prof. Tullio Vardanega

Laureando
Andrea Cecchin
Matricola 2034299

Anno Accademico 2023-2024

"Succede solo a chi ci crede"

— Stefano Pioli

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di *stage* dal laureando Andrea Cecchin, studente dell'Università degli Studi di Padova, presso l'azienda Sync Lab Srl. L'esperienza di *stage*, che ha avuto la durata di trecento ore complessive, è stata svolta sotto la supervisione dell'ingegner Fabio Pallaro, nella figura di tutor aziendale, e del professor Tullio Vardanega, tutor interno e relatore.

Il lavoro di *stage* si inserisce in un progetto di Ricerca e Sviluppo indirizzato alla valutazione e alla comparazione delle *performance* di diversi *Large Language Model*, quando questi sono utilizzati in *task* relative alla *Retrieval-Augmented Generation*. Il principale obiettivo del progetto è la realizzazione di un *benchmark*, appositamente strutturato per valutare un modello linguistico nello specifico caso d'uso.

Il documento è stato suddiviso in quattro capitoli, nei quali ho riportato la descrizione dell'azienda ospitante e il metodo lavorativo adottato durante lo *stage*, le attività pianificate e le tecnologie chiave del progetto, il suo svolgimento e le conclusioni finali. Nella stesura della tesi, ho evidenziato in corsivo le parole in lingue straniere. Inoltre, tutti i termini tecnici, possibilmente ambigui o di uso non comune menzionati sono stati segnalati con una "G" a pedice nella loro prima occorrenza, per essere poi spiegati nel Glossario posto al termine del documento.

Ringraziamenti

Desidero ringraziare la mia famiglia, i miei genitori e mia sorella Martina, per essere al mio fianco a festeggiare questo traguardo e per avermi reso la persona che sono.

Vorrei ringraziare il professor Tullio Vardanega, mio relatore, per la disponibilità, l'aiuto e il tempo che mi ha concesso seguendomi questi mesi durante la realizzazione del progetto e la stesura di questa tesi.

Desidero poi ringraziare l'ingegner Fabio Pallaro per avermi permesso di fare questa esperienza di stage, durante la quale non ha mai mancato di farmi sentire apprezzato e valorizzato. Senza la sua disponibilità, professionalità e dedizione tutto questo non sarebbe stato possibile.

Infine, voglio ringraziare tutti i miei compagni di università, con cui ho vissuto questi tre anni fantastici a Padova. Un ringraziamento speciale è per Leonardo, con cui ho condiviso viaggi in treno, progetti di gruppo e ogni bel ricordo di questa esperienza.

Padova, 19 Luglio 2024 *Andrea Cecchin*

Indice

1 Contesto aziendale **1**

1.1 Introduzione all'azienda . . . 1
1.2 Way of Working . . . 4
1.2.1 Metodologia di lavoro . . . 4
1.2.2 Strumenti adottati . . . 5
1.2.2.1 Strumenti organizzativi . . . 5
1.2.2.2 Strumenti comunicativi . . . 6
1.3 Importanza dell'innovazione . . . 7

2 Introduzione al progetto **9**

2.1 L'idea di partenza . . . 9
2.2 Scopo dello stage . . . 10
2.3 Tecnologie di interesse . . . 10
2.3.1 Large Language Model . . . 10
2.3.1.1 Descrizione . . . 10
2.3.1.2 Funzionamento . . . 11
2.3.1.3 Limiti . . . 12
2.3.1.4 Modelli utilizzati . . . 12
2.3.2 Database vettoriale . . . 13
2.3.3 Retrieval-Augmented Generation . . . 14
2.3.3.1 Descrizione . . . 14
2.3.3.2 Funzionamento . . . 14
2.4 Pianificazione del lavoro . . . 17
2.4.1 Pianificazione settimanale . . . 17
2.4.2 Obiettivi . . . 19
2.4.3 Prodotti attesi . . . 20
2.5 Motivazione della scelta . . . 21

3 Svolgimento **23**

3.1 Prototipazione rapida . . . 23
3.1.1 Descrizione delle attività . . . 23

3.1.2 Tecnologie utilizzate . . . 23
3.1.3 Prodotti software . . . 25
3.1.3.1 Progettazione e sviluppo . . . 25
3.1.3.2 Funzionalità . . . 26
3.2 Analisi dei requisiti . . . 28
3.2.1 Attori . . . 28
3.2.2 Casi d'uso . . . 29
3.2.3 Requisiti . . . 32
3.3 Progettazione e codifica . . . 33
3.3.1 Tecnologie utilizzate . . . 33
3.3.2 Architettura . . . 34
3.3.3 Design pattern . . . 36
3.3.4 Interfaccia grafica . . . 38
3.4 Verifica e validazione . . . 41
3.5 Confronto dei modelli . . . 42
3.5.1 Esperimenti . . . 42
3.5.2 Risultati . . . 43

4 Conclusioni **49**
4.1 Valutazione retrospettiva . . . 49
4.2 Maturazione personale . . . 51
4.3 Università e mondo del lavoro . . . 52

Glossario **53**

Riferimenti **55**

Elenco delle figure

1.1 Motto di Sync Lab S.r.l. 1
1.2 Servizi offerti da Sync Lab S.r.l. 2
1.3 Processi attivi in Sync Lab S.r.l. 2
1.4 Prodotti di punta di Sync Lab S.r.l. 3
1.5 Interfaccia della mia bacheca Trello 5
1.6 Università italiane che collaborano con Sync Lab S.r.l. 7

2.1 Funzionamento di un Large Language Model 12
2.2 Funzionamento della data ingestion 15
2.3 Funzionamento della information retrieval 16
2.4 Funzionamento della generation 17
2.5 Pianificazione delle attività del progetto 20

3.1 Funzionamento dei prototipi backend 27
3.2 Attori individuati 28
3.3 Diagramma dei casi d'uso comuni per l'utente generico e l'admin . . . 29
3.4 Diagramma dei casi d'uso dell'admin 31
3.5 Layered Architecture implementata 35
3.6 Interfaccia grafica della pagina di benvenuto 38
3.7 Interfaccia grafica della pagina di chat 39
3.8 Interfaccia grafica della pagina di autenticazione 40
3.9 Interfaccia grafica della pagina dell'amministratore 40
3.10 Funzionamento del benchmark realizzato 43
3.11 Risposte corrette per modello 44
3.12 Penalità registrate per modello 44
3.13 Punteggio finale per modello 45
3.14 Similarità semantica media delle risposte corrette 45
3.15 Tempo di generazione medio delle risposte 45
3.16 Valori progressivi della similarità semantica delle risposte 46
3.17 Valori progressivi del tempo di generazione delle risposte 46

4.1 Grado di raggiungimento degli obiettivi del progetto 50

Elenco delle tabelle

2.1 Pianificazione delle attività settimanali del progetto 18

3.1 Requisiti funzionali del prototipo finale 32
3.2 Requisiti di vincolo del prototipo finale 33
3.3 Requisiti qualitativi del prototipo finale 33

4.1 Prodotti realizzati durante il progetto 50

Elenco dei codici sorgenti

3.1 Esempio di implementazione della Dependency Injection 36
3.2 Esempio di implementazione del pattern Builder 37
3.3 Implementazione del pattern Higher-Order Components 38

Capitolo 1

Contesto aziendale

1.1 Introduzione all'azienda

L'esperienza di *stage* è avvenuta presso la sede padovana di Sync Lab. *Software house* fondata a Napoli nel 2002, ha da sempre fatto l'attenzione per la ricerca e l'innovazione il proprio punto di forza. Oltre alle già citati sedi di Napoli e Padova, l'azienda è presente anche nelle città di Roma, Milano, Verona e Como. Con più di trecento dipendenti in tutta Italia, Sync Lab conta più di centocinquanta tra clienti e filiali servite di diversi mercati, tra i quali aziende di rilievo come Sky, Telecom Italia, Enel e Trenitalia, oltre ad enti pubblici statali come il Ministero dell'Economia e delle Finanze e Rai.[16]

Figura 1.1: Motto di Sync Lab S.r.l.
Fonte: www.synclab.it

L'elevata varietà dei clienti serviti dimostra la forte competenza dell'azienda nel realizzare prodotti e soluzioni per diversi mercati, passando dal settore energetico, logistico e industriale fino a quello sanitario e finanziario. Operando in settori diversi, Sync Lab possiede elevate conoscenze e competenze in domini tecnologici differenti, che continua ad affinare e consolidare nel tempo per continuare ad offrire servizi di consulenza, presentati nella figura 1.2, di eccellenza e al passo con la costante innovazione tecnologica.
Con gli anni l'azienda è divenuta una tra i principali *System Integrator* a livello nazionale, offrendo ai propri clienti le competenze tecnologiche necessarie per intraprendere il percorso della trasformazione digitale nel proprio *business*.

Figura 1.2: Servizi offerti da Sync Lab S.r.l.
Fonte: www.synclab.it

Durante la fornitura dei propri clienti, Sync Lab adotta l'utilizzo di specifici *stack* tecnologici moderni e adatti ad ogni specifico progetto, con cui realizzare i prodotti richiesti dando loro architetture robuste e ottimizzate per il caso d'uso.
L'azienda basa le proprie attività di sviluppo sull'utilizzo del linguaggio di programmazione *Java*, utilizzato in coppia a *Spring*. Questa specifica struttura tecnologica, altamente utilizzata per lo sviluppo di applicazioni *web*, permette all'azienda di realizzare applicativi basati su servizi *REST*. Per interfacciarsi al *backend* dei propri applicativi, l'azienda utilizza il linguaggio *TypeScript* per sviluppare il *frontend* dei prodotti, affiancando ad esso il framework$_G$ *Angular* con il quale ottenere un prodotto veloce ed efficiente.
I *software* realizzati con queste tecnologie sono altamente robusti, modulari e facilmente manutenibili, facilitando le attività del processo di manutenzione del prodotto. Infatti, qualora richiesto dal cliente, l'azienda offre un servizio di manutenzione del *software* lungo tutto il suo ciclo di vita, assicurando il corretto funzionamento dell'applicativo e la completa soddisfazione di eventuali nuovi requisiti, emersi nel tempo.

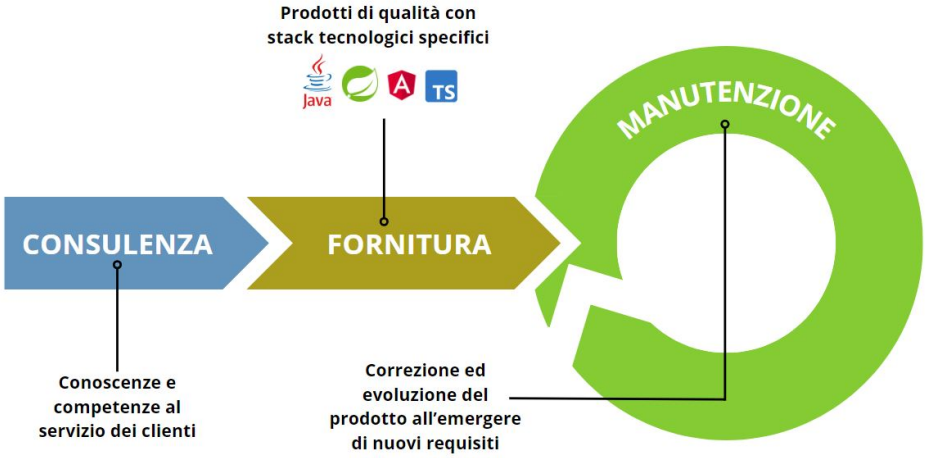


Figura 1.3: Processi attivi in Sync Lab S.r.l.

Un altro aspetto che valorizza e contraddistingue i prodotti di Sync Lab è l'elevato *standard* di qualità adottato nella loro realizzazione, assicurata e accertata da varie certificazioni ISO_G attribuite all'azienda. Alcuni dei prodotti di punta dell'azienda per qualità e successo, presentati anche in figura 1.4, sono di seguito riportati.

SynClinic: sistema *software* integrante tutte le funzionalità necessarie per la gestione informatizzata di tutti i processi clinico amministrativi delle strutture sanitarie. Grazie a questa soluzione applicativa è possibile gestire i dati clinici dei pazienti, i processi di prescrizione e approvvigionamento di farmaci, i processi di accettazione, dimissioni e trasferimento di pazienti in un'unica applicazione.

DPS 4.0: *web application* per la gestione della privacy di dati, in conformità al Regolamento Generale per la Protezione dei Dati e agli *standard* da esso imposti. Come la maggior parte dei prodotti realizzati da Sync Lab, questa applicazione è stata realizzata in *Java*, applicando i migliori paradigmi di programmazione per integrare il *backend* con un'interfaccia grafica prodotta con *TypeScript* e *Angular*.

Wave: dall'acronimo di *Wide Area Videosurveillance Environment*, prodotto *software* realizzato a partire da un progetto del dipartimento di Ricerca e Sviluppo dell'azienda, permette di integrare le informazioni provenienti dai Sistemi Informativi Territoriali per espandere le capacità di una rete di videosorveglianza.

Figura 1.4: Prodotti di punta di Sync Lab S.r.l.
Fonte: www.synclab.it

1.2 Way of Working

1.2.1 Metodologia di lavoro

In linea con l'esperienza e lo stile innovativo proprio dell'azienda, Sync Lab adotta un modello di sviluppo Agile$_G$, con l'intento di monitorare costantemente e gestire flessibilmente l'avanzamento dei progetti interni. Secondo i principi dati da questa metodologia di lavoro[15], l'azienda lavora a stretto contatto con gli stakeholder$_G$ basando lo sviluppo dei progetti sui *feedback* ottenuti e pianificando il lavoro da fare in modo flessibile e dinamico.
Più precisamente, Sync Lab utilizza il modello di sviluppo *Scrum*, che prevede la suddivisione del lavoro di un progetto in più periodi dalla durata ristretta, in genere da una a quattro settimane. Ogni periodo di incremento, denominato *sprint*, prevede delle attività che devono essere svolte per raggiungere degli obiettivi prefissati: l'insieme di tutte le attività previste identifica il *backlog* di un progetto. Al termine di ogni *sprint* avviene la *sprint review*, dove viene analizzato il grado di avanzamento del progetto rispetto le attese, e la *sprint retrospective*, nella quale il team esamina la qualità del proprio *way of working* e interviene per migliorarlo. Dopo aver pianificato il lavoro per lo *sprint* successivo secondo le attività contenuto nel *backlog*, il progetto prosegue con uno nuovo *sprint*.[5]

L'adozione della metodologia *Agile* da parte dell'azienda è di fondamentale importanza per i processi di consulenza, fornitura e manutenzione che offre ai propri clienti, in quanto il costante dialogo con gli *stakeholder* è componente irrinunciabile di ogni progetto. Infatti, la stretta collaborazione e il ricercato dialogo con i propri clienti permettono a Sync Lab di ottenere indicazioni precise sui requisiti e bisogni emergenti di ogni prodotto, permettendo all'azienda di agire e pianificare il lavoro in risposta ai pareri ricevuti dai committenti. Inoltre, in quanto l'obiettivo aziendale principale è la soddisfazione del cliente, l'interazione costante nel tempo consentita da questa metodologia permette di conoscere fin da subito se il prodotto in realizzazione incontra il gradimento atteso.

Il metodo *Agile* dell'azienda si è rispecchiato nell'esperienza di *stage* con la definizione, tramite opportuni strumenti organizzativi sotto riportati, di *sprint* dalla durata settimanale. Ogni *sprint* comprendeva specifici obiettivi da raggiungere e, al termine di ogni periodo incrementale, ho sostenuto dei colloqui di avanzamento con il mio tutor aziendale, in pieno stile *sprint review*. Questo per accertare l'effettivo avanzamento del progetto secondo le attese espresse dal mio Piano di Lavoro.

1.2.2 Strumenti adottati

Nelle attività giornaliere del progetto, sono stati utilizzati specifici strumenti organizzativi e comunicativi già adottati dall'azienda ospitante.

1.2.2.1 Strumenti organizzativi

Trello: *software* per la gestione e il monitoraggio dello stato di avanzamento del progetto, condiviso con il mio tutor aziendale. Utilizzato dall'azienda per tenere traccia del flusso delle attività di ogni progetto di *stage*, divide le attività di un progetto in quattro distinte categorie:

- **Da fare:** contiene tutte le attività che devono essere eseguite per terminare il progetto corrente. Rappresenta quindi il mio *backlog*.
- **In esecuzione:** mostra l'attività attualmente in esecuzione da parte dello stagista, da completare entro la fine dello *sprint* settimanale. Solo un'attività alla volta può essere in esecuzione, così da limitare il *work-in-progress* secondo i principi della metodologia *Agile*.[5]
- **Verifica:** mostra la lista delle attività che sono state concluse ma che non hanno ancora ricevuto l'approvazione. Le attività in stato di verifica vengono considerate terminate solo a seguito dello *sprint review*.
- **Terminati:** contiene tutte le attività completate e approvate al termine di uno *sprint*. Per poter considerare concluso un progetto, tutte le attività inizialmente previste nella colonna "da fare" devono trovarsi qui.

