Alma Mater Studiorum
Università di Bologna
Campus di Cesena

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

Sviluppo di un'applicazione per analisi documentale tramite AI

Relatore:
Prof.ssa Catia Prandi

Presentata da:
Javid Ameri

Correlatore:
Stefano Cemin

Sessione I
Anno Accademico 2024-2025

Ai miei amici,
per aver alimentato l'ardore della speranza
anche nelle piogge più impetuose.

Introduzione

Negli ultimi anni, l'intelligenza artificiale ha conosciuto un progresso straordinario, trasformando numerosi settori e offrendo strumenti capaci di automatizzare attività complesse, analizzare grandi quantità di dati e migliorare l'esperienza d'uso degli utenti. In questo contesto, i modelli linguistici di grandi dimensioni si sono affermati come tecnologie in grado di comprendere e generare linguaggio naturale in modo sempre più accurato, rendendo l'interazione tra esseri umani e macchine più intuitiva ed efficace.

Tuttavia, l'adozione di questi strumenti solleva interrogativi importanti legati alla privacy, alla trasparenza e al controllo dei dati, soprattutto quando le applicazioni si basano su infrastrutture cloud. In risposta a tali sfide, si sta diffondendo un approccio orientato all'esecuzione locale dei modelli, che consente di preservare la riservatezza delle informazioni e garantire una maggiore autonomia operativa.

La presente tesi si inserisce in questo scenario con lo sviluppo di un sistema prototipale per l'interrogazione e l'analisi di documenti testuali tramite un'interfaccia conversazionale basata su modelli di intelligenza artificiale eseguiti in locale. L'obiettivo principale è fornire uno strumento che consenta agli utenti di consultare facilmente documenti complessi attraverso domande in linguaggio naturale, migliorando l'accessibilità alle informazioni e tutelando al contempo la privacy.

Il **Capitolo 1** presenta il contesto teorico e tecnologico alla base del progetto, introducendo le principali nozioni sull'intelligenza artificiale, sui modelli linguistici e sulle tecniche di elaborazione del linguaggio naturale,

nonché le motivazioni che hanno portato alla scelta di un'architettura locale per garantire riservatezza e trasparenza.

Il **Capitolo 2** è dedicato all'analisi e progettazione del sistema, illustrando gli obiettivi del progetto, i requisiti funzionali e non funzionali individuati e le scelte architetturali compiute, con una panoramica sulle componenti principali che costituiscono l'applicativo.

Il **Capitolo 3** descrive nel dettaglio l'implementazione del sistema, spiegando le modalità di sviluppo del frontend e del backend, le tecniche adottate per l'elaborazione e l'indicizzazione dei documenti, e le modalità di interrogazione semantica dei testi attraverso l'interfaccia conversazionale.

Attraverso questo lavoro si intende offrire un contributo pratico alla ricerca di soluzioni etiche e sostenibili per l'utilizzo dell'intelligenza artificiale nell'analisi documentale, dimostrando come sia possibile sviluppare strumenti avanzati che combinino efficienza, semplicità d'uso e tutela della privacy dell'utente.

Indice

Elenco delle figure

Capitolo 1

Contesto

1.1 Tecnologie digitali e automazione: contesto generale

Negli ultimi decenni, l'informatica ha rivoluzionato la società moderna, trasformandosi da disciplina accademica a pilastro essenziale di ogni settore produttivo. Questo processo viene descritto da Brynjolfsson e McAfee, due tra i più autorevoli studiosi contemporanei nel campo dell'economia digitale, come l'avvento di una *Second Machine Age*, in cui l'automazione non si limita più alle attività fisiche, ma si estende anche a quelle cognitive, ridefinendo il ruolo del lavoro umano e le dinamiche produttive [1]. Il passaggio da sistemi di calcolo deterministici a tecnologie intelligenti ha segnato una svolta epocale nel modo in cui affrontiamo problemi complessi, gestiamo grandi quantità di dati e prendiamo decisioni.

Le tecnologie digitali hanno abilitato nuove forme di automazione, a partire dall'elaborazione meccanica dei dati fino ad arrivare all'apprendimento automatico e all'adattamento dei sistemi in base all'ambiente. In particolare, l'automazione dei processi informativi rappresenta oggi una delle principali sfide organizzative, soprattutto in contesti caratterizzati da un'elevata densità documentale, come la pubblica amministrazione, il settore legale e quello sanitario.

Alla base di questa trasformazione troviamo concetti fondamentali come gli algoritmi, ossia sequenze finite di istruzioni eseguibili da una macchina per risolvere un problema, e le strutture dati, che organizzano l'informazione in modo efficiente. Questi strumenti si sono evoluti in sistemi in grado di apprendere autonomamente dai dati: le cosiddette tecniche di *machine learning*, che rappresentano una branca dell'intelligenza artificiale.

Parallelamente, l'aumento esponenziale della disponibilità di dati (*big data*) ha spinto l'informatica verso modelli più flessibili e adattivi, in grado di estrarre informazione utile da contenuti eterogenei e non strutturati. È in questo contesto che si colloca lo sviluppo dell'intelligenza artificiale, che rappresenta l'approdo di una lunga evoluzione tecnologica e teorica, ponendosi oggi come tecnologia abilitante trasversale a molteplici settori.

1.2 L'Intelligenza Artificiale: definizioni e prospettive

1.2.1 Evoluzione storica dell'AI

L' **intelligenza artificiale (AI)** ha attraversato diverse fasi evolutive sin dagli anni '50, quando il matematico e pioniere dell'informatica Alan Turing si pose la celebre domanda *"le macchine possono pensare?"*. Il suo lavoro, insieme a quello di altri pionieri come John McCarthy (che coniò il termine *artificial intelligence* nel 1956), Marvin Minsky e Norbert Wiener (fondatore della cibernetica), ha tracciato le fondamenta teoriche di una disciplina che ancora oggi si interroga sulla natura dell'intelligenza.

Nel corso dei decenni, l'intelligenza artificiale ha vissuto fasi alterne di sviluppo, ma è attualmente in una fase di rinascita. Come affermano Michael Haenlein e Andreas Kaplan, in un articolo pubblicato sulla rivista *California Management Review*, *"l'attuale ondata di interesse per l'intelligenza*

artificiale è stata alimentata da tre fattori: la disponibilità di big data, i miglioramenti nella potenza di calcolo e i progressi negli algoritmi di apprendimento automatico" [2]. Questa combinazione di elementi ha reso possibile lo sviluppo di sistemi intelligenti in grado di affrontare compiti complessi e generare impatti concreti nei contesti produttivi, scientifici e sociali.

1.2.2 Definizioni e classificazioni

Una definizione ampiamente accettata di AI è quella fornita da da Stuart Russell e Peter Norvig, due tra i principali esperti nel campo dell'intelligenza artificiale e autori del noto manuale universitario Artificial Intelligence: A Modern Approach. Questa descrive l'intelligenza artificiale come *"la disciplina che studia la progettazione e lo sviluppo di sistemi in grado di svolgere compiti che, se effettuati da esseri umani, richiederebbero intelligenza"*. [3].

Esistono diverse modalità di classificazione dell'AI. Una prima distinzione fondamentale, come riconosciuto da Norvig e Russell [3], è tra **AI debole** (*narrow AI*), progettata per svolgere compiti specifici (come il riconoscimento vocale o la diagnosi medica), e **AI forte** (*strong AI*), che mira a replicare le capacità cognitive generali dell'essere umano, inclusa la coscienza e l'autoconsapevolezza.

Per chiarire ulteriormente, l'AI debole è quella oggi più diffusa: include sistemi come i motori di raccomandazione (Netflix, Amazon), assistenti vocali (Siri, Alexa), sistemi di visione artificiale e strumenti di traduzione automatica. L'AI forte, invece, è ancora oggetto di ricerca teorica e non ha ancora trovato realizzazioni concrete: mira a riprodurre le capacità cognitive umane in senso ampio, inclusa la coscienza e l'autoconsapevolezza.

1.2.3 Applicazioni dell'AI nei contesti industriali e aziendali

Le applicazioni dell'AI sono ormai pervasive: dalla manutenzione predittiva in ambito industriale, all'automazione dei processi decisionali nei settori

finanziario, legale e sanitario. In particolare, l'analisi documentale rappresenta un ambito chiave in cui l'AI contribuisce a migliorare l'efficienza operativa e la qualità dell'accesso all'informazione [4].

Nel settore sanitario, ad esempio, l'AI è utilizzata per l'analisi di immagini mediche, la diagnosi assistita e l'estrazione automatica di informazioni cliniche dai referti. Tali applicazioni sono oggetto di crescente attenzione anche nella letteratura scientifica, dove l'AI viene riconosciuta come uno strumento in grado di aumentare l'accuratezza diagnostica e l'efficienza dei percorsi clinici [5]. In ambito finanziario, l'automazione dei processi di verifica contrattuale, di conformità e di gestione del rischio è diventata realtà grazie a sistemi intelligenti in grado di apprendere da casistiche storiche. Anche nella pubblica amministrazione si stanno diffondendo strumenti di AI per la gestione documentale, l'assistenza virtuale e il supporto decisionale basato su dati.

1.3 L'elaborazione del linguaggio naturale

1.3.1 Cos'è il Natural Language Processing (NLP)

Il **Natural Language Processing** (NLP) è un settore dell'intelligenza artificiale che si occupa dell'interazione tra il linguaggio umano e i sistemi computazionali. L'obiettivo principale del NLP è quello di consentire alle macchine di comprendere, interpretare e generare linguaggio naturale (cioè quello utilizzato quotidianamente dagli esseri umani), trattandolo in modo simile a come farebbe una persona. Come spiegano i linguisti computazionali Daniel Jurafsky e James H. Martin nel manuale *Speech and Language Processing*, il fine del NLP è proprio quello di permettere ai computer di analizzare testi e discorsi in modo simile a un essere umano [6].

Le attività tipiche del NLP includono il riconoscimento del linguaggio naturale (speech recognition), l'analisi grammaticale e sintattica (parsing),

l'estrazione di informazioni (information extraction), la classificazione di testi, l'analisi del sentimento e la traduzione automatica. Negli ultimi anni, il NLP ha raggiunto livelli di precisione prima inimmaginabili, tuttavia, nonostante i progressi, vi sono ancora numerose sfide da affrontare. Tra queste vi sono l'ambiguità semantica (parole con significati diversi a seconda del contesto), la gestione di linguaggi specialistici e il trattamento di lingue con risorse limitate.

1.3.2 Applicazioni del NLP in ambito documentale

Nel contesto aziendale e istituzionale, il NLP viene impiegato per semplificare la gestione della conoscenza. Tra le applicazioni più rilevanti troviamo l'automazione dell'analisi contrattuale, la lettura assistita di normative complesse, la creazione di riassunti automatici, la classificazione tematica dei documenti e l'assistenza virtuale per la consultazione di archivi testuali.

Tali strumenti permettono di ridurre il carico cognitivo degli utenti e di accelerare i processi decisionali. In ambito accademico, numerosi studi hanno dimostrato come l'impiego di tecnologie NLP nella gestione documentale consenta di migliorare la produttività e ridurre l'errore umano [7]. Ad esempio, un'impresa può utilizzare un sistema NLP per individuare automaticamente clausole critiche nei contratti, oppure un ente pubblico può analizzare grandi volumi di segnalazioni dei cittadini per estrarre le problematiche più ricorrenti.

1.4 Modelli di linguaggio di grandi dimensioni (LLM)

1.4.1 Principi di funzionamento

I **modelli di linguaggio di grandi dimensioni** (Large Language Models, **LLM**) sono reti neurali profonde addestrate su enormi quantità di testo per apprendere le regolarità linguistiche e semantiche del linguaggio naturale. Come descritto dal ricercatore Tom B. Brown [8], la loro struttura si basa su un approccio probabilistico: dato un input testuale, il modello predice la parola successiva più plausibile in base al contesto. Questo meccanismo di generazione sequenziale è alla base delle loro capacità conversazionali, di riassunto, traduzione e risposta a domande.