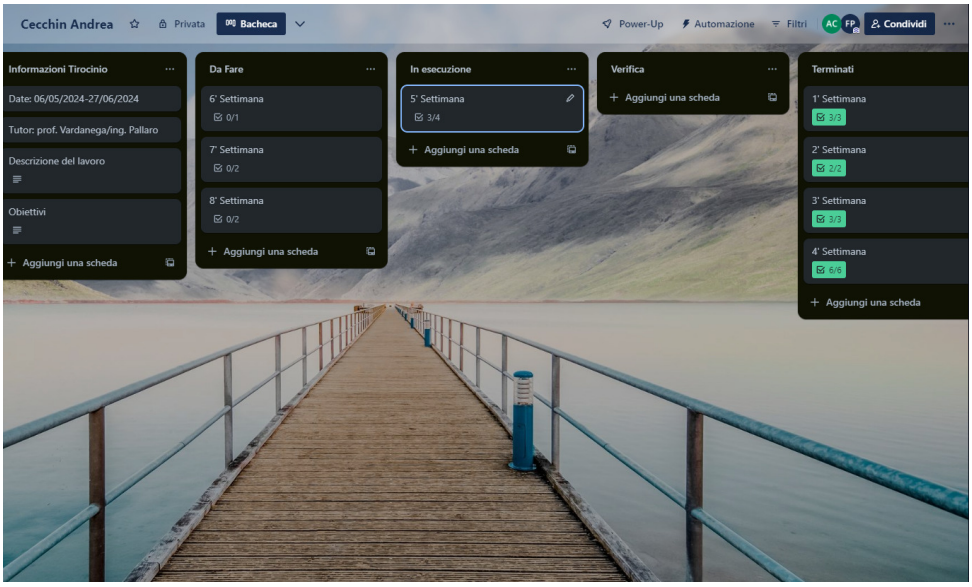


Figura 1.5: Interfaccia della mia bacheca Trello

Nello specifico caso del mio progetto di *stage*, come anche visibile in figura 1.5, le attività da fare erano nominate con il numero della settimana di lavoro a cui si riferivano. Ogni attività settimanale comprendeva delle sotto-attività, definite dal Piano di Lavoro dello *stage*, che dovevano essere interamente completate per poter mettere l'attività nello stato di "verifica". Completata una sotto-attività apponevo una spunta nella *checklist* relativa, e una volta completata un'attività spostavo manualmente la scheda sullo stato successivo, per segnalare l'avanzamento al mio tutor. L'approvazione di un'intera attività avveniva a seguito dell'incontro settimanale con il tutor aziendale, lo *sprint review*, nel quale mostravo l'avanzamento del mio lavoro e dimostravo l'effettivo completamento di tutte le sotto-attività previste.
L'utilizzo di questo strumento mi ha permesso di monitorare il mio ritmo di avanzamento rispetto le attese, dandomi la possibilità di riorganizzare le attività del progetto. In particolare, grazie al controllo del progresso nel progetto tramite *Trello*, ho potuto subito individuare e quantificare l'anticipo rispetto alle aspettative iniziali, così da aggiungere nuove attività al mio Piano di Lavoro e aumentare il valore aggiunto, sia personale che per l'azienda, del mio *stage*.

Google Calendar: sistema di calendarizzazione offerto da *Google*, è utilizzato in tutte le sedi di Sync Lab per pianificare e organizzare le attività lavorative. Attraverso questo strumento, sono evidenziati i giorni di apertura della sede di Padova al personale, permettendo la pianificazione della propria presenza. Infatti, l'azienda adotta l'*hybrid working*, una modalità di lavoro che permette ai propri dipendenti di lavorare da casa quando la sede aziendale non è aperta. La mia esperienza di *stage* è stata quindi svolta in modalità ibrida, presenziando nei soli giorni di apertura della sede. Ciò non ha in alcun modo influito sul grado di comunicazione e collaborazione con il tutor aziendale e gli altri stagisti, grazie agli strumenti comunicativi adottati.

1.2.2.2 Strumenti comunicativi

Discord: piattaforma di messaggistica istantanea che permette la comunicazione tra persone con messaggi di testo, chiamate vocali e video chiamate. L'azienda dispone di un *server* privato nel quale *chat* e canali vocali sono suddivisi per tematiche e sedi aziendali. Grazie all'adozione di questo strumento, è stato possibile mettermi in contatto con il tutor aziendale e gli altri stagisti per scambiare, in modo facile e veloce, informazioni utili al progetto.
Oltre ad essersi rivelato un importante mezzo comunicativo, *Discord* è stata

un'utilissima fonte di informazioni e risorse didattiche, grazie ai canali aziendali appositamente dedicati a raccogliere documentazioni, *link* e video *tutorial* per le principali tecnologie e argomenti utili allo *stage*.
Con l'adozione di *Discord* ho potuto semplificare la fase di studio e formazione autonoma sulle tecnologie del progetto, avendo accesso a fonti informative già evidenziate da stagisti precedenti e dal mio tutor aziendale.

1.3 Importanza dell'innovazione

Il forte e costante interesse di Sync Lab per le più moderne tecnologie informatiche, dall'utilizzo del *Metaverso* e dei sistemi olografici tridimensionali in ambito pubblicitario e *marketing* fino alle più recenti novità sulle *Blockchain*, dimostra la propensione verso l'innovazione che caratterizza tutte le attività aziendali.
Per mantenere le conoscenze e competenze aziendali al passo con la progressiva innovazione del mondo informatico e *tech*, Sync Lab svolge attività di Ricerca e Sviluppo nel proprio dipartimento interno fin dal 2003, lavorando a stretto contatto con aziende ed enti governativi. Un esempio è *Fast Ride*, progetto di ricerca sostenuto e realizzato in collaborazione all'Unione Europea e la Repubblica Italiana, che mira alla realizzazione di un sistema informatico per l'integrazione di tecnologie e sensori *Internet of Things* per l'efficientamento del trasporto pubblico nazionale.

Figura 1.6: Università italiane che collaborano con Sync Lab S.r.l.

L'importanza data allo studio e all'introduzione di nuove tecnologie e processi ha spinto l'azienda a espandere le proprie attività di ricerca, prendendo stretto contatto con il mondo accademico e stipulando numerose collaborazioni con le maggiori università italiane ed internazionali.
Aprendo i propri progetti interni con queste realtà universitarie, l'azienda è in grado di potenziare le attività di Ricerca e Sviluppo, aggiungendo al proprio lavoro le competenze e le conoscenze di ricercatori e scienziati da tutta Italia.
Inoltre, lo stretto legame con il mondo accademico fa sì che Sync Lab si avvalga degli *stage* per sostenere le proprie attività di ricerca, organizzando e proponendo progetti di tirocinio finalizzati all'esplorazione di nuove tematiche e alla sperimentazione delle tecnologie relative all'ambito di interesse, attraverso il lavoro dello studente. Grazie a questa strategia, l'azienda è in grado di massimizzare le competenze su una determinata innovazione, riducendo i costi di ricerca e offrendo agli stagisti la possibilità di acquisire nuove conoscenze interfacciandosi al mondo del lavoro.

Capitolo 2

Introduzione al progetto

2.1 L'idea di partenza

L'Intelligenza Artificiale è la tematica attualmente più popolare e di maggior interesse nell'intero panorama mondiale della *computer science*. Sempre più piattaforme ed applicazioni offrono servizi che integrano la *Generative AI*, ovvero sistemi di Intelligenza Artificiale capaci di generare testi, immagini e altri *media*, fino alla più fedele riproduzione di una voce umana.

Compreso l'enorme potenziale di questa innovazione, con moltissimi ambiti di utilizzo possibili, Sync Lab si sta interessando alle tecnologie utili alla realizzazione di *chatbot* così da permetterne l'integrazione di questi in altri prodotti aziendali. Ad esempio, l'azienda sta programmando la realizzazione di *avatar* olografici, in grado di sostituire il personale *front-desk* incaricato di fornire informazioni a chi le richiede. Per questo motivo, in accordo alla propensione per l'innovazione già discussa in precedenza, l'azienda sta svolgendo, con la partecipazione di stagisti universitari, una serie di progetti indirizzati all'individuazione e allo studio delle tecnologie che permettono l'integrazione della *AI* generativa, in varie tipologie di prodotti *software*.

L'interesse principale di Sync Lab è rivolto alla possibilità di definire il dominio di conoscenze del modello alla base del *chatbot*, affinché esso sia in grado di generare delle risposte a domande relative ad un determinato contesto. Questo dominio conoscitivo può essere definito attraverso dei semplici documenti testuali, dai quali i modelli di *Artificial Intelligence* denominati *Large Language Model* (definizione nella sezione 2.3.1) possono estrarre le informazioni rilevanti per la formulazione della risposta. Diventa quindi fondamentale per l'azienda individuare quali sono i modelli con le *performance* migliori nello specifico caso d'uso: è da questa necessità che deriva lo scopo del mio progetto.

2.2 Scopo dello stage

Le prestazioni di un *chatbot*, sia nell'individuare le informazioni utili che nel generare una risposta a partire da esse, sono quindi determinate da quelle del modello integrato. Questo progetto di *stage* ha quindi lo scopo di misurare e confrontare le *performance* di vari modelli quando utilizzati in un sistema integrante la *Retrieval-Augmented Generation*, o *RAG* (definizione nella sezione 2.3.3), così da individuare il modello linguistico più adatto al caso d'uso.

Per fare ciò, è stato prima necessario individuare, studiare e testare dei modelli di Intelligenza Artificiale, oltre alle tecnologie con cui interfacciarsi ad essi. Successivamente, è stata richiesta la progettazione e la realizzazione di vari prototipi, sia *backend-only* che *fullstack*, implementanti tutte le funzionalità principali della *RAG*: definizione del dominio conoscitivo del modello, con l'aggiunta e la rimozione dei documenti, e interrogazione dei modelli stessi, interfacciandosi ad un *chatbot*. Infine, sono state effettuate le operazioni di *benchmarking* dei modelli testati, valutando le *performance* registrate e confrontandole tra loro. Utilizzando i risultati ottenuti da queste valutazioni, considerando anche altre informazioni reperibili dalla loro documentazione tecnica, è stato possibile identificare i modelli più adatti all'utilizzo desiderato dall'azienda.

Rispetto a tutti gli altri progetti di tirocinio organizzati dall'azienda sulla *Generative AI*, lo scopo del mio *stage* non prevede come obiettivo primario la realizzazione di un prodotto *software* finito, bensì la valutazione dei modelli linguistici. Il mio lavoro diventa quindi di importanza strategica per Sync Lab, in quanto i risultati che derivano da esso influenzeranno direttamente ogni altro progetto interno aziendale che preveda l'utilizzo di *Large Language Model*.

2.3 Tecnologie di interesse

2.3.1 Large Language Model

2.3.1.1 Descrizione

Un *Large Language Model* (*LLM*), o modello linguistico di grandi dimensioni, è un modello computazionale di Deep Learning$_G$, costituito da una rete neurale artificiale, addestrato per svolgere *task* di *Natural Language Processing*, ovvero processare, comprendere e generare dati in linguaggio naturale. Deve il suo nome alla capacità

di comprendere e riprodurre il linguaggio umano, cogliendo il legame sintattico e semantico delle parole, e al numero elevato di parametri (nell'ordine dei miliardi) che costituiscono la sua rete neurale.
I *LLM* apprendono il significato e le relazioni delle parole, la grammatica e la struttura semantica del linguaggio umano grazie all'addestramento su un enorme *dataset*. In questa fase, denominata *pre-training*, il modello processa un ampio quantitativo di dati testuali, come libri, articoli e siti *web*, in modo completamente autonomo tramite unsupervised learning$_G$.
Successivamente, i modelli vanno incontro alla fase di *fine-tuning*, nella quale ricevono *dataset* da processare mirati a quella che sarà la *task* primaria del modello. Ad esempio, se l'obiettivo è produrre un modello per utilizzarlo in ambito medico, esso sarà poi addestrato con un'enorme quantità di dati e testi clinici, andando a specializzare il dominio conoscitivo del modello stesso.[10]

2.3.1.2 Funzionamento

Tutti i principali modelli linguistici adottano un'architettura conosciuta con il nome di *Transformer*[8]. Questa particolare tipologia di rete neurale è composta da due principali elementi, l'*encoder* e il *decoder*, che permettono l'elaborazione di un *input* testuale per generare un *output* in linguaggio umano.
I *Large Language Model* non sono in grado di elaborare un testo a partire dalle parole grezze, ma operano su dei valori numerici. Per questo motivo, l'*input* dato al modello, che può essere una frase da tradurre o un testo da riassumere, viene prima suddiviso in *token*, ovvero nelle unità fondamentali processabili dal modello. I *token*, che posso corrispondere a parole, frammenti di parole o solo caratteri, sono associati a dei valori numerici stabiliti dal vocabolario$_G$ utilizzato dal modello. Ogni *token*, in base al valore numerico ad esso associato, è successivamente convertito in un vettore di numeri reali multidimensionale: parole semanticamente correlate risulteranno vicine nello spazio multidimensionale ottenuto.
Una volta convertito l'*input* in vettori numerici, questi sono passati all'*encoder* che, processando l'informazione nei vari *layer* che lo compongono, comprende le proprietà del testo in *input* e genera delle rappresentazioni complesse in grado di definire contesto e significato di ogni *token*.
Queste rappresentazioni astratte rappresentano l'*input* del *decoder*, il quale ha il compito di processare ulteriormente le informazioni ottenute trasformandole nei vettori multidimensionali finali: questi saranno poi utilizzati per la generazione dell'*output*. La risposta è generata dal *decoder* in modo iterativo e autoregressivo, in quanto il modello seleziona il *token* più probabile contestualmente a quelli che già compongono la sequenza di *output*.

Una volta ottenuto l'intero *output*, esso viene tradotto in linguaggio umano in base al medesimo vocabolario utilizzato in fase di *tokenizzazione* e finalmente restituito come risposta del *LLM*.[8][17]

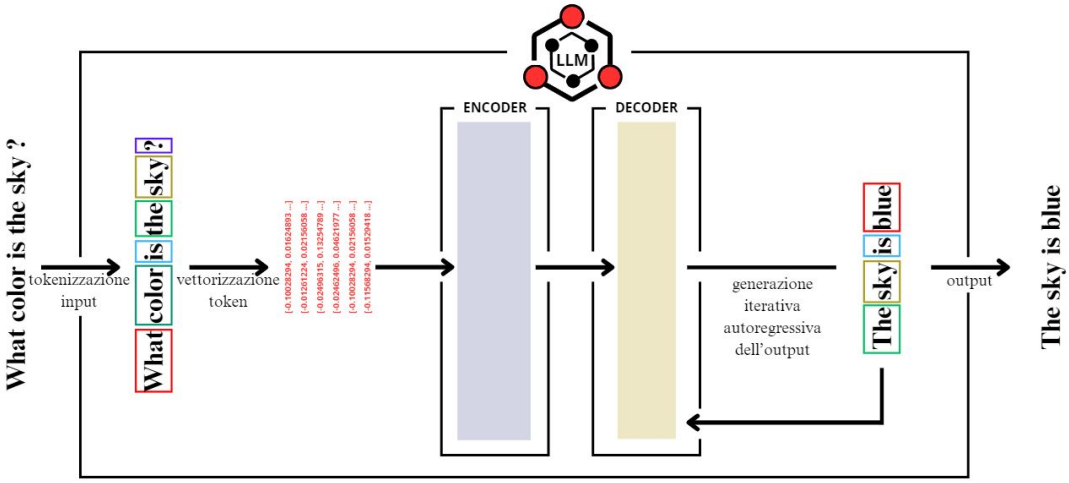


Figura 2.1: Funzionamento di un Large Language Model

2.3.1.3 Limiti

Nonostante le grandi capacità, questa tecnologia non è esente da limiti. Innanzitutto, le *performance* dei modelli sono strettamente legate alla loro dimensione e alla qualità del *dataset* usato in fase di *training*. Aumentando il numero dei parametri che compongono la rete neurale del modello, non aumenterà solo la sua prestazione ma anche il suo costo in termine di risorse hardware, economiche e temporali necessarie per il suo addestramento e utilizzo. Inoltre, tutti i bias$_G$ presenti nei dati di addestramento saranno ereditati dal modello.
Un altro grande limite dei *LLM* è dato dal dominio conoscitivo del modello. Se il modello non ha integrato alcun metodo per il recupero di informazioni aggiuntive, come la ricerca di informazioni sul *web*, esso sarà in grado di fornire risposte esclusivamente a domande inerenti al *dataset* utilizzato per il suo addestramento. È questo il limite che tenta di risolvere la *Retrieval-Augmented Generation*, tecnica di *information retrieval* oggetto di studio di questo progetto di *stage*.

2.3.1.4 Modelli utilizzati

Durante le varie fasi del progetto sono stati utilizzati e confrontati cinque modelli linguistici, sia on-premise$_G$ in locale che come servizio *cloud*. Il primo *LLM* installato e utilizzato in locale è Phi-3 Mini, un modello progettato e pubblicato da *Microsoft*, scelto per le buone *performance* nei principali *benchmark* nonostante i soli 3,8 miliardi di parametri della sua architettura. Le sue dimensioni ridotte gli permettono di

richiedere poche risorse *hardware* nel funzionamento, tanto da poter essere eseguito anche in uno *smartphone*.[7] L'altro modello utilizzato in locale è Llama3 di *Meta*, nella sua versione con 8 miliardi di parametri, che ho scelto di valutare in questo progetto perché presentato come il più capace modello *open-source* disponibile dalla stessa azienda produttrice.[13]
Per quanto riguarda i modelli in *cloud*, l'attenzione dell'azienda e di questo progetto è stata rivolta principalmente a Gemini 1.5 Pro. Modello linguistico in sviluppo da *Google* e pubblicato in versione *preview*, è il *LLM* con la più ampia *context window* mai prodotto: è infatti in grado di processare una richiesta lunga fino a un milione di *token*. Ha fatto inoltre registrare prestazioni elevatissime nei pochi *benchmark* in cui è stato testato.[2] Gemini è considerato uno dei modelli di seconda generazione di *Google*. Per comparare le diverse *performance* tra modelli di generazioni differenti, un ulteriore modello utilizzato nel progetto è PaLM 2 Bison, *LLM* della prima generazione dell'azienda statunitense.[1]
L'ultimo modello utilizzato in questo progetto è GPT-3.5 Turbo di *OpenAI*.[12] Conosciuto globalmente per essere il modello implementato su *ChatGPT*, è stato scelto per essere testato e confrontato in quanto uno dei più popolari e utilizzato da sviluppatori e aziende di tutto il mondo.

2.3.2 Database vettoriale

Un *database* vettoriale, o *vector database*, è una base di dati in grado di memorizzare dei vettori multidimensionali detti *embedding*. Questi vettori di numeri reali, calcolati da particolari modelli detti *Embedding Model*, rappresentano in uno spazio multidimensionale una o più frasi, similmente a quanto accade con i *token* nei *LLM*. Frasi relative allo stesso argomento, semanticamente simili o comunque correlate hanno la particolarità di essere vicine nello spazio multidimensionale. Al contrario, frasi relative a contesti del tutto differenti saranno tra loro distanti.

I *database* vettoriali, oltre a memorizzare gli *embedding*, sono ottimizzati nell'individuare, dato un vettore come *query* di ricerca, gli altri *embedding* più simili ad esso. Questa caratteristica li rende particolarmente utili nel recuperare informazioni utili e rilevanti ad un dato contesto.
In particolare, nella *Retrieval-Augmented Generation* i *vector database* contengono tutti i vettori multidimensionali generati a partire dal dominio conoscitivo aggiuntivo del modello. La domanda che si vuole porre al modello sarà *embeddizzata* e utilizzata per ricercare e recuperare, dalla base di dati, le informazioni utili alla generazione della risposta.

2.3.3 Retrieval-Augmented Generation

2.3.3.1 Descrizione

La *Retrieval-Augmented Generation*[3] (*RAG*) è un metodo di *information retrieval* applicato alla *Generative AI*, utilizzato per ampliare il dominio conoscitivo di un modello con informazioni e conoscenze non presenti nel *dataset* di addestramento. Qualsiasi informazione testuale o convertibile in testo, come documenti, audio, video e immagini, può essere utilizzata come fonte aggiuntiva per il *LLM*.

Quando si interroga un modello, nel momento della generazione della risposta vengono fornite le informazioni provenienti dalla fonte aggiuntiva. In questo modo, il modello è in grado di generare una risposta che tiene conto non solo della fase di *pre-training* e *fine-tuning*, ma anche delle informazioni aggiuntive recuperate.

2.3.3.2 Funzionamento

Il processo di *Retrieval-Augmented Generation* si divide in tre fasi distinte: *data ingestion*, *information retrieval* e *generation*. Nella prima avviene la definizione del dominio conoscitivo aggiuntivo del modello linguistico, processando le informazioni e convertendole ad un formato che permette il loro recupero. Durante la fase di *information retrieval*, le informazioni rilevanti alla richiesta fatta al modello sono recuperate e utilizzate nella *generation* per comporre la risposta finale.[18]

Data ingestion

La *data ingestion* è la prima delle tre fasi della *RAG* e consiste nella composizione del dominio conoscitivo aggiuntivo del modello linguistico. Qui le informazioni sono processate e convertite in vettori di *embedding*, per poi essere recuperate solo in caso di necessità durante la *retrieval*.
Il processo di *ingestion* inizia con la conversione in testo di tutti i *file* e dati che si vuole fornire al modello. I documenti testuali, come *file PDF* e *DOCX*, sono quindi processati per estrarne il contenuto. I *file* audio e video sono trascritti, mentre dalle immagini viene recuperato il testo alternativo che le descrive.
Una volta raccolto il testo dai documenti, esso viene suddiviso in frazioni più piccole, denominate *chunk*. Il testo può essere diviso in parti a lunghezza fissa, come il numero di caratteri o di *token*, oppure variabile, dividendo il testo per frasi, paragrafi o intere pagine. Ogni frazione di testo viene convertito in *token* e processato da un *Embedding Model*, dal quale sarà generato l'*embedding* relativo: attraverso questa rappresentazione, sarà possibile calcolare la similarità semantica delle informazioni

con la domanda da porre al *LLM*, confrontando la distanza delle frasi nello spazio multidimensionale risultante, così da individuare e recuperare le sole informazioni rilevanti.
L'*ingestion* termina con la memorizzazione in un *database* vettoriale degli *embedding* generati. In particolare, ogni *record* presente nel *vector database* conterrà il *chunk* e il vettore associato.

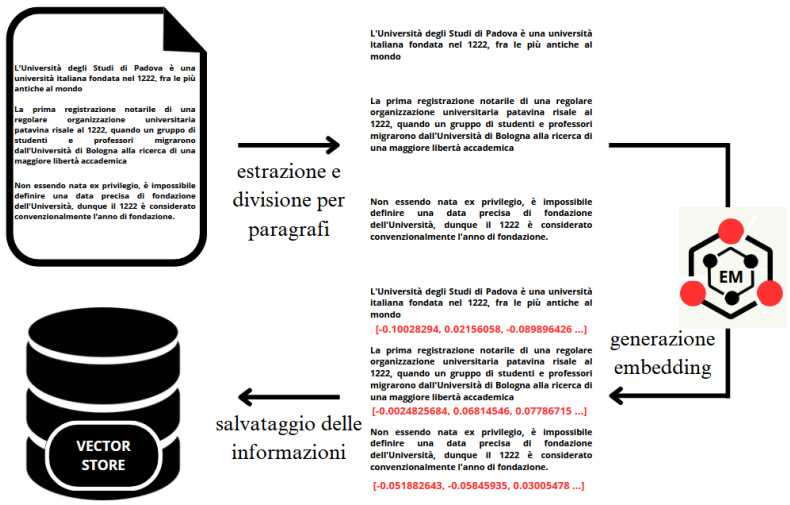


Figura 2.2: Funzionamento della data ingestion

Information retrieval

Durante la fase di *information retrieval*, la seconda della *Retrieval-Augmented Generation*, avviene il recupero delle informazioni utili dal dominio delle conoscenze aggiuntive. In particolare, il messaggio inviato al modello linguistico, che può essere una domanda o una *task* generica, viene processato dallo stesso *Embedding Model* utilizzato per produrre i vettori presenti nel *database* vettoriale e anch'esso convertito in *embedding*.
Il vettore della domanda ottenuto viene poi comparato con quelli presenti nel *vector store*, andando a calcolare la loro similarità semantica. Più precisamente, viene calcolata la *cosine similarity* dei due vettori, ovvero il coseno dell'angolo che intercorre tra i vettori nello spazio multidimensionale. Dati u e v vettori di *embedding* da confrontare, la similarità del coseno sarà calcolata con la seguente formula:

$$\text{Cosine Similarity}(u, v) = \cos(\theta) = \frac{u \cdot v}{||u|| \cdot ||v||}$$

dove θ è l'angolo tra i due vettori.
Il risultato di questo calcolo è utilizzato per definire la similarità semantica delle

frasi origine dei vettori: un valore vicino a 1, ottenuto con due vettori quasi allineati, indica che le frasi sono semanticamente simili e che probabilmente condividono la stessa tematica o lo stesso contesto. Un valore vicino allo 0, o addirittura negativo, indica invece una dissomiglianza semantica tra le frasi comparate, escludendo che i soggetti e le tematiche trattate da esse siano correlate.[6]
Basandoci su questi valori, è possibile identificare la presenza o meno di informazioni rilevanti nel contesto conoscitivo aggiuntivo fornito al modello, andando a recuperare dal *database* solo le informazioni che hanno registrato un valore di similarità elevato.

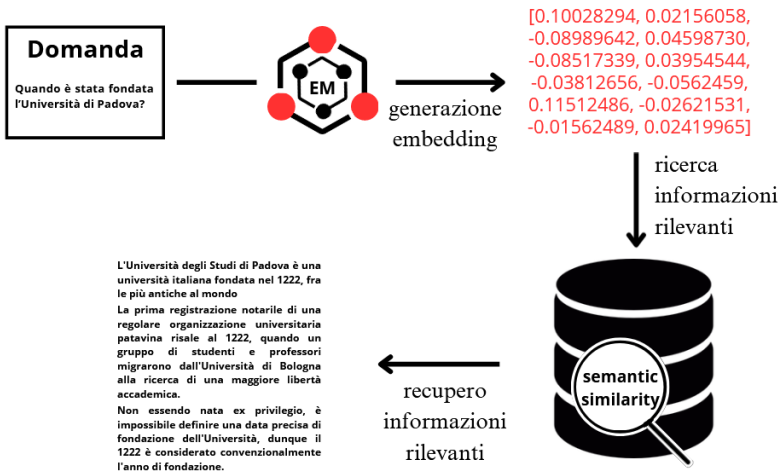


Figura 2.3: Funzionamento della information retrieval

Poter recuperare le sole informazioni rilevanti alla richiesta inviata permette di limitare la quantità di dati da mandare al modello. Questo risulta essere fondamentale, in quanto le prestazioni del *LLM* sono migliori al diminuire della quantità di informazioni che deve processare.

Generation

Come intuibile dal suo nome, durante la fase di *generation* viene completato il processo di *Retrieval-Augmented Generation* con la formulazione della risposta finale del modello. Recuperate le informazioni utili dal *database* vettoriale, il modello viene finalmente interrogato.
Il *prompt*, ovvero il testo in linguaggio naturale che specifica la richiesta da inoltrare al modello linguistico, viene composto a partire dalla richiesta originale, utilizzata nella *retrieval* per la ricerca dei contenuti rilevanti, e utilizzando le informazioni recuperate. In questa fase di formulazione della richiesta, definita in gergo *prompt engineering*, viene specificato al *LLM* l'attività che dovrà svolgere, le informazioni

aggiuntive che può utilizzare nell'esecuzione della *task* e altre eventuali modalità di risposta. Una volta composto, il *prompt* viene passato al modello, il quale genererà la risposta finale.

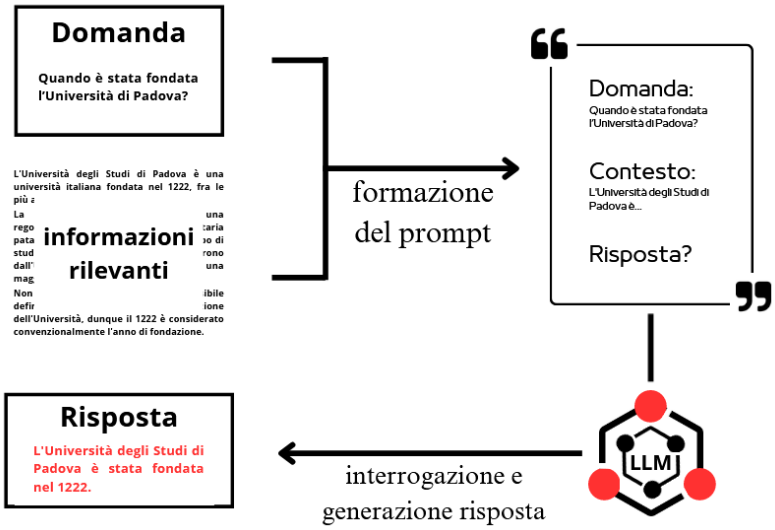


Figura 2.4: Funzionamento della generation

Nello specifico caso di questo progetto di *stage*, i *Large Language Model* sono utilizzati per formulare le risposte di un *chatbot* con accesso ai soli documenti caricati nel *vector database*. Per questo motivo, il *prompt* contiene la domanda da porre al modello, seguita dalle informazioni rilevanti recuperate, oltre alla specifica richiesta di generare delle risposte basandosi solo sulle informazioni fornite dal *prompt* stesso. Questa particolare formulazione è utile per valutare la capacità dei *LLM* di eseguire specifiche richieste, accertando che essi restituiscano una risposta corretta solo perché correttamente comprese ed elaborate le informazioni fornite come contesto utile, e non perché già presenti nel *dataset* del loro addestramento.

2.4 Pianificazione del lavoro

2.4.1 Pianificazione settimanale

L'intera pianificazione delle attività da svolgere durante il progetto, per una durata totale di trecento ore, è stata concordata con il mio tutor aziendale a seguito di un incontro tenutosi a una settimana dall'inizio dello *stage*. L'intera pianificazione è stata poi riportata in un documento denominato Piano di Lavoro, per poi essere successivamente approvato dal tutor interno.
Di seguito sono riportate le attività dello *stage*, secondo la suddivisione settimanale riportata nel documento di pianificazione.

Periodo	**Attività**
Prima settimana	Ripasso dei concetti di Metodologia *Agile/Scrum*
	Studio e formazione individuale sul linguaggio di programmazione *Java*
	Studio e formazione individuale sui principi della programmazione *SOLID* e della *clean programming*
	Studio e formazione individuale sui servizi *REST* e sui messaggi *JSON*
Seconda settimana	Studio e formazione individuale sui *Large Language Model* e sulla *Retrieval-Augmented Generation*
	Studio e formazione individuale sul *framework Java Spring*
Terza settimana	Individuazione, studio e formazione individuale su un *LLM* locale
	Progettazione di un prototipo che integra *Spring*, con l'utilizzo di un *LLM* locale, e le tecnologie per la *RAG*
	Implementazione di un prototipo che integra *Spring*, con l'utilizzo di un *LLM* locale, e le tecnologie per la *RAG*
Quarta settimana	Studio e formazione individuale sul modello Gemini di *Google*
	Progettazione di un prototipo che integra *Spring*, con l'utilizzo di Gemini, e le tecnologie per la *RAG*
	Implementazione di un prototipo che integra *Spring*, con l'utilizzo di Gemini, e le tecnologie per la *RAG*
Quinta settimana	Analisi dei requisiti di un prototipo finale con cui inserire i documenti di interesse e interagire con un *chatbot*
	Studio e formazione individuale su *React*
	Progettazione di un prototipo finale che integra *Spring*, con un frontend *React*, e le tecnologie per la *RAG*
	Implementazione di un prototipo finale che integra *Spring*, con un frontend *React*, e le tecnologie per la *RAG*
Sesta settimana	Implementazione nel prototipo finale della funzionalità di autenticazione, in base alle categorie di utente individuate nell'analisi dei requisiti
Settima settimana	Implementazione nel prototipo finale delle funzionalità di inserimento dei documenti
	Implementazione nel prototipo finale delle funzionalità del *chatbot*
Ottava settimana	Studio e formazione individuale sui metodi di *evaluation* dei *LLM*
	Analisi comparativa delle *performance* dei *LLM* impiegati

Tabella 2.1: Pianificazione delle attività settimanali del progetto

2.4.2 Obiettivi

Insieme al mio tutor aziendale ho anche concordato la definizione di una serie di obiettivi, da raggiungere durante il periodo del progetto. Essi sono stati suddivisi in base alla loro importanza, secondo le seguenti notazioni:

- O per gli obiettivi obbligatori, vincolanti in quanto obiettivo primario richiesto dall'azienda e dal tutor aziendale;
- D per gli obiettivi desiderabili, non vincolanti o necessari, ma dal riconoscibile valore aggiunto;
- F per gli obiettivi facoltativi, rappresentanti discreto valore aggiunto.