I LLM operano su rappresentazioni numeriche del linguaggio chiamate *embedding*, che permettono di trasformare le parole in vettori. Queste rappresentazioni catturano relazioni tra parole, frasi e concetti, consentendo al modello di trarne il significato anche in contesti complessi.

Il termine "grandi dimensioni" si riferisce a tre principali caratteristiche:

- **La dimensione del dataset di addestramento**: i LLM vengono esposti a centinaia di miliardi di parole, così da apprendere le strutture statistiche, semantiche e sintattiche del linguaggio.

- **Il numero di parametri**: si tratta delle variabili interne al modello che vengono ottimizzate durante la fase di apprendimento. I modelli più avanzati possono superare i 100 miliardi di parametri, rendendoli capaci di rappresentare relazioni linguistiche complesse.

- **La potenza computazionale necessaria**: l'addestramento e l'esecuzione (inferenza) di questi modelli richiedono notevoli risorse hardware, come una GPU, e infrastrutture di calcolo distribuito.

Queste caratteristiche rendono i LLM straordinariamente potenti, ma anche costosi in termini di tempo, energia e accessibilità. Per questo motivo, la ricerca recente si sta concentrando sullo sviluppo di modelli più leggeri, efficienti e facilmente distribuibili in ambiti produttivi, senza sacrificare la qualità delle prestazioni.

1.4.2 Architetture: reti neurali e Transformer

Alla base del funzionamento dei modelli linguistici di grandi dimensioni vi sono le reti neurali artificiali, strutture computazionali ispirate al funzionamento del cervello umano. Una rete neurale è composta da nodi (neuroni artificiali) organizzati in strati (di input, nascosti e di output), dove ogni neurone riceve un segnale in ingresso e, dopo averlo elaborato, ne trasmette il risultato agli strati successivi. Attraverso ripetute esposizioni ai dati, la rete impara ad associare input e output modificando progressivamente i "pesi" delle connessioni tra i neuroni. Questo processo di apprendimento permette alla rete di rappresentare anche relazioni molto complesse tra le informazioni, migliorando le proprie prestazioni nel tempo.

Nel contesto del Natural Language Processing, lo sviluppo delle architetture neurali ha portato a soluzioni sempre più efficaci nella gestione del linguaggio. Una svolta significativa si è avuta nel 2017 con l'introduzione del modello **Transformer**, da parte di Ashish Vaswani e il suo team di ricerca [9], il quale ha segnato una svolta radicale rispetto ai modelli precedenti.

Il principale vantaggio dell'architettura Transformer risiede nella sua capacità di elaborare intere sequenze in parallelo (ossia analizzando simultaneamente tutte le parole della frase, invece di procedere una parola alla volta), rendendo il processo di addestramento più efficiente rispetto agli approcci ricorrenti. Inoltre, il meccanismo di self-attention permette al modello di pesare dinamicamente l'importanza di ciascun termine rispetto agli altri, migliorando la comprensione del contesto semantico.

Queste innovazioni hanno contribuito a rendere l'elaborazione del linguaggio naturale più efficiente, accessibile e versatile, aprendo la strada a nuove

applicazioni in ambito industriale, scientifico ed educativo.

1.4.3 Modelli noti: GPT, Claude, LLaMA, Mistral e altri

Negli ultimi anni, l'ecosistema dei modelli di linguaggio di grandi dimensioni (LLM) si è arricchito notevolmente, con soluzioni proposte da grandi aziende tech e realtà emergenti. Di seguito una panoramica sintetica dei principali modelli:

GPT. Sviluppato da OpenAI, il modello *Generative Pretrained Transformer*[1] ha visto diverse versioni fino a GPT-4, che introduce capacità multimodali (cioè la possibilità di elaborare sia testo che immagini) e prestazioni avanzate in vari compiti. I GPT sono noti per l'uso del *fine-tuning* supervisionato (una fase di riaddestramento del modello su compiti specifici usando dati etichettati) e del *reinforcement learning from human feedback* – RLHF (una tecnica in cui il modello viene ottimizzato sulla base di valutazioni fornite da esseri umani). Sono accessibili sia tramite API che attraverso interfacce conversazionali come ChatGPT.

Claude. Claude[2] è il modello sviluppato da Anthropic, un'azienda fondata da ex membri di OpenAI, che pone particolare enfasi sulla sicurezza e l'allineamento etico dell'AI. Claude è progettato per fornire risposte più sicure e spiegabili, con un approccio all'addestramento basato su principi dichiarativi e supervisione umana.

LLaMA. LLaMA[3] (Large Language Model Meta AI) è una serie di modelli rilasciati da Meta. A differenza di GPT, LLaMA è pensato per essere più leggero ed efficiente, con versioni da 7, 13 e 70 miliardi di parametri. I pesi

[1] `https://openai.com/gpt-4`

[2] `https://www.anthropic.com/index/claude`

[3] `https://ai.meta.com/llama`

del modello (numeri che determinano quanto "importante" è un'informazione per il modello quando prende decisioni) sono resi disponibili pubblicamente, rendendolo popolare nella comunità open source e adatto a esecuzioni locali su hardware meno costoso.

Mistral. Mistral[4] AI ha realizzato modelli come Mistral 7B, progettati per massimizzare l'efficienza computazionale (cioè l'uso ottimale di risorse hardware). Il modello utilizza tecniche come la *quantizzazione* (riduzione della precisione numerica per velocizzare l'elaborazione) e il *pruning* (rimozione di connessioni inutili nella rete neurale) per ridurre le dimensioni senza compromettere la qualità.

Altri modelli emergenti. Tra i modelli recenti più leggeri e accessibili si segnalano anche:

- **Gemma (Google)**[5]: orientato alla trasparenza e responsabilità, disponibile con licenze permissive;
- **DeepSeek**[6]: sviluppato con l'obiettivo di combinare efficienza e accuratezza, usato anche nella ricerca accademica;
- **Phi (Microsoft)**[7]: noto per l'ottimo compromesso tra dimensione contenuta e qualità linguistica.