Di seguito sono riportati gli obiettivi individuati, divisi per categoria di rilevanza.

- Obbligatori:
  - O-1: Realizzazione di un prototipo *backend* che integra correttamente le tecnologie impiegate per la *RAG* con un *LLM* locale, utilizzando il *framework Java Spring*;
  - O-2: Realizzazione di un prototipo *backend* che integra correttamente le tecnologie impiegate per la *RAG* con il modello Gemini di *Google*, utilizzando il *framework Java Spring*;
  - O-3: Realizzazione di un prototipo finale, implementato con *Java Spring* e *React*, che permetta l'inserimento di documenti e la loro consultazione tramite l'interazione con un *chatbot*;
  - O-4: Ampia, esaustiva e precisa valutazione delle *performance* dei modelli implementati nel prototipo finale, con redazione di un documento di report.
- Desiderabili:
  - D-1: Implementazione nel prototipo finale di un terzo modello, GPT-3.5 di *OpenAI*, su cui svolgere le operazioni di *evaluation* e comparazione con gli altri due modelli obbligatori.
- Facoltativi:
  - F-1: Implementazione nel prototipo finale di un meccanismo di impostazione *real-time* del modello da utilizzare nelle operazioni.

2.4.3 Prodotti attesi

Al termine del progetto, era attesa la realizzazione di alcuni prodotti, sia *software* che documentali, per poter definire il profitto del progetto. Questi prodotti, anch'essi definiti nel Piano di Lavoro, sono i seguenti:

- Prototipo *software* implementato con *Java Spring*, che permette la *Retrieval-Augmented Generation* con un *Large Language Model* locale.
- Prototipo *software* implementato con *Java Spring*, che permette la *Retrieval-Augmented Generation* con Gemini.
- Prototipo finale implementato con *Java Spring* e *React*, che permette l'inserimento di documenti e la *Retrieval-Augmented Generation* su di essi, attraverso i modelli già utilizzati nei prototipi precedenti.
- Documento di report delle attività di analisi dei requisiti e di progettazione del prototipo finale. Tale documento dovrà riportare considerazioni sullo studio dei requisiti, dei casi d'uso, della progettazione dell'interfaccia grafica e del funzionamento del *backend*.
- Documento di report delle attività di analisi comparativa delle *performance* dei modelli. Tale documento dovrà riportare considerazioni su popolarità del modello, grado di completezza della documentazione, costo, risorse necessarie, velocità, correttezza delle risposte, frequenza di errori e altri parametri finalizzati all'*evaluation* delle performance.

Figura 2.5: Pianificazione delle attività del progetto

2.5 Motivazione della scelta

Il primo contatto con l'azienda ospitante è avvenuto durante StageIT, un evento a cadenza annuale che vede l'incontro dei laureandi dell'Università degli Studi di Padova con alcune delle aziende, del territorio e non, che offrono progetti di tirocinio. In quell'occasione, ho avuto modo di entrare in contatto con il mondo del lavoro per la prima volta, svolgendo vari colloqui con diverse realtà aziendali. Oltre ad aver subito avuto un'ottima impressione di Sync Lab, dopo un breve ma utile confronto con quello che sarebbe diventato il mio tutor aziendale, sono rimasto subito colpito dall'interesse esposto dall'azienda verso le tematiche dell'Intelligenza Artificiale e della *Retrieval-Augmented Generation*.

Grazie al progetto didattico del corso di Ingegneria del *Software*, svolto fino a qualche settimana prima dell'inizio dello *stage*, avevo già avuto modo di esplorare la tecnica di *information retrieval* alla base di questo progetto. Per questo motivo, ho ritenuto altamente formativo intraprendere un progetto di tirocinio con cui approfondire le conoscenze già in mio possesso, andando ad apprendere i metodi di valutazione dei modelli linguistici, studiando i *benchmark* già esistenti fino a realizzarne uno. Inoltre, svolgere un progetto riguardante l'*AI* rientrava tra i miei primi parametri di scelta, in quanto sarebbe stato coerente alle mie aspettative accademiche e lavorative future.

Oltre a queste ragioni riguardanti la tematica dello *stage*, la mia decisione di collaborare con Sync Lab deriva dalla mia alta aspettativa di crescita personale. Ho infatti cercato un'azienda che desse la possibilità di lavorare anche in presenza, così da poter osservare dall'interno le dinamiche di un ambiente lavorativo.
Inoltre, a seguito di un secondo colloquio, ho definito con l'azienda l'utilizzo nel progetto di tecnologie, come il linguaggio *Java*, mai utilizzate prima, così da ottenere nuove competenze relative allo sviluppo *software*.

A partire da queste motivazioni, ho deciso di pormi due obiettivi personali per questo progetto di *stage*. Il primo di questi prevedeva il soddisfacimento di tutti gli obiettivi aziendali, unica metrica oggettiva per misurare la qualità del mio lavoro, indipendentemente dall'importanza attribuito ad ognuno di essi. Questo è stato il mio intento primario durante tutta la realizzazione del progetto, in quanto il dover acquisire conoscenze e competenze solide riguardanti le tecnologie necessarie allo *stage* è condizione necessaria per poter soddisfare tutti gli obiettivi previsti. Così facendo, raggiungendo questo obiettivo personale avrei avuto la certezza di aver

realizzato dei prodotti di qualità, acquisendo nel contempo nuove competenze.
Il mio secondo obiettivo personale riguarda il metodo di valutazione e comparazione dei modelli. Secondo il Piano di Lavoro pensato e realizzato insieme al mio tutor aziendale, il confronto tra i modelli avrebbe dovuto basarsi su alcuni semplici *test* svolti sui *Large Language Model* e in gran parte sulle informazioni reperibili *online*, come costo, risorse necessarie e popolarità. Il mio obiettivo personale prevedeva la realizzazione di un *benchmark* con cui svolgere la misurazione e il confronto delle *performance*, andando a realizzare il *dataset* necessario e sviluppando un ambiente di *benchmarking* con cui poter valutare i modelli linguistici in modo semi-automatico. Il raggiungimento di questo obiettivo non avrebbe solo reso più attendibili i risultati del mio progetto, ma anche dimostrando da parte mia l'aver ottenuto una conoscenza profonda dei moderni metodi di valutazione di *LLM*.

Capitolo 3

Svolgimento

3.1 Prototipazione rapida

3.1.1 Descrizione delle attività

Prima di poter analizzare e confrontare i *Large Language Model*, scelti in accordo con il tutor aziendale e già esposti nella sezione 2.3.1.4, è stato necessario progettare e realizzare dei prototipi con cui interfacciarsi a tali e integrare un sistema di *information retrieval*. A questo scopo, ho suddiviso la prima metà del progetto in due parti. Il primo periodo, denominato "prototipazione rapida", è stato indirizzato allo studio preliminare delle tecnologie necessarie allo *stage* e alla realizzazione di prototipi *backend-only*, con lo scopo di acquisire rapidamente conoscenze sul linguaggio *Java* e i vari *framework* utilizzati per la codifica e l'accesso ai modelli.
Solo successivamente, ho progettato e realizzato un prototipo finale, comprensivo di interfaccia grafica, con cui poter eseguire i primi reali test sui *LLM* e confrontare facilmente le risposte generate.

3.1.2 Tecnologie utilizzate

GitHub: piattaforma di *hosting* per progetti, basato sul *software* di controllo della versione *Git*. Permette il versionamento e la condivisione di codice, conservandolo in una repository$_G$ in remoto. Oltre a tenere traccia delle modifiche apportate al codice prodotto, ho utilizzato questa tecnologia per condividerlo con il mio tutor aziendale.

IntelliJ IDEA: ambiente di sviluppo integrato, sviluppato da *JetBrains*, per i linguaggi di programmazione *Java* e *Kotlin*. Ho scelto questo strumento per lo sviluppo del codice del *backend* per i numerosi vantaggi che offre. Oltre ad analizzare il codice scritto in tempo reale, informando tempestivamente la presenza di errori e fornendo suggerimenti per il *refactoring* automatico, sup-

porta e facilita l'utilizzo di altri *framework* utilizzati come *Spring* e *Maven*, semplificando notevolmente le attività di codifica.

Java: linguaggio di programmazione ad alto livello ed orientato agli oggetti, è stato utilizzato per la codifica del *backend* dei prototipi. Oltre ad essere il linguaggio maggiormente utilizzato per lo sviluppo delle applicazioni dall'azienda ospitante, è stato scelto anche per i suoi numerosi pregi, come robustezza, sicurezza e portabilità.

LangChain4J: libreria *open-source* di sviluppo, semplifica l'integrazione dei modelli linguistici in applicazioni *Java*. Progettata a partire dalla libreria *LangChain* per *Python* e *JavaScript*, offre tutti gli strumenti necessari per implementare la *Retrieval-Augmented Generation*, dall'*ingestion* delle informazioni fino all'*information retrieval* e generazione della risposta. Nonostante la documentazione di questa libreria sia attualmente poco sviluppata, l'ho preferita a *Spring AI*, principale alternativa per l'integrazione di modelli linguistici in *Java*, dato il suo elevato numero di tecnologie supportate. Infatti, *LangChain4J* offre l'integrazione di una più ampia platea di *Large Language Model*, *Embedding Model*, *database* vettoriali e *framework Java*.[14]
Ulteriore punto a favore della libreria scelta, attraverso delle funzioni denominate *chain* offre la possibilità di integrare la *RAG* in modo completamente automatico, indicando solo quale *LLM*, *Embedding Model* e *vectore store* utilizzare. Nell'implementazione dei prototipi non sono state tuttavia utilizzate queste funzioni, per poter aver maggior controllo del funzionamento delle varie fasi della *Retrieval-Augmented Generation*.

Ollama: piattaforma *open-source* che consente l'installazione e l'esecuzione in locale di *Large Language Model*. L'integrazione dei modelli è semplice e veloce, grazie all'utilizzo di apposite API$_G$ per interagire con il *framework*. Ho utilizzato questa tecnologia per installare ed eseguire *on-premise* i modelli in locale.

Postman: piattaforma di *testing* per progettare, testare e monitorare il funzionamento di *API*. È stata utilizzata per raggiungere gli *endpoint* dei servizi esposti dai prototipi, così da eseguire le funzionalità implementate attraverso l'invio di richieste HTTP$_G$ e visualizzando le risposte restituite dal *server*.

Spring: *framework open-sorce* per lo sviluppo modulare e semplificato di applicazioni in *Java*. Questo strumento è organizzato in più componenti, denominati moduli, che forniscono funzionalità aggiuntive per la gestione dei problemi più ricorrenti della programmazione *Java*. Il suo utilizzo nello sviluppo di appli-

cazioni *web* è pratica comune, in quanto favorisce l'adozione di *design pattern* con cui ottenere un'architettura robusta, modulare e facilmente testabile.

3.1.3 Prodotti software

3.1.3.1 Progettazione e sviluppo

Durante questa prima fase dello *stage*, sono stati prodotti tre diversi prototipi: uno integrante Phi-3 Mini in locale, un secondo per utilizzare i modelli *Google* Gemini 1.5 Pro e PaLM 2 Bison, e l'ultimo per interfacciarsi a GPT-3.5 Turbo. Tutti e tre i prodotti *software* condividono caratteristiche comuni come architettura, *endpoint* e tecnologie per le funzionalità di *Retrieval-Augmented Generation*.
Per prendere confidenza con il linguaggio *Java* e il *framework Spring*, mai utilizzati prima e tecnologie alla base dei prodotti *software*, ho progettato e sviluppato i prototipi applicando il *Controller-Service-Repositoy pattern*. Il *backend* è stato così organizzato in tre classi, in base alla logica e alla funzione del codice implementato in esse.
Nel *Controller*, sviluppato utilizzando i decoratori resi disponibili dal *framework Spring* così come il *Service* e il *Repository*, sono stati esposti i quattro *endpoint* per le chiamate *HTTP* alle funzionalità dei prodotti: inserimento e rimozione di documenti dal *database* vettoriale, lettura dei documenti già presenti e interazione con il modello linguistico implementato. Per raggiungere questi *endpoint* attraverso delle richieste *HTTP* e visualizzare le risposte, viene utilizzata la piattaforma *Postman*.
Ogni metodo del *Controller* chiama la funzione corrispettiva presente nel *Service*, con la quale svolgere le operazioni necessarie al completamento della funzionalità desiderata. Per svolgere tutte le funzionalità necessarie alla *Retrieval-Augmented Generation*, nei *Service* è utilizzata la libreria *LangChain4J*. Quest'ultima, per poter utilizzare un modello linguistico in locale, interagisce con *Ollama* per inviare una richiesta al *LLM* installato. La classe *Service*, a sua volta, utilizza la *Repository* per svolgere le operazioni sui dati contenuti nel *database* vettoriale, come lettura, scrittura ed eliminazione, nell'esecuzione dei propri metodi.
Nei prototipi ho implementato un *database* vettoriale *in-memory*, fornito dalla libreria *LangChain4J* e persistito su disco in formato *JSON*. Rispetto a una tradizionale base di dati con persistenza, questa tipologia di *vector store* permette prestazioni migliori, con maggiore velocità ed efficienza nelle operazioni, quando la mole di dati da memorizzare è limitata. Considerata la natura prototipale dei prodotti *software* che ho sviluppato, e quindi il numero limitato di documenti da inserire nel *database* durante le operazioni di valutazione dei modelli, il limite di questa tecnologia *in-memory* è stato ritenuto irrilevante.

Altro aspetto comune ai tre prototipi è l'*Embedding Model* utilizzato per la generazione dei vettori: all-MiniLM-L6-v2[9]. Questo modello, durante la fase di *data ingestion*, genera dei vettori di 384 dimensioni a partire dal testo che riceve. Ho scelto questo modello perché, oltre a possedere buone performance nel catturare le relazioni semantico logiche delle frasi e ricercare quelle più semanticamente simili, ha delle dimensioni abbastanza ridotte (inferirore a 100 MB) da essere facilmente scaricato ed eseguito localmente su ogni *device*, necessitando di pochissime risorse.

3.1.3.2 Funzionalità

Lettura del database

Grazie ad una chiamata all'*endpoint* associato, senza dover specificare alcun parametro nel corpo della richiesta, viene chiamato il metodo del *Controller* con cui recuperare il nome dei documenti contenuti nel *database* vettoriale.
Il contenuto del *vectore store* viene così recuperato dalla *Repository* e restituito al *Service*. Ogni *record* del *database*, oltre al *chunk* e al suo *embedding*, ha associato un metadato riportante il nome del documento da cui è stato estratto il testo. Recuperando tutti i metadati contenuti nella base di dati, viene restituito al *Controller* una lista di stringhe con i nomi dei documenti contenuti nel *database* vettoriale, e restituito come risposta alla richiesta iniziale.

Inserimento documento

Con una chiamata *POST* e un documento *PDF* inviato nel *body*, la richiesta inviata con *Postman* raggiunge il *Controller*, il quale invia al servizio il documento ricevuto. Qui avviene l'intera fase di *data ingestion* della *RAG*: il testo contenuto nel documento è estratto e suddiviso in *chunk* corrispondenti ai paragrafi.
Dato che la qualità dell'*embedding* è influenzata dalla grandezza del *chunk*, ogni frammento del testo ha lunghezza massima di settecento caratteri. In caso di paragrafi più ampi, sono divisi mantenendo un *overlap* di cento caratteri, ovvero i frammenti di ogni paragrafo contengono i cento caratteri finali del precedente, così da assicurare che nessuna informazione venga troncata prima di essere vettorizzata e memorizzata nella base di dati.
Terminata la fase di *chunkerizzazione*, ogni porzione di testo estratto viene processato da all-MiniLM-L6-v2, e gli *embedding* così generati sono inseriti nel *vector database* insieme al testo originale e al nome del documento da cui proviene. Modificato il *database*, viene aggiornato il file *JSON* utilizzato per persistere la base di dati *in-memory*. La risposta alla chiamata *HTTP* di partenza è una stringa che informa del corretto inserimento o meno del documento.

Rimozione documento

Tramite una chiamata *DELETE*, contenente nel corpo della richiesta il nome del documento da rimuovere, è possibile rimuovere i dati relativi ad un *PDF* inserito in precedenza. Così come nelle altre funzionalità di lettura e scrittura, il *Service* opera sul *database* vettoriale attraverso la *Repository*: utilizzando i metadati relativi al documento fonte di una informazione, tutti i *record* contenenti i *chunk* e i vettori multidimensonali del documento indicato nella richiesta sono rimossi. Aggiornato il file *JSON* di persistenza, il *Controller* restituisce una stringa che informa della corretta rimozione o meno del documento.