Il confronto tra questi modelli mostra come il campo sia in rapida evoluzione e ricco di sperimentazioni, con una crescente attenzione non solo alle prestazioni, ma anche a temi come l'efficienza, la sicurezza, l'apertura e la responsabilità.

[4] https://mistral.ai/news/introducing-mistral-7b

[5] https://ai.google.dev/gemma

[6] https://deepseek.com/

[7] https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/

1.4.4 Vantaggi e limiti degli LLM

I modelli di linguaggio di grandi dimensioni (LLM) rappresentano uno dei traguardi più significativi dell'intelligenza artificiale contemporanea. Tuttavia, il loro impiego solleva una serie di considerazioni sia di ordine tecnico che etico. In questa sezione analizziamo i principali vantaggi offerti da questi modelli, nonché le criticità che ancora li caratterizzano.

Vantaggi:

1. Versatilità nei compiti testuali. Gli LLM possono affrontare una vasta gamma di attività testuali: dalla generazione di contenuti alla traduzione, dal riassunto alla risposta a domande. Questa capacità consente di applicarli continuamente a nuovi scenari.

2. Comprensione contestuale. Grazie alla loro architettura Transformer e alla grande quantità di dati su cui sono addestrati, gli LLM sono in grado di riconoscere il contesto (anche in scenari complessi) e mantenere la coerenza del discorso anche in testi lunghi o articolati.

3. Riduzione del carico cognitivo. In ambienti professionali, gli LLM possono fungere da assistenti intelligenti, semplificando l'accesso alla conoscenza e automatizzando attività ripetitive come la lettura di documenti, l'estrazione di informazioni e la redazione di testi. Questo comporta un risparmio di tempo e (in alcuni casi) una riduzione degli errori umani.

4. Accessibilità estesa. La disponibilità di modelli open-source e di strumenti di facile utilizzo ha reso l'accesso all'AI possibile anche per utenti non esperti. Ciò potrebbe aprire nuove opportunità ad esempio in enti pubblici e settori educativi, tradizionalmente meno esposti alle tecnologie avanzate.

Limiti:

1. Opacità e interpretabilità limitata. Gli LLM sono spesso descritti come *black box*: pur producendo risultati di alta qualità, il meccanismo con cui giungono a una determinata risposta è difficilmente interpretabile. Questo pone problemi di affidabilità, specialmente in ambiti sensibili come il diritto o la medicina.

2. Rischio di allucinazioni. Una delle criticità più discusse è la capacità dei modelli di "inventare" fatti o informazioni apparentemente plausibili ma non veritiere. Questo fenomeno, detto *hallucination*, è particolarmente problematico in contesti dove è essenziale l'accuratezza delle informazioni fornite [10].

3. Bias e imparzialità. Poiché gli LLM vengono addestrati su grandi copri di testo provenienti dal web, riflettono (e talvolta amplificano) i pregiudizi culturali, politici e sociali presenti nei dati. La presenza di *bias* (distorsione della valutazione causata da pregiudizi) può influenzare la qualità delle risposte e produrre effetti discriminatori se non gestita adeguatamente.

4. Impatto ambientale e computazionale. L'addestramento e l'esecuzione di LLM richiedono risorse computazionali considerevoli, con costi economici ed energetici elevati. Secondo stime recenti, riportate da un'analisi pubblicata su Scientific American, l'addestramento di un singolo modello come GPT-3 ha comportato l'emissione di circa 502 tonnellate metriche di CO_2 equivalenti, sollevando interrogativi sulla sostenibilità di questi approcci [11].

5. Problemi legali e di copyright. La generazione automatica di contenuti può violare diritti di proprietà intellettuale, specialmente se i modelli attingono da dati non esplicitamente autorizzati. Inoltre, la responsabilità legale in caso di output dannosi o errati rimane un tema ancora poco regolamentato.

In sintesi, i LLM offrono capacità straordinarie per l'elaborazione del lin-

guaggio naturale, ma richiedono un utilizzo attento e consapevole. Le sfide legate alla trasparenza, all'affidabilità, all'equità e alla sostenibilità dovranno essere affrontate congiuntamente da sviluppatori, legislatori e utenti finali per garantire un impiego etico e responsabile di queste tecnologie.

1.4.5 Esecuzione in locale vs su cloud: impatti su privacy e prestazioni

Un aspetto rilevante nell'adozione dei modelli di linguaggio di grandi dimensioni (LLM) riguarda la modalità di esecuzione: locale o tramite cloud.

L'esecuzione su cloud (ad esempio tramite OpenAI) consente l'accesso immediato a modelli molto potenti, senza necessità di hardware dedicato. Offre scalabilità, aggiornamenti continui e semplicità d'uso. Tuttavia, comporta anche criticità in termini di privacy, dipendenza da terze parti e costi ricorrenti.

L'alternativa è l'esecuzione in locale, cioè direttamente su dispositivi dell'utente o dell'organizzazione. Questa soluzione — resa oggi più accessibile grazie a modelli ottimizzati come Mistral o LLaMA — offre vantaggi in termini di controllo dei dati, personalizzazione e indipendenza dal cloud, ma richiede anche risorse computazionali e competenze tecniche superiori.

Una riflessione più approfondita su questi temi, in particolare sugli aspetti legati alla sicurezza e alla normativa, sarà trattata nel paragrafo **1.7**.

1.5 Analisi documentale assistita da AI

1.5.1 Il problema della consultazione di documenti complessi

In ambito aziendale, legale, amministrativo e accademico, la gestione e l'analisi di documenti rappresentano attività centrali ma spesso onerose. Ma-

nuali tecnici, contratti, relazioni, atti normativi e referti medici costituiscono una mole crescente di informazione, che richiede tempo, attenzione e competenze specifiche per essere consultata efficacemente. La difficoltà cresce esponenzialmente con l'aumentare di fattori quali la lunghezza e il tecnicismo dei contenuti.