Interazione con il modello

Per poter interagire con il modello linguistico, ho implementato un *endpoint* specifico da raggiungere con la domanda da porre contenuta nel corpo della richiesta. Raggiunto il metodo relativo nel servizio, la domanda è utilizzata per le fasi di *information retrieval* prima e *generation* poi. Il testo inviato viene infatti *embeddizato*, con lo stesso modello utilizzato nella *data ingestion*, e utilizzato per ricercare le informazioni rilevanti presenti nel *database* vettoriale.
L'*embedding* della domanda viene confrontata con ogni altro vettore presente nel *database*, calcolando la loro similarità semantica mediante la *cosine similarity*: vengono così individuati tutti i *chunk* con similarità minima del 50% e recuperati i primi cinque. La domanda originale e le informazioni rilevanti recuperate, insieme ad altre istruzioni specifiche per il modello, sono utilizzate per comporre il *prompt* finale inviato al *Large Language Model* implementato nel prototipo. La risposta da lui generata viene ritornata al *Controller*, che a sua volta la invia nel *body* della risposta alla richiesta iniziale.

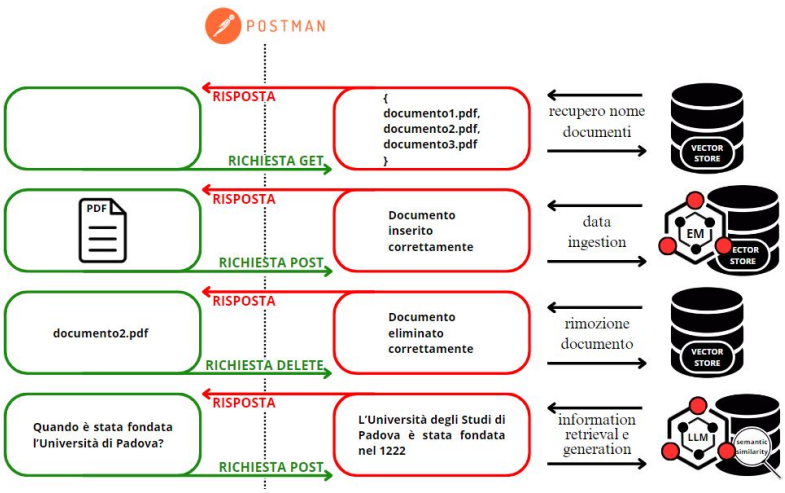


Figura 3.1: Funzionamento dei prototipi backend

3.2 Analisi dei requisiti

Sviluppati i prototipi *backend-only* con cui svolgere la *Retrieval Augmented-Generation*, ho potuto iniziare le attività indirizzate alla realizzazione del prototipo finale. In particolare, ho svolto un'attenta analisi dei requisiti per individuare gli utilizzatori di un possibile prodotto finale e le funzionalità che esso deve implementare.
Questa attività è stata particolarmente importante in quanto ha definito le aspettative, dal punto di vista delle funzionalità implementate, del prototipo finale del progetto. Definendo in modo chiaro e documentato i requisiti, andando anche a specificare la fonte di ognuno di essi, ho potuto guidare le attività di progettazione e codifica, successive all'analisi dei requisiti, sulla base di essi.

3.2.1 Attori

Gli attori rappresentano la categoria o il ruolo che un utente ricopre durante l'interazione con il sistema, e si differenziano tra loro per gli obiettivi che vogliono raggiungere. Poter comprendere chi sono gli attori che possono interagire con il sistema permette una più facile ed esaustiva definizione dei requisiti di un prodotto. Da un'analisi condotta insieme al mio tutor aziendale, durante una riunione nella quale abbiamo deciso come strutturare il prototipo, gli attori individuati sono due:

Utente generico: rappresenta un utente che non ha completato alcun processo di autenticazione, con accesso a una porzione limitata delle funzionalità dell'applicativo.

Admin: rappresenta un utente che ha effettuato l'autenticazione come amministratore, ricevendo l'accesso a tutte le funzionalità implementate nell'applicazione.

Potendo svolgere tutte le funzionalità a cui l'utente generico ha già accesso, oltre ad altre aggiuntive, l'amministratore risulta essere una sua generalizzazione.

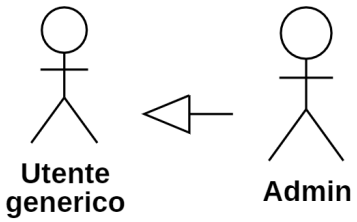


Figura 3.2: Attori individuati

3.2.2 Casi d'uso

I casi d'uso, o *use case*, sono un insieme di azioni, dette scenari, compiute da un attore e finalizzate al raggiungimento di un preciso scopo finale. Individuando i casi d'uso è possibile identificare le funzionalità che il prodotto deve implementare e come esse sono percepite dall'utente.
Per identificare i *use case* del prototipo ho realizzato dei diagrammi UML$_G$ specifici, detti *Use Case Diagram*, riportando per ognuno di essi la descrizione del caso d'uso, gli attori coinvolti, le pre e le post condizioni, lo scenario ed eventuali estensioni.
Ogni caso d'uso identificato è stato associato ad un codice identificativo univoco, composto dalla sigla UC e seguito da un numero progressivo: tutti quelli il cui numero è nel formato `X.Y` identificano un sotto caso d'uso `Y` figlio di UC`X`, ovvero uno *use case* che ha in comune col padre lo scenario principale.

Essendo il prototipo finalizzato alla realizzazione di un prodotto *software* con solo le principali funzionalità legate alla *Retrieval Augmented-Generation*, i diagrammi risultano semplici e in numero ridotto. Di seguito, con le figure 3.3 e 3.4, sono riportati i diagrammi da me prodotti, raffiguranti le interazioni degli attori con il sistema e accompagnati dalla descrizione dei principali casi d'uso individuati.

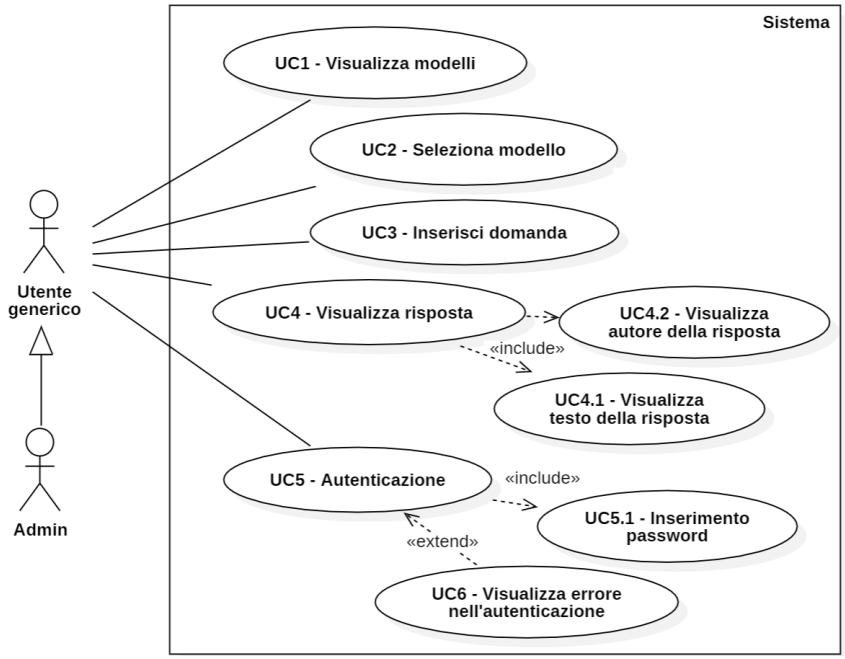


Figura 3.3: Diagramma dei casi d'uso comuni per l'utente generico e l'admin

UC2: Seleziona modello

Descrizione: L'utente vuole selezionare il modello dal quale ricevere le risposte alle proprie domande.
Attori: Utente generico, Admin.
Precondizioni: L'utente ha visualizzato la lista dei modelli supportati.
Postcondizioni: Il sistema utilizzerà il modello selezionato per generare le risposte.
Scenario:

1. L'utente visualizza la lista dei modelli supportati (UC1);
2. L'utente seleziona quale modello usare nella generazione delle risposte.

UC3: Inserisci domanda

Descrizione: L'utente vuole inviare una domanda al *chatbot*, il quale dovrà rispondere utilizzando il modello desiderato.
Attori: Utente generico, Admin.
Precondizioni: L'utente ha selezionato il modello da interrogare.
Postcondizioni: Il sistema riceve la domanda posta dall'utente.
Scenario:

1. L'utente visualizza la lista dei modelli supportati (UC1);
2. L'utente seleziona quale modello usare nella generazione delle risposte (UC2);
3. L'utente digita la domanda e la invia al sistema.

UC4: Visualizza risposta

Descrizione: L'utente vuole visualizzare la risposta del *chatbot* alla domanda che ha posto in precedenza.
Attori: Utente generico, Admin.
Precondizioni: L'utente ha inviato una domanda al *chatbot*.
Postcondizioni: Il sistema mostra la risposta elaborata dal sistema.
Scenario:

1. L'utente digita la domanda e la invia al sistema (UC3);
2. L'utente visualizza il messaggio di risposta ricevuto e il nome del modello che lo ha generato.

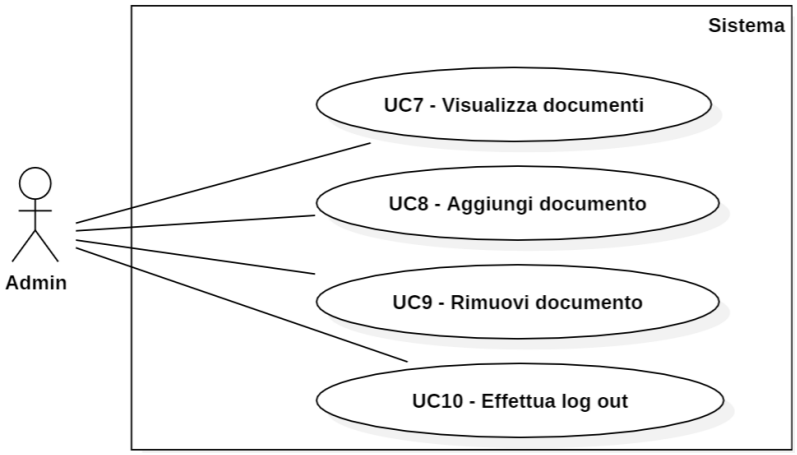


Figura 3.4: Diagramma dei casi d'uso dell'admin

UC8: Aggiungi documento

Descrizione: L'amministratore vuole aggiungere un nuovo documento nel *database*, su cui poter fare delle domande al *chatbot*.
Attori: Admin.
Precondizioni: L'utente ha effettuato l'autenticazione come amministratore.
Postcondizioni: Il sistema ha processato correttamente il nuovo documento e mostra la lista dei documenti aggiornata.
Scenario:

1. L'utente ha effettuato l'accesso come amministratore (UC5);
2. Il sistema mostra la lista di tutti i documenti presenti nel *database* (UC7);
3. L'amministratore inserisce il nuovo documento.

UC9: Rimuovi documento

Descrizione: L'amministratore vuole rimuovere un documento nel *database*, così che il *chatbot* non fornisca più risposte su di esso.
Attori: Admin.
Precondizioni: L'utente ha effettuato l'autenticazione come amministratore.
Postcondizioni: Il sistema ha rimosso dal *database* il documento e mostra la lista dei documenti aggiornata.
Scenario:

1. L'utente ha effettuato l'accesso come amministratore (UC5);
2. Il sistema mostra la lista di tutti i documenti presenti nel *database* (UC7);
3. L'amministratore elimina uno dei documenti presenti dalla lista.

3.2.3 Requisiti

Da un'attenta analisi dei casi d'uso individuati, oltre che degli obiettivi riportati nel Piano di Lavoro dello *stage*, sono stati definiti i requisiti che il prototipo deve soddisfare. Questi sono stati suddivisi in categorie: in riferimento all'importanza del suo soddisfacimento, un requisito può essere obbligatorio (O) desiderabile (D) o opzionale (Z). In riferimento alla tipologia, un requisito può essere funzionale (F), qualitativo (Q) o di vincolo (V).
Per ogni requisito individuato è stato anche riportata la sua fonte: per quelli funzionali è stato specificato il caso d'uso relativo, mentre per i qualitativi e di vincolo la fonte di riferimento è il Piano di Lavoro, definito in accordo con i miei due tutor, e i colloqui svolti con il mio solo tutor interno, denominando tale fonte come "interna".

Requisiti funzionali

Requisito	Descrizione	Caso d'uso
RFZ-1	L'utente deve poter visualizzare i modelli utilizzabili nell'interazione col *chatbot*.	UC1
RFZ-2	L'utente deve poter selezionare quale modello utilizzare nell'interazione col *chatbot*.	UC2
RFO-3	L'utente deve poter inviare delle domande al *chatbot*.	UC3
RFO-4	L'utente deve poter visualizzare la risposta data dal *chatbot* e il modello autore.	UC4
RFO-5	L'utente deve poter effettuare l'autenticazione per accedere alle funzionalità dell'amministratore.	UC5
RFO-6	L'utente deve visualizzare un messaggio di errore quando l'autenticazione non va a buon fine.	UC6
RFO-7	L'amministratore deve poter visualizzare quali sono i documenti presenti nel *database*.	UC7
RFO-8	L'amministratore deve poter inserire un nuovo documento su cui poter interrogare il *chatbot*.	UC8
RFO-9	L'amministratore deve poter inserire rimuovere un documento dal *database*.	UC9
RFO-10	L'amministratore deve poter effettuare il *logout*.	UC10

Tabella 3.1: Requisiti funzionali del prototipo finale

Requisiti di vincolo

Requisito	Descrizione	Fonte
RVO-1	Il *backend* del prototipo deve essere sviluppato in *Java*, utilizzando *Spring*.	Piano di Lavoro
RVO-2	Il *frontend* del prototipo deve essere sviluppato in *TypeScript*, utilizzzando *React*.	Piano di Lavoro
RVO-3	Il prototipo deve permettere l'utilizzo del modello Phi-3 Mini.	Piano di Lavoro
RVO-4	Il prototipo deve permettere l'utilizzo del modello Gemini 1.5 Pro.	Piano di Lavoro
RVD-5	Il prototipo deve permettere l'utilizzo del modello GPT-3.5 Turbo.	Piano di Lavoro
RVZ-6	Il prototipo deve permettere l'utilizzo del modello PaLM 2 Bison.	Interna
RVZ-7	Il prototipo deve permettere la selezione in tempo reale del modello da interrogare.	Piano di Lavoro

Tabella 3.2: Requisiti di vincolo del prototipo finale

Requisiti qualitativi

Requisito	Descrizione	Fonte
RQO-1	Deve essere realizzato un documento che riporti le fasi di analisi dei requisiti, progettazione e codifica del prototipo.	Piano di Lavoro
RQD-2	Il codice prodotto deve essere condiviso con l'azienda in una *repository GitHub*.	Interna

Tabella 3.3: Requisiti qualitativi del prototipo finale

3.3 Progettazione e codifica

3.3.1 Tecnologie utilizzate

Oltre alle tecnologie già utilizzate nella realizzazione dei prodotti durante la prototipazione rapida, presentate nella sezione 3.1.2, ho avuto necessità di individuare e usare altre tecnologie: questo perché il prototipo finale presenta anche un'interfaccia grafica. Di seguito, sono riportate le altre tecnologie usate nella codifica del prodotto *software* finale.

H2: gestore di basi di dati scritto in *Java*, è stato utilizzato per contenere le credenziali con cui eseguire l'autenticazione come amministratore nel prototipo. Ho scelto questo *database* perché supportato per l'utilizzo di *Spring Data JPA*, un modulo dell'omonimo *framework* per l'implementazione automatica dei *Data Access Object*, e *Spring Security*, utilizzato per criptare le *password* con funzione di *hashing* Bcrypt$_G$.

NodeJs: ambiente *runtime* di codice *JavaScript*, permette la creazione e l'esecuzione lato *server* di applicazioni *web* scalabili e performanti.

React: libreria *JavaScript* per la realizzazione di interfacce grafiche, è stata utilizzata per la realizzazione del *frontend* del prodotto.

TypeScript: linguaggio di programmazione *open-source superset* del linguaggio *JavaScript*, ovvero che ne estende la sintassi. Rispetto a quest'ultimo, *TypeScript* presenta il supporto dei tipi statici, che permette di identificare errori di tipo in fase di compilazione.

Visual Studio Code: ambiente di sviluppo integrato utilizzato durante la realizzazione del codice *frontend*, grazie al terminale integrato permette la rapida esecuzione del *server* di sviluppo.

3.3.2 Architettura

A seguito di una prima fase di progettazione, il prototipo finale da me prodotto implementa una *Layered Architecture*. *Pattern* architetturale comunemente utilizzato per la maggior parte delle applicazioni sviluppate in *Java*, prevede la suddivisione di un sistema in strati paralleli in base alla logica e alla funzione del codice in esso contenuto. Ogni *layer* dipende solo dallo strato immediatamente successivo e risponde ad una sola responsabilità.
In particolare, l'architettura finale presenta quattro strati:

Presentation Layer: contiene la logica presentazionale del prototipo, gestendo la visualizzazione dei dati e l'interazione lato utente. Questo strato si concretizza nell'interfaccia grafica in *React* del *frontend TypeScript* e nel codice del *Controller* nel *backend Java* del *software*: quest'ultimo inoltra le richieste provenienti dal *client* al *layer* sottostante, esponendo i servizi raggiunti con delle richieste *HTTP* durante l'esecuzione del prototipo con *NodeJs*.

Business Layer: contiene la logica di *business* del *software*, svolgendo le funzionalità implementate dall'applicativo. Si concretizza nelle due classi *Service* del prototipo, il *RagService* e il *AuthService*, con le quali svolgere rispettivamente

le funzionalità di *Retrieval-Augmented Generation* e di autenticazione. In particolare, il *RagService* permette di svolgere tutte le funzionalità, già presentate in precedenza nella sezione 3.1.3.2, dei prototipi *backend* attraverso l'utilizzo di *LangChain4J* e *Ollama*: gli utenti, dopo aver selezionato il modello linguistico da utilizzare, possono interagire con il *chatbot*, mentre l'amministratore può anche visualizzare, aggiungere e rimuovere documenti dal sistema. L'*AuthService* invece gestisce le operazioni di autenticazione dell'applicativo, permettendo il confronto delle credenziali inserite dall'utente in fase di autenticazione con quelle criptate presenti nel *database*, così da consentire o meno l'accesso dell'utente alle funzionalità di sola competenza dell'*admin*.

Persistence Layer: contiene la logica di persistenza dell'applicativo, responsabile di svolgere le operazioni sui dati contenuti nei *database*. Nel prodotto sono state implementate due classi *Repository*, l'*EmbeddingRepository* e la *CredentialRepository*, con la quale interfacciarsi rispettivamente al *database* vettoriale e alla base di dati contenente le credenziali dell'amministratore.

Database Layer: questo strato è costituito dalle basi di dati implementate e raggiungibili dalle classi *Repository*. Nel prototipo finale è stato utilizzato il *database H2*, criptato utilizzando *Spring*, per memorizzare la *login* dell'*admin*, mentre per contenere gli *embedding* e i *chunk* dei documenti è stato utilizzato, come per i prototipi *backend-only*, il *vector database* della libreria *LangChain4J*.

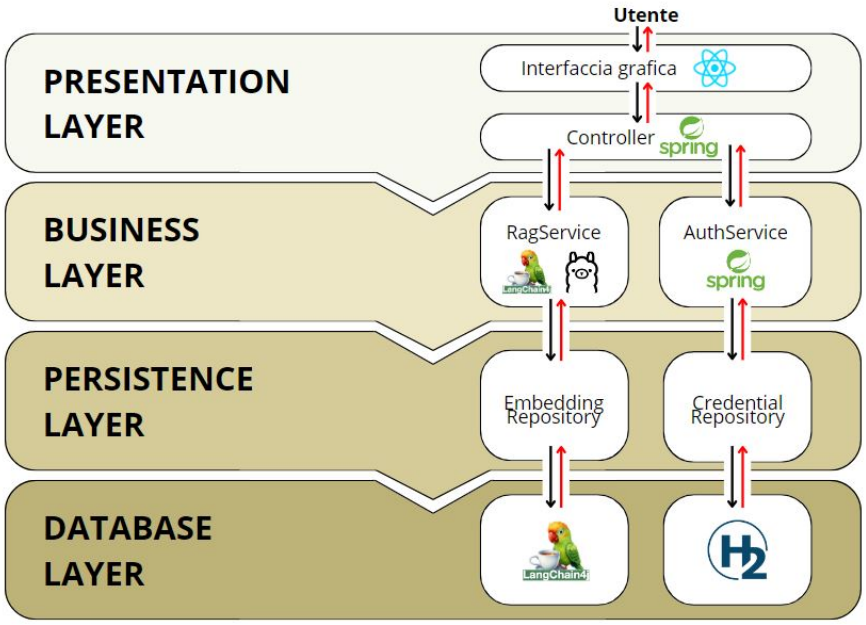


Figura 3.5: Layered Architecture implementata

3.3.3 Design pattern

Controller-Service-Repository

Coerentemente alla decisione di implementare un'architettura stratificata, ho suddiviso le classi del *backend* in *Controller*, *Service* e *Repository* così da ottenere una suddivisione del codice per logica e responsabilità. Il *Controller* gestisce le richieste provenienti dal *frontend* del prototipo, chiamando le funzioni del *Service* per l'esecuzione delle funzionalità. Il *Repository* rappresenta un'astrazione tra il codice di *business* e le basi di dati, e tutte le operazioni *CRUD* sono svolte da questa classe. Utilizzando questo *pattern* ho potuto ottenere la separazione delle responsabilità delle classi implementate, ottenendo un prodotto modulare e manutenibile.

Dependency Injection

Grazie all'utilizzo di *Spring*, sono stato in grado di applicare l'*Inversion of Controll* in modo automatico. Questo *pattern* permette di minimizzare le dipendenze del codice, separando il comportamento di una componente dalla risoluzione delle sue dipendenze, così da minimizzare il grado di accoppiamento.
In particolare, ho utilizzato l'annotazione `@Autowired` fornita dal *framework* per ottenere una iniezione della dipendenza nel costruttore della classe *target*. Un esempio del suo utilizzo per iniettare le dipendenze del *RagService* è riportato di seguito.

```
1  @Service
2  public class RagService {
3
4      private final EmbeddingRepository embeddingRepository;
5
6      @Autowired
7      public RagService(EmbeddingRepository embeddingRepository) {
8          this.embeddingRepository = embeddingRepository;
9      }
```

Codice 3.1: Esempio di implementazione della Dependency Injection

Builder

Il *Builder* è un *design pattern* finalizzato alla costruzione di oggetti complessi passo per passo. Il suo utilizzo è particolarmente utile nel caso in cui bisogna creare oggetti complessi con numerosi attributi o opzioni di configurazione, consentendo di specificare i soli attributi d'interesse, mantenendo i rimanenti al valore di *default*.
In particolare, ho implementato questo *pattern* per costruire gli oggetti rappresentanti i modelli linguistici, facilitando la loro instanziazione andando a configurare

solo determinati valori, come il nome del modello o la sua temperatura. Un esempio di come ho applicato il *Builder pattern* per costruire gli oggetti delle classi dei modelli è riportato di seguito.

```
1   ChatLanguageModel chatModel = VertexAiChatModel.builder()
2           .project("gemini-synclab-proj")
3           .location("us-central1")
4           .modelName("chat-bison-32k")
5           .publisher("google")
6           .endpoint("us-central1-aiplatform.googleapis.com:443")
7           .temperature(0.6)
8           .build();
9       }
```

Codice 3.2: Esempio di implementazione del pattern Builder

Compound Components

Avendo codificato il *frontend* del prototipo con la libreria *React*, ho ritenuto opportuno adottare alcune delle *best practice* e dei *pattern* più comuni di questa tecnologia. Uno di questi è il *Compund Components*, un *design pattern* tipico di *React* che prevede la realizzazione di molti componenti, ovvero singoli elementi che compongono l'interfaccia grafica, così da elevare la modularità del codice prodotto e permettere il loro riutilizzo senza ridefinire più volte lo stesso elemento. Seguendo questo *pattern*, oltre a rispettare il principio *Don't Repeat Yourself* dello sviluppo *software*, ho evitato di codificare intere schermate in un unico grande componente, così da facilitare la comprensione e la manutenibilità del codice.

Higher-Order Components

Un altro *pattern React* utilizzato nel prototipo è il *Higher-Order Components*, sfruttato per implementare la funzionalità di autenticazione nel prodotto. Questa soluzione prevede l'espansione della logica di un componente attraverso la definizione di un secondo che lo ingloba.
In particolare, ho adottato questa tecnica per definire una funzione di controllo dell'autorizzazione per l'accesso alla parte del prototipo dedicata dell'amministratore: se si tenta di visualizzare la pagina dell'*admin*, il componente con la funzione di sicurezza che la avvolge controlla se è avvenuta o meno l'autenticazione e, in caso contrario, blocca l'accesso reindirizzando l'utente nella pagina di autenticazione. Di seguito è riportato il codice del componente *Higer-Order* implementato.

```
const requireAuth = (WrappedComponent: any) => {
    return (props: any) => {
        const [isAuth, setIsAuth] = useState(false);
        useEffect(() => {
            const isAdmin = localStorage.getItem('isAdmin');
            if (!isAdmin) {
            window.location.href = '/auth';
            } else {
                setIsAuth(true);
            }
        }, []);

        if (!isAuth) return <main></main>;
        return <WrappedComponent {...props} />;
    };
};

export default requireAuth;
```

Codice 3.3: Implementazione del pattern Higher-Order Components

3.3.4 Interfaccia grafica

Avendo identificato a seguito dell'analisi dei requisiti due categorie di utente, ovvero l'amministratore e lo *user* generico, il prototipo presenta due pagine specializzate per l'utente che le deve utilizzare, oltre alle pagine di benvenuto e di autenticazione.

Pagina di benvenuto

Nella *home page* del prototipo sono presenti i due bottoni con cui accedere alle due pagine principali dell'applicazione, quella per gli utenti generici e l'amministratore, affiancati dal logo del prodotto.

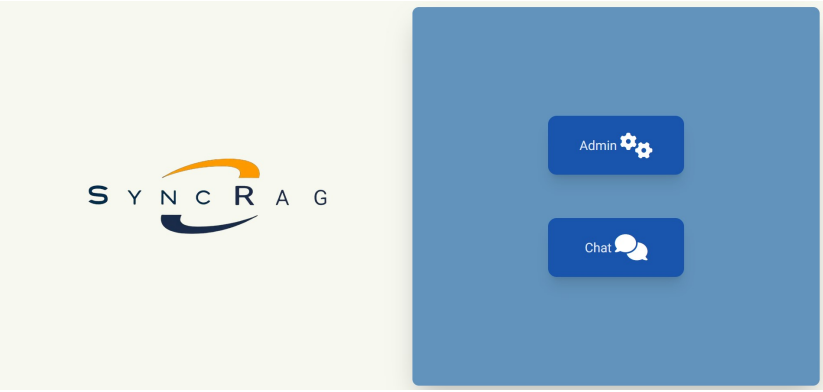


Figura 3.6: Interfaccia grafica della pagina di benvenuto

Pagina di chat

Nella pagina di *chat*, ad accesso libero senza necessità di autenticazione, l'utente può interagire con il *chatbot*, selezionando il modello linguistico da utilizzare e inviando domande. Le risposte ricevute, così come stabilito dai requisiti individuati prima della progettazione del prototipo, oltre al testo del messaggio riportano il nome del modello linguistico autore della risposta.
I messaggi, realizzati a partire da uno stesso componente *React* riutilizzato, permettono di distinguere facilmente se è stato inviato dall'utente o dal sistema, attraverso la differenziazione del colore di sfondo, oltre che dalla posizione del messaggio stesso.

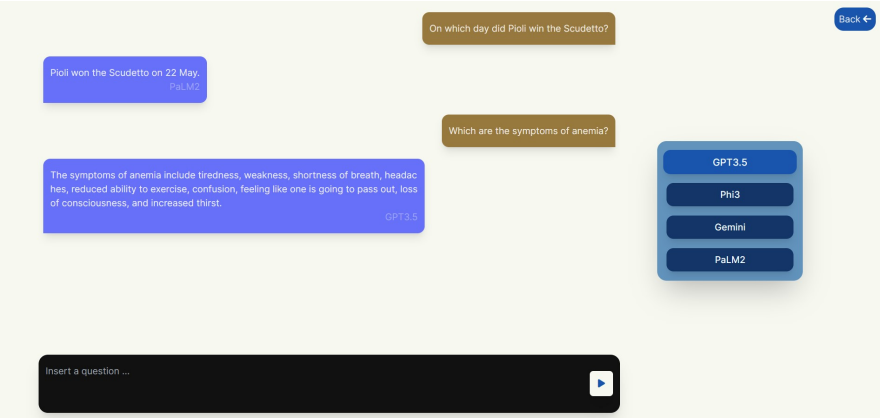


Figura 3.7: Interfaccia grafica della pagina di chat

Pagina di autenticazione

La pagina utilizzata dall'amministratore per eseguire l'autenticazione ed accedere all'area personale presenta un semplice *form*, nel quale poter inserire la *password* con cui accedere, dando la possibilità di scegliere se visualizzare o meno l'*input* che si sta digitando. Non è richiesto l'inserimento di alcun *username*, in quanto il prodotto prevede l'autenticazione di un solo utente, ovvero l'amministratore.
Una volta cliccato l'apposito bottone per effettuare il *login*, possono accadere due scenari distinti. Qualora l'utente abbia inserito la *password* sbagliata, il sistema notificherà tale errore con un messaggio a comparsa. Se la *password* è stata invece inserita correttamente, l'applicazione mostrerà la pagina dell'amministratore, nella quale saranno disponibili tutte le funzionalità del prototipo finale.

Figura 3.8: Interfaccia grafica della pagina di autenticazione

Pagina dell'amministratore

L'area del prototipo riservata all'*admin* presenta molte similitudini con la pagina adibita all'utilizzo dell'utente generico. Sono infatti presenti gli stessi componenti *React* utilizzati per i messaggi e il menù di selezione del modello, oltre al *form* con cui l'amministratore può caricare un nuovo documento: l'applicazione permette di effettuare la *data ingestion* di documenti *PDF*, per cui il sistema permette la selezione e l'*upload* di solo quell'estensione specifica. Qui è anche possibile visualizzare i documenti già presenti nel sistema, utilizzati per la generazione delle risposte del *chatbot*, e richiederne la rimozione definitiva dal *vector database*.
Quando l'amministratore richiede l'aggiunta o la rimozione di un documento, un messaggio a comparsa notifica la buona riuscita o meno dell'operazione.

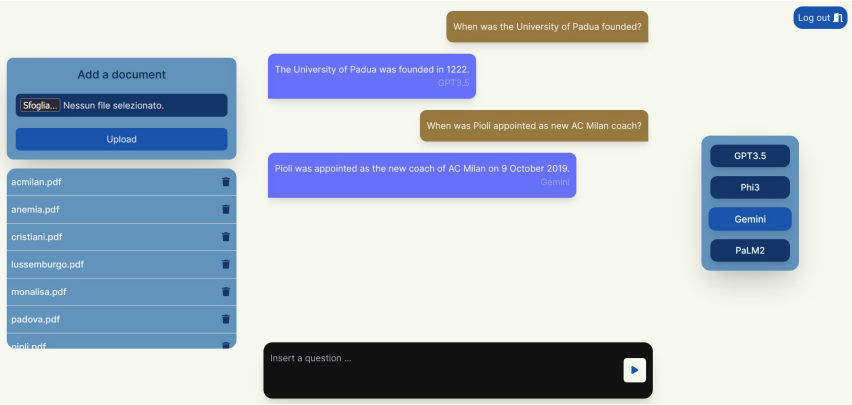


Figura 3.9: Interfaccia grafica della pagina dell'amministratore

Quando l'*admin* effettua il *logout*, utilizzando l'apposito bottone presente in alto a destra dell'interfaccia grafica, l'applicazione torna a mostrare la pagina di benvenuto e sarà richiesta nuovamente l'autenticazione per accedere ancora alla pagina dell'amministratore.

3.4 Verifica e validazione

Data la natura prototipale del prodotto *software* realizzato, pensato non per essere un prodotto finito ma per effettuare dei *test* delle *performance* dei modelli linguistici che implementa, in accordo con gli obiettivi aziendali non è stato realizzato alcun *test* automatico di unità per l'applicativo. L'attività di verifica si è infatti limitata all'analisi statica del codice, svolta in parallelo agli incontri di avanzamento svolti con il tutor aziendale, per garantire che il codice che producevo fosse robusto e conforme alle *best practice* della *clean programming* e dei principi *SOLID*.

La validazione del prototipo finale è avvenuta a sviluppo terminato, sempre durante una riunione con il mio tutor. In questo incontro, sono stati effettuati dei *test* di accettazione, durante i quali sono state provate tutte le funzionalità del prodotto *software*, così da verificare che tutti i requisiti obbligatori, individuati durante la fase di analisi, fossero stati soddisfatti.
Sono state quindi testate le funzionalità di autenticazione, *chat* e selezione del *Large Language Model*, fino all'aggiunta e rimozione dei documenti: oltre a valutare il corretto funzionamento di queste azioni, è stata verificata la robustezza del sistema nel gestire eventuali errori, come il tentativo di inserimento di un documento con un formato diverso da quello supportato.
Al termine della validazione, è stato correttamente verificato il soddisfacimento di ogni requisito del prototipo, osservando una completa gestione degli errori, portando il prodotto ad essere ritenuto sufficientemente robusto per essere accettato.