In molte situazioni operative, l'utente è chiamato a cercare all'interno di un documento risposte a domande specifiche — ad esempio, identificare clausole contrattuali, comprendere regolamenti, individuare obblighi normativi o confrontare il documento con altri. In assenza di strumenti automatizzati, tali attività sono effettuate manualmente, con costi in termini di tempo e rischio di errori. Il carico associato alla lettura di documentazione estesa incide negativamente sull'efficienza, sulla qualità delle decisioni e sul benessere dell'operatore.

1.5.2 Vantaggi dell'approccio basato su NLP e AI

Le tecnologie di NLP e AI offrono una risposta concreta a queste problematiche, permettendo di automatizzare la lettura e l'interpretazione di testi complessi. Un sistema di analisi documentale assistita può, ad esempio:

- Estrarre automaticamente passaggi rilevanti in risposta a una domanda in linguaggio naturale.
- Riassumere contenuti estesi, riducendo la quantità di testo da leggere.
- Classificare documenti in base al contenuto.
- Evidenziare concetti chiave, clausole critiche o incongruenze nei testi.

L'approccio semantico adottato dai moderni sistemi AI non si limita alla ricerca per parole chiave, ma si basa sulla comprensione del significato delle frasi. Ciò consente di gestire sinonimi, parafrasi e domande formulate in modo naturale, aumentando l'efficacia dell'interazione.

Questi vantaggi si traducono in un miglioramento della produttività, una maggiore accessibilità alla conoscenza (anche da parte di utenti non esperti) e un supporto decisionale più rapido e preciso.

1.5.3 Interfacce conversazionali per l'accesso alla conoscenza

Un'evoluzione particolarmente interessante dell'analisi documentale è rappresentata dall'integrazione con interfacce conversazionali basate su AI. In questo paradigma, l'utente non è più costretto a navigare manualmente un documento, ma può porre domande in linguaggio naturale (es. "dove si parla delle penali per ritardo?") e ricevere risposte puntuali, con riferimenti diretti ai passaggi pertinenti.

Questi sistemi combinano diverse tecnologie: da un lato il retrieval semantico (ossia la ricerca di contenuti basata sul significato, e non su semplici parole chiave) per identificare i blocchi di testo rilevanti, dall'altro la generazione automatica della risposta tramite modelli di linguaggio. L'interfaccia conversazionale funge da mediatore intelligente tra l'utente e il contenuto, trasformando il documento da oggetto statico a risorsa dinamica interrogabile.

Questo tipo di interazione rende l'accesso alla conoscenza più naturale, riduce la curva di apprendimento e democratizza l'utilizzo di documentazione tecnica, facilitando l'inclusione anche di utenti meno esperti o con disabilità cognitive/linguistiche.

1.5.4 Studi ed esperimenti rilevanti nel settore

Negli ultimi anni sono stati condotti diversi esperimenti per valutare l'efficacia di soluzioni basate su AI nella gestione documentale. Uno dei casi più rilevanti è quello condotto dal governo del Regno Unito, che ha sperimentato l'utilizzo di Microsoft 365 Copilot su un campione trasversale di dipartimenti pubblici. I risultati dello studio hanno mostrato che i partecipanti hanno

risparmiato in media 26 minuti al giorno nello svolgimento di attività amministrative legate alla lettura e comprensione dei documenti, corrispondenti a circa due settimane lavorative per persona su base annua [12].

Un altro studio, condotto da Yizhe Zhang (ricercatore presso Apple MLR), ha mostrato come interfacce conversazionali basate su LLM siano in grado di fornire risposte accurate e pertinenti su contenuti tecnici, migliorando significativamente la soddisfazione dell'utente rispetto ai sistemi tradizionali di ricerca testuale [13].

Esperienze simili sono emerse anche nel settore legale, dove start-up e grandi studi stanno adottando modelli AI per la revisione automatica dei contratti, con risultati paragonabili (in termini di accuratezza) a quelli ottenuti da revisori umani, ma in tempi notevolmente inferiori.

Queste evidenze sperimentali confermano il potenziale concreto delle tecnologie AI per rivoluzionare il modo in cui accediamo e interpretiamo l'informazione contenuta nei documenti, aprendo la strada a nuovi scenari di automazione cognitiva.

1.6 Retrieval semantico e pipeline di elaborazione

1.6.1 Dal parsing all'embedding

Per rendere possibile la consultazione intelligente di un documento tramite linguaggio naturale, è necessario costruire una *pipeline* di elaborazione composta da più fasi, ognuna delle quali contribuisce alla trasformazione del contenuto testuale in una forma interrogabile da un sistema AI.

Il primo passo è il **parsing** del documento, ovvero l'estrazione del testo da file PDF, Word o HTML e la sua conversione in formato leggibile. Que-

sta fase può includere l'analisi della struttura del documento (intestazioni, paragrafi, elenchi puntati, tabelle), utile per mantenere il contesto semantico e gerarchico delle informazioni.

Segue poi la fase di **segmentazione** o *chunking*, in cui il testo viene suddiviso in blocchi coerenti (es. paragrafi, frasi o sezioni tematiche). Il chunking consente di trattare frammenti del documento come unità di significato, semplificando il confronto con le query dell'utente.

Una volta segmentato, il contenuto viene trasformato in rappresentazioni numeriche mediante tecniche di **embedding semantico**. Questi embedding mappano ogni blocco testuale in uno spazio vettoriale ad alta dimensione, dove la distanza tra i vettori riflette la similarità semantica tra i testi [14].

1.6.2 Tecniche di similarità semantica

Il **retrieval semantico** consiste nel confrontare la rappresentazione vettoriale della domanda dell'utente con quelle dei blocchi di testo memorizzati. Lo strumento matematico più comunemente utilizzato per questo confronto è la *cosine similarity* (in italiano, similarità del coseno), che misura l'angolo tra due vettori nello spazio. Più l'angolo tra i vettori è piccolo (cioè più essi "puntano" nella stessa direzione), maggiore è la somiglianza semantica tra i testi rappresentati [14].