Durante la validazione sono anche state effettuate delle valutazioni sulle prestazioni del prototipo, attraverso dei *test* di carico. In particolare, per poter assicurare la buona riuscita delle successive operazioni di *evaluation* dei modelli linguistici, è stato misurato il tempo necessario al sistema di *Retrieval-Augmented Generation* per effettuare l'*ingestion* di un documento, oltre a valutare le prestazioni dell'applicativo quando presenti a sistema un elevato quantitativo di documenti *embeddizati*.
Entrambi questi *test* hanno portato ad esiti soddisfacenti, in quanto il tempo richiesto al sistema per processare un documento si è rilevato inferiore al secondo, e le prestazioni del prototipo non sono diminuite con più di venti documenti inseriti.
Per questi motivi, il prodotto *software* ha correttamente superato la fase di validazione, permettendomi così di proseguire il progetto con la valutazione comparativa dei *LLM*.

3.5 Confronto dei modelli

3.5.1 Esperimenti

Terminata la realizzazione del prototipo finale, ho potuto iniziare la fase di valutazione dei modelli. In primo luogo, sfruttando i prodotti *software* delle attività precedenti, ho potuto testare la qualità delle risposte dei *Large Language Model* al variare della struttura del *prompt* generato e di altri parametri, come temperatura dei modelli linguistici e valore soglia di similarità semantica.

Per confrontare le prestazioni dei modelli, nello specifico caso in cui viene applicata la *Retrieval-Augmented Generation*, ho deciso di realizzare un *benchmark*, ovvero un insieme di *task* da far svolgere comunemente ai modelli linguistici in esame, raccogliendone e confrontandone i risultati. Considerata la necessità di valutare le capacità dei modelli in un sistema integrante la *RAG*, i modelli hanno dovuto rispondere a delle domande utilizzando le sole informazioni rilevanti recuperate da un *database* vettoriale.
In particolare, il *dataset* del *benchmark* è composto da una raccolta di domande, riguardanti il contenuto di dieci differenti documenti, e dal *vector store* dove sono raccolti gli *embedding* dei documenti stessi. Volendo testare la capacità dei *LLM* di fornire risposte senza informazioni esterne al *database* vettoriale, i documenti riportano informazioni provenienti da pagine *Wikipedia*, quindi dati di cui i modelli probabilmente hanno già conoscenza, potendo potenzialmente fornire informazioni aggiuntive al dominio conoscitivo fornito.

Da ogni documento sono state ricavate dieci domande, per un totale di cento, variando sulla tipologia di risposta attesa: alcune domande semplicemente richiedevano la ricopiatura di un'informazione presente nel testo, altre necessitavano di una corretta comprensione ed elaborazione del contenuto del documento. Per ognuna delle domande definite, il *dataset* da me prodotto comprendeva anche la risposta attesa, da confrontare con quella ottenuta per verificarne la correttezza e calcolarne la qualità, misurando la *semantic similarity* tra le due, come mostrato in figura 3.10.
Per verificare che il modello fosse in grado di limitare le risposte al solo contenuto dei documenti, alcune domande richiedevano informazioni non contenute nel *vector database*: in questi casi, la risposta attesa è una frase *standard*, esposta nel *prompt* dato, che riporta l'impossibilità del modello di fornire una risposta.

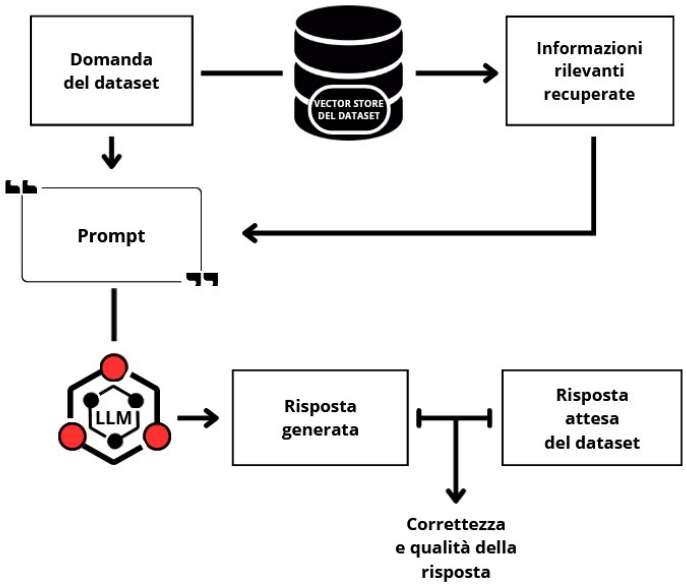


Figura 3.10: Funzionamento del benchmark realizzato

Una volta che il *LLM* ha risposto a tutte le domande, esso riceve un punteggio, espresso in centesimi, in base alla correttezza delle risposte fornite. Ogni risposta completamente corretta riceve un punto, mentre quando il modello genera una risposta incompleta, o non ne fornisce alcuna riportando di non avere abbastanza informazioni, non ne riceve alcuno.
Nel caso in cui il modello non segue correttamente l'istruzione definita dal *prompt*, ovvero quando fornisce informazioni non presenti nel *dataset* o fornisce una risposta non coerente a quella che era la domanda, esso riceve una penalità di mezzo punto. Lo stesso punteggio negativo è assegnato al verificarsi di allucinazioni, ovvero nel caso in cui il modello, mal comprendendo il testo recuperato dal *database* e fornito nel *prompt*, fornisce una risposta con informazioni sbagliate.
A completare la valutazione del modello linguistico, oltre alla correttezza e qualità delle risposte è calcolato anche il tempo di generazione.

3.5.2 Risultati

Per eseguire la valutazione dei *LLM*, ho implementato un ambiente di *benchmarking* semi-automatico in *Java*. Ogni risposta generata da un modello è classificata come corretta o meno in modo automatico, basando la scelta sulla similarità semantica tra la risposta attesa e quella effettiva, e corredata dal tempo di generazione e dal valore di similarità stesso. Successivamente, i dati così raccolti sono rifiniti tramite *human evaluation*, verificando manualmente la presenza di penalità da assegnare ad ogni risposta, potendo così calcolare i risultati finali.

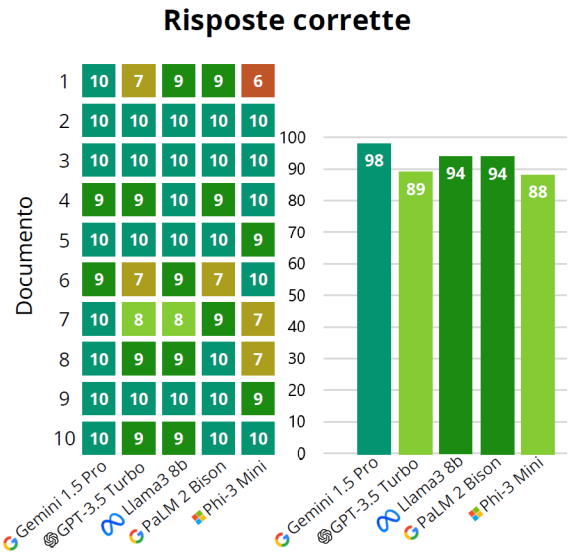


Figura 3.11: Risposte corrette per modello

Nella figura 3.11 ho riportato il grafico delle risposte corrette generate dai cinque modelli in esame. Di seguito, con la figura 3.12, è visibile il numero di risposte per le quali i modelli hanno ottenuto una penalità di mezzo punto, o perché non rispettato il *prompt* o per la presenza di un'allucinazione.

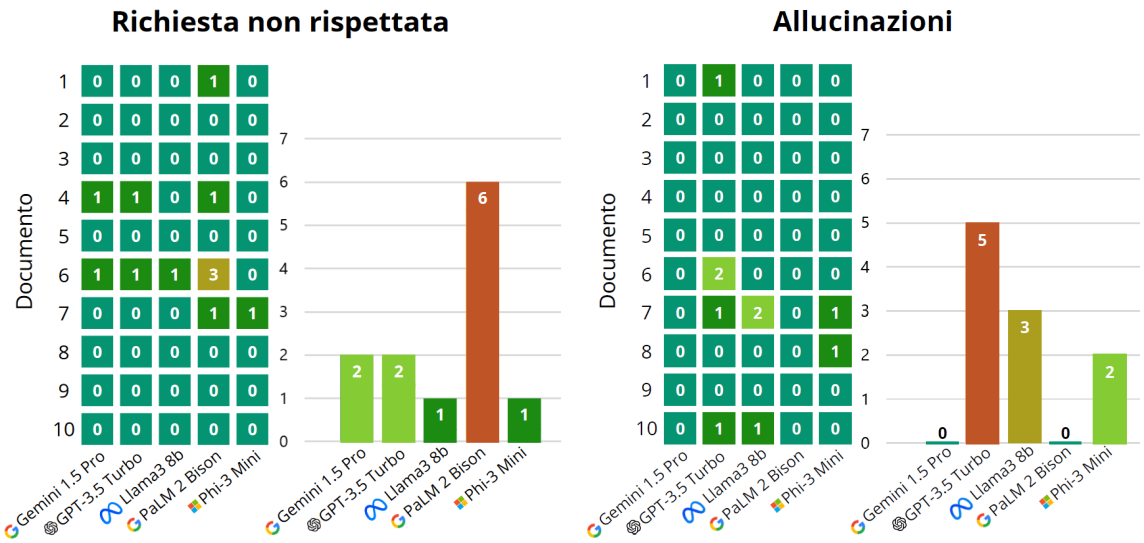


Figura 3.12: Penalità registrate per modello

Al netto del numero di risposte corrette e delle penalità assegnate, il punteggio registrato dai cinque *Large Language Model* nel *benchmark* realizzato è di seguito riportato.

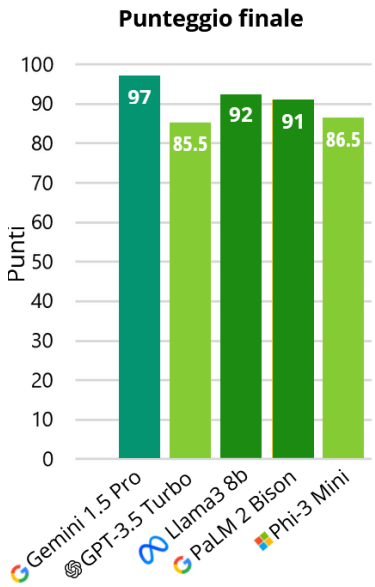


Figura 3.13: Punteggio finale per modello

A completare la valutazione dei modelli nel *dataset* proposto, nelle figure 3.14 e 3.15 sono riportati rispettivamente i valori medi di similarità semantica, calcolata attraverso *cosine similarity*, delle sole risposte corrette rispetto quelle attese e i secondi medi impiegati dal *LLM* per generare una risposta, da quando ha ricevuto in *input* il *prompt*.

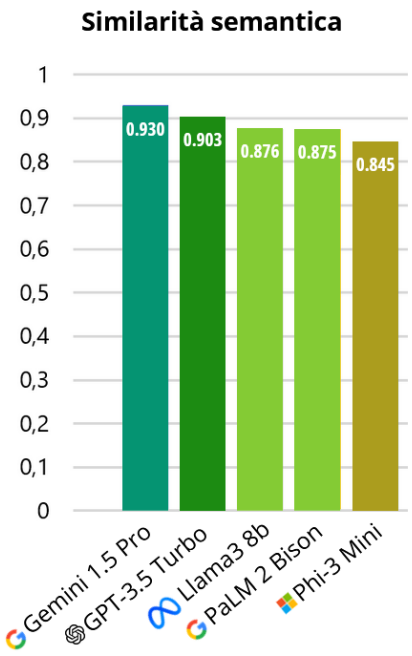


Figura 3.14: Similarità semantica media delle risposte corrette

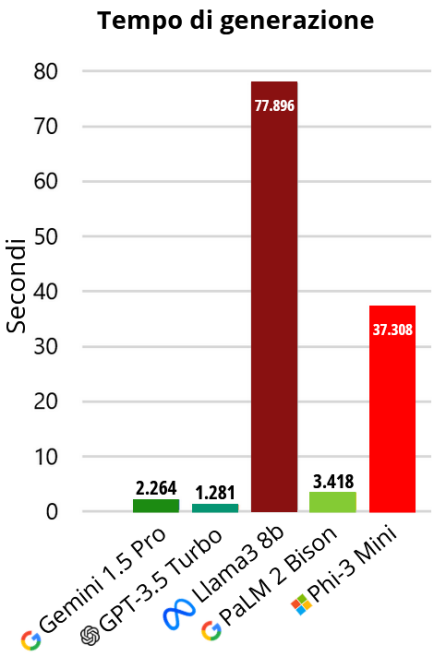


Figura 3.15: Tempo di generazione medio delle risposte

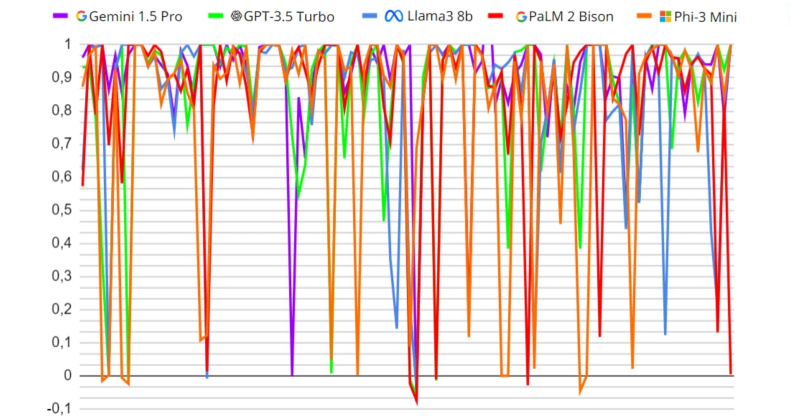


Figura 3.16: Valori progressivi della similarità semantica delle risposte

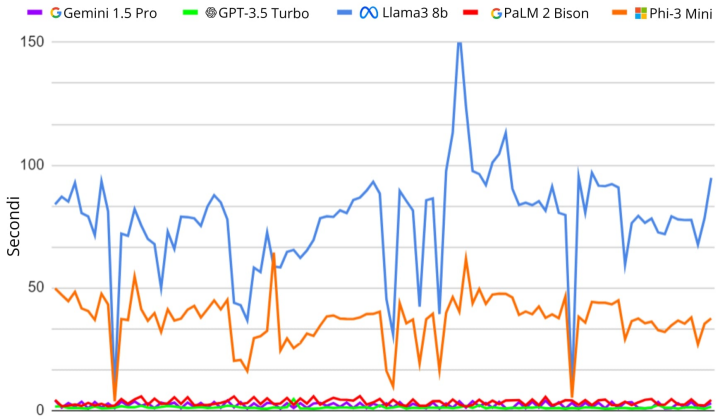


Figura 3.17: Valori progressivi del tempo di generazione delle risposte

Come facilmente osservabile dai grafici precedenti, il *Large Language Model* che ha registrato le prestazioni migliori nelle *task* del *benchmark* è Gemini 1.5 Pro. Questo modello non solo ha registrato il punteggio maggiore e la più alta percentuale di correttezza, ma è anche risultato il migliore per similarità semantica delle risposte fornite rispetto quelle attese e secondo per velocità di risposta.
Il secondo modello per *performance* è stato Llama3 8b, il migliore tra quelli eseguiti in locale, con risultati di poco superiori a quelli segnati da PaLM 2 Bison, modello *Google* della generazione precedente a quella di Gemini. Punteggio finale simile tra GPT-3.5 Turbo e Phi-3 Mini, la cui differenza tra i due risulta essere la qualità della risposta e, soprattutto, la velocità di risposta, migliori nel modello di *OpenAI*.
Interessante notare come il migliore modello risulti essere quello con il più elevato costo di utilizzo, con i valori di similarità semantica dei tre modelli *cloud* che rispettano l'ordine decrescente di costo, mostrando come la qualità dell'*output* di un modello è coerente a quanto richiesto per il servizio.