A differenza della ricerca per keyword, che si limita a identificare corrispondenze letterali, il retrieval semantico permette di recuperare frammenti di testo che esprimono concetti simili, anche se formulati in modo diverso. Ciò risulta particolarmente utile in ambiti dove la terminologia può variare (es. sinonimi, parafrasi) o in cui è necessario comprendere il significato latente delle domande.

1.6.3 Il ruolo della generazione linguistica nella risposta

Una volta identificati i blocchi di testo più rilevanti rispetto alla domanda posta, entra in gioco un **modello generativo** (tipicamente un LLM), incaricato di sintetizzare una risposta coerente e contestualizzata. Questo passaggio viene spesso denominato *retrieval-augmented generation* (RAG) [15].

Il modello LLM riceve come input sia la domanda dell'utente, sia i blocchi di testo recuperati. Su questa base, genera una risposta formulata in linguaggio naturale, spesso includendo citazioni testuali, riepiloghi o spiegazioni semplificate. L'obiettivo è restituire un output informativo, preciso e comprensibile, che riduca il bisogno di consultare direttamente l'intero documento.

Questo approccio garantisce all'utente un'esperienza conversazionale efficace.

1.6.4 Criticità e scelte progettuali nella costruzione di una pipeline AI

La costruzione di una pipeline di analisi documentale intelligente richiede scelte progettuali complesse. Tra le principali criticità vi sono:

- **Scalabilità**: documenti molto lunghi possono generare centinaia di chunk e quindi richiedere migliaia di confronti semantici, impattando sulle performance.
- **Qualità dell'embedding**: la precisione del retrieval dipende fortemente dalla qualità della rappresentazione semantica. Modelli generici possono fallire su domini specialistici.
- **Bilanciamento tra sintesi e citazione**: una risposta generata deve essere fedele al contenuto, ma anche chiara. È cruciale evitare allucinazioni o errori introdotti dal modello.

- **Sicurezza e privacy**: l'intero processo deve essere compatibile con requisiti di riservatezza, soprattutto quando i documenti contengono informazioni sensibili.

Una pipeline ben progettata affronta queste criticità combinando strumenti open source, modelli ottimizzati per il dominio e tecniche di caching e compressione semantica. In contesti ad alta criticità, si può optare per l'esecuzione locale dell'intero stack, sacrificando in parte la velocità per ottenere maggiore controllo sui dati.

1.7 Il ruolo della privacy e della sicurezza nei sistemi AI

1.7.1 Criticità legate all'uso di servizi cloud

L'amplia adozione di strumenti basati su AI ha portato con sé numerose sfide legate alla **gestione dei dati sensibili**. Molti dei servizi oggi disponibili, come i modelli generativi o le API di analisi testuale, operano in modalità *cloud-based*, ovvero elaborano le informazioni su server remoti di terze parti.

Tale architettura comporta vantaggi evidenti in termini di scalabilità (cioè capacità di adattarsi dinamicamente al numero di richieste), aggiornamenti continui e facilità di integrazione, ma introduce anche rischi significativi: una volta inviati, i dati escono dal controllo diretto dell'utente o dell'organizzazione, con possibili violazioni della riservatezza, fughe di dati o utilizzi impropri da parte dei fornitori del servizio [16].

Nei contesti ad alta sensibilità (es. sanità, finanza, pubblica amministrazione), questi rischi sono amplificati, poiché la documentazione gestita può contenere informazioni personali, dati biometrici, documenti legali o segreti aziendali. L'utilizzo di servizi cloud, senza adeguate misure di mitigazione,

può quindi risultare incompatibile con i principi di protezione dei dati imposti dalla normativa vigente.

1.7.2 Esecuzione locale come strategia di mitigazione

Una delle principali risposte a queste criticità è l'impiego di modelli linguistici e pipeline AI **in esecuzione locale**, cioè direttamente su dispositivi controllati dall'utente (es. laptop). Questo approccio, sebbene più oneroso in termini computazionali, consente di mantenere il controllo totale sui dati trattati.

L'esecuzione locale garantisce che nessuna informazione venga trasmessa a server esterni, eliminando alla radice il rischio di data leakage. Inoltre, permette una maggiore personalizzazione e una più efficace gestione della sicurezza, grazie all'integrazione diretta con le policy e le infrastrutture dell'ente o dell'azienda.

Va tuttavia sottolineato che l'elaborazione locale richiede risorse adeguate, sia hardware che software. Modelli come quelli della famiglia LLaMA o Mistral, progettati per funzionare in modo efficiente anche su dispositivi meno potenti, si prestano bene a questo scopo, soprattutto se si utilizzano tecniche per alleggerirli (come la quantizzazione) e li si esegue su schede grafiche di uso comune.

1.7.3 Quadro normativo europeo: GDPR e protezione dei dati

In Europa, il trattamento dei dati personali è regolato dal **Regolamento Generale sulla Protezione dei Dati (GDPR)**, in vigore dal 2018. Questo quadro normativo impone obblighi stringenti a chiunque raccolga, elabori o conservi dati personali di cittadini europei [17].

Tra i principi chiave del GDPR figurano:

- **Minimizzazione dei dati**: raccogliere solo i dati strettamente necessari per il fine dichiarato.

- **Trasparenza**: informare chiaramente l'utente su come e perché i dati vengono trattati.
- **Accountability**: dimostrare in ogni momento di operare in conformità con il regolamento.
- **Privacy by design e by default**: progettare sistemi che rispettino la privacy fin dalla fase di sviluppo.

L'utilizzo di AI comporta quindi una responsabilità aggiuntiva: oltre a garantire le performance tecniche, è necessario assicurarsi che le soluzioni rispettino il diritto alla privacy degli individui, evitando trattamenti automatizzati non giustificati o potenzialmente discriminatori.

1.7.4 Riflessioni etiche sull'utilizzo dell'AI per l'analisi documentale

Oltre alle questioni legali e tecniche, l'impiego dell'intelligenza artificiale nell'analisi documentale solleva anche interrogativi **etici**. Affidare a sistemi automatizzati il compito di leggere, interpretare e sintetizzare documenti può comportare rischi in termini di:

- **Affidabilità**: i modelli possono generare risposte scorrette, incomplete o fuorvianti.
- **Bias**: gli LLM possono riflettere pregiudizi impliciti nei dati su cui sono stati addestrati.
- **Trasparenza**: è spesso difficile spiegare perché un certo contenuto è stato selezionato o come è stata generata una risposta.
- **Sostituzione umana**: in alcuni ambiti, l'automazione può ridurre il ruolo di esperti umani, con possibili effetti sull'occupazione e sulla qualità dell'interpretazione.

Per affrontare queste sfide, si rende necessario adottare un approccio critico e multidisciplinare, in cui le competenze tecniche siano integrate da quelle giuridiche, filosofiche e sociologiche. La progettazione di strumenti AI per l'analisi documentale deve quindi bilanciare efficienza e responsabilità, innovazione e rispetto dei diritti, automazione e supervisione umana.

Capitolo 2

Analisi e Progettazione

2.1 Requisiti e obiettivi

2.1.1 Contesto aziendale e motivazioni progettuali

Il progetto è stato realizzato presso **Brainy Labs**, una giovane software house italiana specializzata nello sviluppo di soluzioni digitali personalizzate. L'azienda si distingue per un approccio snello e sperimentale, orientato all'apprendimento continuo, alla prototipazione rapida e all'adozione di tecnologie emergenti.

Brainy Labs promuove attivamente la collaborazione con studenti universitari attraverso tirocini, tesi di laurea e progetti open-source. Questo consente di testare sul campo nuove idee, confrontarsi con problematiche reali e contribuire concretamente alla formazione di giovani sviluppatori.

Nel caso specifico, l'azienda ha fornito il supporto tecnico e organizzativo per la realizzazione di una piattaforma basata su intelligenza artificiale, progettata principalmente per l'analisi automatica di documenti PDF. L'applicazione, che ho personalmente scelto di denominare **"JaJai"** (gioco di parole basato sul mio soprannome d'infanzia "JaJi" e il termine "AI"), è in grado

di interpretare e restituire contenuti in modo interattivo, attraverso un'interfaccia conversazionale moderna, accessibile e reattiva.

Figura 2.1: Logo ufficiale dell'applicativo, realizzato personalmente da me.

Uno degli aspetti centrali che hanno guidato la progettazione del sistema è stato il tema della **privacy e della sicurezza dei dati**. Negli ultimi anni, l'intelligenza artificiale ha ampliato la propria diffusione in numerosi ambiti della vita quotidiana, dai sistemi di suggerimento sui social network, agli assistenti vocali, fino a strumenti professionali di supporto decisionale. Tuttavia, la crescente dipendenza da servizi cloud solleva preoccupazioni significative: in molti casi, le informazioni elaborate — spesso sensibili o proprietarie — vengono inviate a server esterni, non sempre sotto il controllo diretto dell'utente o dell'organizzazione.

Per rispondere a queste criticità, Brainy Labs ha scelto di adottare un approccio alternativo, sviluppando un sistema completamente **locale**, in grado di eseguire tutte le fasi dell'elaborazione direttamente sul dispositivo dell'utente. Questa architettura consente di mantenere il pieno controllo sui dati, riducendo drasticamente i rischi di esposizione e migliorando la conformità a

normative europee come il *General Data Protection Regulation* (GDPR), che impone criteri rigorosi per la protezione e il trattamento delle informazioni personali.

Oltre all'attenzione per la sicurezza, Brainy Labs ha fatto una scelta strategica anche in termini di tecnologie adottate. Il prototipo è stato infatti sviluppato utilizzando strumenti ampiamente diffusi nel contesto lavorativo moderno, in modo da garantire non solo la qualità e l'attualità della soluzione proposta, ma anche un'esperienza formativa concreta e allineata con le esigenze reali del settore.

2.1.2 Scopo del Sistema

Il presente lavoro si inserisce all'interno di un percorso di **ricerca** e **sviluppo** volto a sperimentare una soluzione prototipale per l'analisi conversazionale di documenti PDF, con un focus particolare su:

- L'integrazione di **modelli linguistici avanzati** in applicazioni reali.
- L'**esecuzione** locale della catena di elaborazione, con benefici sia in termini di efficienza che di controllo sui dati.
- L'applicazione di tecniche di **retrieval semantico** (recupero di informazioni basato sul significato del testo, piuttosto che su semplici corrispondenze di parole chiave), in grado di restituire le parti più rilevanti del testo a partire da una domanda dell'utente.
- La valutazione dell'usabilità e dell'efficacia di un'interfaccia utente di tipo conversazionale, in grado di **semplificare l'interazione con dati complessi**.

Lo scopo del sistema sviluppato è quello di fornire una piattaforma web locale per l'interazione con modelli di linguaggio di grandi dimensioni (LLM),

in grado di comprendere e rispondere a domande relative sia a conversazioni libere che a documenti PDF caricati dall'utente.

Il sistema consente infatti all'utente di:

- avviare una nuova sessione di chat con un assistente AI;
- caricare un documento PDF, che viene associato in modo univoco alla chat corrente;
- riaprire qualsiasi chat passata, nella quale fosse stato caricato un documento, per rileggerne i messaggi o per scriverne di nuovi;
- porre domande relative al contenuto del documento, sfruttando l'indicizzazione semantica effettuata in fase di upload;
- ricevere risposte contestuali e pertinenti da parte del modello di intelligenza artificiale;
- scaricare in qualsiasi momento il documento associato alla chat;
- eliminare chat e documenti in modo semplice e sicuro, mantenendo il pieno controllo sui propri dati;
- cambiare, in qualsiasi momento, il modello di LLM dal quale ricevere risposte.