Nella stesura del documento di report delle attività di valutazione dei modelli, uno dei prodotti attesi dallo *stage*, non ho potuto non contestualizzare i valori del tempo di generazione delle risposte, riportati nelle figure 3.15 e 3.17. I *LLM* che hanno necessitato di più tempo per generare una risposta sono entrambi modelli eseguiti *on-premise* localmente. Questo non deve infatti sorprendere, in quanto le *performance* di un modello linguistico eseguito in locale sono fortemente influenzate dalle risorse *hardware* disponibili del dispositivo nel quale sono installati. Questa limitazione non è invece presente quando accediamo ad un *Large Language Model* proprietario attraverso un'*API*, in quanto è ospitato ed eseguito in *server* in remoto appositamente progettati per gestire alti carichi di lavoro.

Un'altra importante considerazione che è possibile fare, analizzando i risultati finali dei modelli nel *benchmark*, riguarda l'elevata percentuale di correttezza delle risposte generate. Il *LLM* che ha fornito il minor numero di risposte corrette, ovvero Phi-3 Mini, ha comunque ottenuto un'ottima percentuale di precisione del 88%, nonostante esso abbia una dimensione considerevolmente ridotta: meno della metà degli 8 miliardi di parametri di Llama3, secondo modello più piccolo valutato.
Questi risultati dimostrano l'enorme potenziale della *Retrieval-Augmented Generation*, che grazie alla ricerca e raccolta delle sole informazioni rilevanti è in grado di comporre un *prompt* relativamente piccolo da essere compreso anche dai modelli con meno parametri e una *context window* ridotta.

Infine, nel documento di report ho sottolineato come per poter effettuare l'analisi delle *performance* dei modelli quando utilizzati per la generazione dell'*output* nella *RAG*, ottenendo dei risultati direttamente confrontabili e coerenti, tutte le variabili relative alla *pipeline* del precesso di *Retrieval-Augmented Generation*, come *Embedding Model* utilizzato e valori soglia di rilevanza delle informazioni, sono stati mantenuti costanti per tutti e cinque i *LLM* valutati.
Potenzialmente, il *benchmark* da me prodotto potrebbe essere esteso alla valutazione dell'intero sistema di *information retrieval* implementato, e non del solo modello linguistico, andando a confrontare i risultati ottenuti nella combinazione di un *Large Language Model* al variare del modello di *embedding* utilizzato per vettorizzare i documenti del *dataset*, dei singoli parametri di ricerca delle informazioni rilevanti e della struttura del *prompt* dato al modello.

Capitolo 4

Conclusioni

4.1 Valutazione retrospettiva

Terminate tutte le attività previste dal progetto, ho svolto insieme al tutor aziendale una breve riunione conclusiva, nella quale è stato analizzato il grado di raggiungimento degli obiettivi dello *stage*, riassunto con la figura 4.1.
Tutti gli obiettivi stabiliti dal Piano di Lavoro, riportati nella sezione 2.4.2, sono stati pienamente soddisfatti, superando anche le attese iniziali. Non solo ho realizzato un prototipo *backend* integrante Phi-3 Mini, modello in locale, e un secondo con Gemini 1.5 Pro, ma ho reso accessibili tramite un prototipo anche i modelli PaLM 2 Bison e GPT-3.5 Turbo: il primo inizialmente non era previsto né richiesto dal progetto, mentre l'integrazione del *LLM* di *OpenAI* era prevista solo in un secondo momento, nel prototipo finale, come obiettivo desiderabile. Ho inoltre redatto un documento tecnico che descrive nel dettaglio il funzionamento dei prototipi, esponendo in modo chiaro e conciso il funzionamento del processo di *Relative-Augmented Generation* implementato.

Il prototipo finale, come richiesto dall'obiettivo obbligatorio, permette l'inserimento di documenti testuali e la loro consultazione tramite *chatbot*, e la sua realizzazione è terminata con la stesura di un documento tecnico, riportante le attività di analisi dei requisiti, progettazione e codifica. In questo prodotto *software* è stato implementato un menù per configurare il modello linguistico da utilizzare per generare la risposta del *chatbot*, soddisfacendo così anche l'ultimo obiettivo facoltativo. Inoltre, è stato valutato positivamente dal tutor aziendale l'aver soddisfatto tutti i requisiti funzionali, di vincolo e qualitativi del prototipo, riportati nella sezione 3.2.3.

Il *benchmark* da me realizzato ha costituito la base del confronto delle *performance* dei cinque modelli linguistici valutati: oltre ai quattro modelli già utilizzati nei prototipi, è stato aggiunto in un secondo momento Llama3 8b, anch'esso non previsto

inizialmente nel Piano di Lavoro. La decisione di utilizzare e valutare più modelli è stata accolta molto positivamente dall'azienda, in quanto raccogliendo più dati e allargando il confronto a nuovi *LLM* ho avuto modo di formulare considerazioni sui risultati finali maggiormente approfondite e accurate.

Infine, è stato riscontrata la realizzazione di tutti i prodotti attesi dal progetto, presentati nella sezione 2.4.3, riportati nella tabella 4.1 con un valore indicativo della loro dimensione: i prodotti *software* sono stati misurati in linee di codice (*Lines of Code*, o *LoC*), mentre quelli documentali in pagine.

Attività	**Prodotto**	**Dimensione**
Prototipazione rapida	Prototipi *backend* con Gemini 1.5 Pro, GPT-3.5 Turbo, PaLM 2 Bison e Phi-3 Mini	450 *LoC*
	Documentazione tecnica sui prototipi *backend*	15 pagine
Prototipazione finale	Prototipo finale - *frontend*	800 *LoC*
	Prototipo finale - *backend*	550 *LoC*
	Documentazione tecnica sul prototipo finale *backend*	40 pagine
Confronto dei modelli	Ambiente di *benchmarking*	200 *LoC*
	Documento di report della valutazione dei *LLM*	80 pagine

Tabella 4.1: Prodotti realizzati durante il progetto

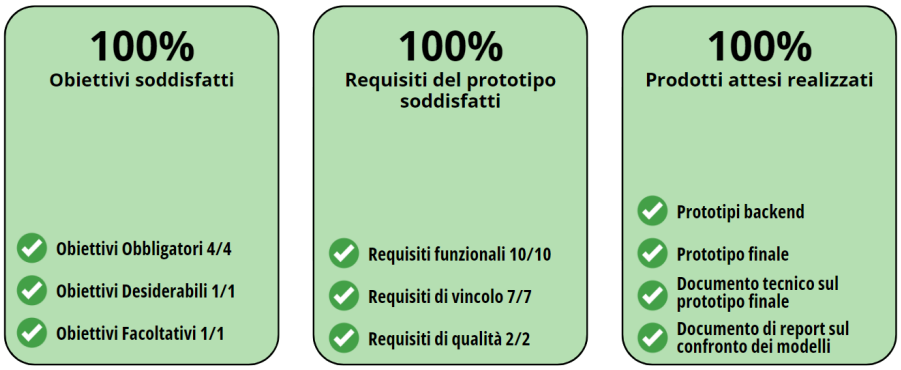


Figura 4.1: Grado di raggiungimento degli obiettivi del progetto

4.2 Maturazione personale

Il progetto di *stage* si è rivelato estremamente proficuo e formativo, permettendomi di acquisire nuove competenze e di affinarne altre, ampliando il mio bagaglio conoscitivo accademico. Grazie a questa esperienza ho avuto modo di programmare per la prima volta in *Java*, imparando come applicare i suoi principali *design pattern* per realizzare prodotti robusti e di qualità. Ho potuto inoltre esplorare a fondo tematiche interessanti come la *Generative AI* e la *Retrieval-Augmented Generation*, tecnologie che hanno già e avranno ancora in futuro un enorme impatto nel mondo della *computer science*.

Oltre ad avermi permesso di maturare nuove competenze e conoscenze in ambito informatico, questo progetto è stato utile per la mia crescita personale perché mi ha dato l'occasione per interfacciarmi al mondo del lavoro, sicuramente diverso per dinamiche da quello accademico. Relazionandomi al mio tutor aziendale in qualità di superiore, collaborando e discutendo sui temi del progetto con altri stagisti in Sync Lab, ho potuto acquisire molte *soft skill*, maturando le mie capacità comunicative ed organizzative.
Altrettanto formativa è stata la libertà che mi è stata concessa con lo studio autonomo delle tecnologie chiave dello *stage*, la quale mi ha permesso di proporre personalmente l'utilizzo di alcune di esse al mio tutor aziendale, in colloqui alla pari in cui ho potuto dimostrare le mie competenze sulle tematiche trattate. Grazie alla grande considerazione per il lavoro da me svolto, in questi confronti con il tutor e gli altri tirocinanti ho potuto discutere dei possibili sviluppi dei *Large Language Model* esponendo il mio personale punto di vista e imparando dalle considerazioni altrui.

In riferimento agli obiettivi personali, esposti in precedenza nella sezione 2.5, posso ritenermi pienamente soddisfatto, in quanto sono stati entrambi soddisfatti. Sono riuscito a raggiungere tutti gli obiettivi definiti dal progetto di *stage*, realizzando tutti i prodotti attesi e soddisfacendo ogni requisito del prototipo finale. Inoltre, a maggior dimostrazione della bontà e qualità del mio lavoro, riuscendo a valutare e confrontare dei modelli linguistici aggiuntivi, e a redarre un ulteriore documento descrittivo dei prototipi *backend* e del funzionamento della *Retrieval-Augmented Generation*, ho mostrato di aver eseguito con efficienza e rigore le attività del progetto tanto da raccogliere un anticipo rispetto le attese del Piano di Lavoro che mi ha permesso di fare attività addizionali.
Sempre grazie all'anticipo accumulato nel corso del progetto, ho avuto l'occasione di soddisfare anche il mio secondo personale obiettivo, ovvero la realizzazione di

un *benchmark* su cui basare la valutazione comparativa dei *LLM*. Aver studiato e compreso il funzionamento di *benchmark* esistenti, fino alle realizzazione di un ambiente di *benchmarking* semi-automatico, ha rappresentato per me una grande fonte di maturazione personale, perché ho potuto acquisire una conoscenza approfondita dei moderni metodi di *evaluation* dei *LLM*.

4.3 Università e mondo del lavoro

Osservando lo svolgimento di questo progetto in relazione alla mia esperienza universitaria, ho potuto constatare il legame che intercorre tra il mondo lavorativo e quello accademico. In entrambi ho avuto modo di comprendere l'importanza dell'autonomia, incontrata nello studio e nella formazione personale sulle tecnologie necessarie a realizzare un progetto. Questo aspetto è chiaramente più accentuato nel mondo del lavoro, ma la medesima necessità l'ho incontrata nello svolgimento di alcuni progetti didattici all'Università, nella quale le lezioni hanno costituito solo la base per l'approfondimento di una tecnologia. Lo stesso è avvenuto durante lo *stage*, dove ho avuto modo di utilizzare le mie conoscenze pregresse riscontrando comunque la necessità di ampliarle attraverso una formazione autonoma.
Un altro legame che ho notato tra Università e mondo del lavoro l'ho identificato nell'importanza data all'organizzazione. L'esperienza maturata attraverso il progetto didattico del corso di Ingegneria del *Software*, nel quale ho imparato l'importanza di un solido *way of working* e di una buona pianificazione delle attività, è stata utile durante lo svolgimento dello *stage*, permettendomi di rispettare puntualmente tutte le scadenze del progetto.

Il mondo universitario e quello lavorativo risultano quindi strettamente legati e interconnessi, in quanto quello che impari nel primo è fondamentale e applicabile nel secondo. Sono altresì complementari tra loro, in quanto senza questa esperienza di *stage* il mio percorso accademico sarebbe risultato incompleto, in quanto la sfera maggiormente teorica dell'Università perderebbe valore se non integrata con la pratica di un ambiente lavorativo.

Glossario

Agile Modello di sviluppo *software*, caratterizzato dall'elevata importanza data al dialogo continuo tra sviluppatore e *stakeholder*. Lo stile *Agile* punta fortemente sulla capacità di saper adattare un progetto a imprevisti o cambiamenti, attraverso un'organizzazione flessibile delle attività in *sprint*. 4

Application Programming Interface Insieme di procedure e meccanismi che permettono la comunicazione tra *software*, definendo protocolli specifici per la strutturazione delle richieste e delle risposte tra i componenti in dialogo. 24

Bcrypt Funzione di *hashing* per *password*, basato sull'algoritmo di cifratura *Blowfish*. La sua affidabilità e robustezza deriva dall'utilizzo di valori di *salt* utilizzato nella cifratura delle chiavi, che permettono la generazione di *hash* sempre diversi a partire dalla medesima *password*. Questa funzione è inoltre efficace contro attacchi *brute force*, permettendo l'aumento della complessità computazionale dell'*hashing*, rallentando questa classe di attacchi. 34

Bias In riferimento all'Intelligenza Artificiale, forma di errore dovuto da assunzioni errate durante il processo di apprendimento automatico, causato dalla presenza di informazioni distorte da pregiudizi umani nel *training dataset*. 12

Deep Learning Branca del *Machine Learning* che, utilizzando reti neurali per imitare il funzionamento del cervello umano, studia come rendere i *computer* capaci di analizzare e apprendere grandi moli di dati. 10

Framework Applicazione o struttura di supporto che fornisce un insieme di strumenti, librerie e *best practice* per semplificare lo sviluppo *software*. 2

HyperText Transfer Protocol Principale protocollo di comunicazione utilizzato per trasmettere informazioni sul *web*. La comunicazione tra *client* e *server* avviene mediante lo scambio di richieste e risposte. 24

International Organization for Standardization Organizzazione a livello mondiale per la definizione di norme tecniche e *standard* di qualità. 3

On-Premise Installazione, gestione ed esecuzione di un *software* su un *server* fisico di una organizzazione, in contrapposizione all'utilizzo di servizi esterni con accesso a *server* remoti. 12

Repository Archivio digitale centralizzato per la conservazione, la gestione, la condivisione e il versionamento di codice sorgente. 23

Stakeholder Persona, gruppo o organizzazione interessato e direttamente coinvolto in un progetto o attività, influenzandone la realizzazione. 4

Unified Modeling Language Linguaggio di modellazione visivo *standard* finalizzato all'analisi e descrizione di sistemi *software*. 29

Unsupervised Learning Tecnica di *Machine Learning* nella quale si fornisce al sistema di Intelligenza Artificiale un *training dataset* senza alcuna supervisione umana del processo. 11

Vocabolario Raccolta di tutti i *token* riconosciuti e utilizzati da un modello linguistico: ad ognuno di essi è associato un valore numerico univoco. 11

Riferimenti

Bibliografia

[1] Google. *PaLM 2 Technical Report*. arXiv preprint arXiv:2305.10403, 2023 (cit. a p. 13).

[2] Gemini Team Google. *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*. arXiv preprint arXiv:2403.05530, 2024 (cit. a p. 13).

[3] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv preprint arXiv:2005.11401, 2021 (cit. a p. 14).

[4] Felipe Maia Polo et al. *tinyBenchmarks: evaluating LLMs with fewer examples*. arXiv preprint arXiv:2402.14992, 2024.

[5] Darrell Rigby, Jeff Sutherland e Hirotaka Takeuchi. *How to master the process that's transforming management*. Harvard Business Review, 2016, pp. 40–48 (cit. alle pp. 4, 5).

[6] Amit Singhal. *Modern Information Retrieval: A Brief Overview*. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24, 2001, pp. 46–37 (cit. a p. 16).

[7] Microsoft Phi Team. *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*. arXiv preprint arXiv:2404.14219, 2024 (cit. a p. 13).

[8] Ashish Vaswani et al. *Attention is All you Need*. arXiv preprint arXiv:1706.03762, 2017 (cit. alle pp. 11, 12).

[9] Wenhui Wang et al. *MINILM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers*. arXiv preprint arXiv:2002.10957, 2020 (cit. a p. 26).

Sitografia

[10] *A Comprehensive Guide to Large Language Models*. URL: https://blog.fabrichq.ai/a-comprehensive-guide-to-large-language-models-4b202b50abc8 (cit. a p. 11).

[11] *Glossary of LLM and Generative AI*. URL: https://medium.com/@shuchaobi/glossary-of-llm-and-generative-ai-b3111da41da7.

[12] *GPT 3.5 Turbo - OpenAI Models*. URL: https://platform.openai.com/docs/models/gpt-3-5-turbo (cit. a p. 13).

[13] *Introducing Meta Llama 3: The most capable openly available LLM to date*. URL: https://ai.meta.com/blog/meta-llama-3/ (cit. a p. 13).

[14] *LangChain4J - Integrations*. URL: https://docs.langchain4j.dev/integrations/ (cit. a p. 24).

[15] *Manifesto Agile*. URL: https://agilemanifesto.org/iso/it/manifesto.html (cit. a p. 4).

[16] *Sync Lab srl*. URL: https://www.synclab.it/home (cit. a p. 1).

[17] *Transformer (deep learning architecture)*. URL: https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture) (cit. a p. 12).

[18] *What is Retrieval-Augmented Generation*. URL: https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/ (cit. a p. 14).