Il sistema è progettato per essere eseguito interamente in locale, senza la necessità di connessioni a server esterni o servizi cloud, con l'obiettivo di garantire:

- la massima tutela della **privacy** dell'utente;
- indipendenza da connessioni internet;
- elevata portabilità e facilità di distribuzione.

Nel complesso, l'obiettivo principale è quello di dimostrare l'efficacia di un approccio conversazionale all'interrogazione di documenti testuali, unendo la potenza dei modelli linguistici di ultima generazione a un'interfaccia accessibile e intuitiva, in un contesto pienamente controllato e replicabile.

2.1.3 Requisiti funzionali e non funzionali

Nella fase iniziale di analisi del progetto, è stato essenziale individuare e formalizzare in modo chiaro i requisiti che l'applicazione avrebbe dovuto soddisfare, al fine di guidare efficacemente l'intero processo di progettazione e sviluppo. I requisiti sono stati suddivisi in due categorie: **funzionali** e **non funzionali**, corrispondenti rispettivamente a ciò che il sistema deve fare e a come il sistema deve comportarsi.

Il progetto mira alla realizzazione di un'applicazione web interattiva e intelligente in grado di assistere l'utente nell'analisi di documenti PDF caricati direttamente da interfaccia. L'obiettivo è quello di fornire un'interazione in linguaggio naturale, simile a una conversazione con un assistente umano, ma mediata da un LLM eseguito in locale, senza necessità di connessione a servizi esterni o cloud.

Requisiti Funzionali

I requisiti funzionali definiscono le funzionalità essenziali del sistema, ovvero i comportamenti che deve essere in grado di eseguire per soddisfare le aspettative dell'utente finale:

- **Upload di documenti PDF**: l'applicazione deve consentire all'utente di caricare uno o più file PDF direttamente dal proprio dispositivo tramite un'interfaccia web semplice e intuitiva.
- **Analisi del contenuto testuale**: una volta caricato il documento, il sistema deve estrarne il contenuto testuale in maniera accurata, mantenendo la suddivisione per pagina per agevolare successivi riferimenti localizzati.
- **Interazione conversazionale**: il cuore del sistema è rappresentato da una chat in linguaggio naturale, attraverso la quale l'utente può porre domande, richiedere chiarimenti, riassunti o approfondimenti relativi al contenuto del documento.

- **Risposte intelligenti e pertinenti**: il sistema deve comprendere le richieste dell'utente e fornire risposte contestualizzate, coerenti e utili, utilizzando un LLM eseguito in locale e un meccanismo di *retrieval* semantico.
- **Gestione di più sessioni di chat**: l'utente deve poter iniziare nuove conversazioni in qualsiasi momento, con associazione a nuovi documenti o a nuove interazioni su documenti esistenti.
- **Storico delle conversazioni**: tutte le chat precedenti devono essere visualizzabili dalla homepage, permettendo la consultazione delle interazioni passate.
- **Associazione tra documenti e chat**: ogni chat è legata a uno specifico documento PDF; l'utente deve poterlo consultare e scaricare nuovamente dalla UI.
- **Persistenza dei dati**: tutti i messaggi devono essere salvati nel database locale per garantire continuità delle sessioni tra utilizzi successivi.
- **Elaborazione completamente in locale**: nessuna dipendenza da servizi esterni per l'elaborazione dei dati. Tutto deve essere eseguito localmente per garantire una maggiore privacy.

Requisiti Non Funzionali

I requisiti non funzionali descrivono aspetti qualitativi del sistema, spesso trasversali, che non riguardano una singola funzionalità ma l'intero comportamento dell'applicazione:

- **Performance e responsività bilanciati con esecuzione locale**: i tempi di risposta devono essere compatibili con un uso conversazionale fluido.
- **Interfaccia utente semplice ed efficace**: la UI deve essere moderna, accessibile e intuitiva per utenti di ogni livello.

- **Modularità e manutenibilità**: il sistema deve essere modulare, con componenti riutilizzabili e facilmente aggiornabili.

- **Uso di tecnologie moderne e open-source**: il progetto impiega tecnologie all'avanguardia, selezionate per efficienza, trasparenza e adesione ai principi della comunità open-source.

2.2 Tecnologie utilizzate

L'implementazione del sistema descritto nella presente tesi si basa sull'integrazione di una serie di tecnologie moderne, selezionate per soddisfare i requisiti di interattività, prestazioni, compatibilità e rispetto della privacy. In questa sezione vengono illustrate nel dettaglio le principali tecnologie adottate per il frontend, il backend, l'integrazione con modelli di AI locali e la gestione dei file utente.

2.2.1 Frontend

React

Il frontend dell'applicazione è realizzato con **React**, una popolare libreria JavaScript open source sviluppata da Meta (ex Facebook) per la costruzione di interfacce utente dinamiche e componenti interattivi. React adotta un paradigma dichiarativo e component-based, dove la UI è suddivisa in moduli riutilizzabili chiamati componenti. Ogni componente è in grado di gestire il proprio stato interno tramite strumenti come gli `hook` (ad esempio `useState`, `useEffect`, `useRef`), migliorando la modularità e la manutenibilità del codice [18].

Una caratteristica distintiva di React è l'uso del Virtual DOM, una rappresentazione in memoria dell'albero DOM reale, che consente di aggiornare

in modo efficiente solo le porzioni della pagina modificate, evitando costosi ricalcoli e migliorando le prestazioni complessive dell'interfaccia.

Next.js

Per la gestione delle rotte, l'organizzazione dei file e il supporto al rendering lato server, è stato adottato **Next.js**, un framework full-stack costruito sopra React. Next.js introduce funzionalità avanzate come il Server Side Rendering (SSR), il pre-rendering statico, il routing dinamico e la generazione automatica delle API. In particolare, la struttura dell'applicazione segue il modello introdotto delle "app directory" di Next.js, che permette di definire percorsi dinamici.
A differenza del tradizionale approccio basato su una directory `pages`, la nuova struttura adotta una logica più modulare e dichiarativa. Ogni sottocartella all'interno della `app/` directory rappresenta una rotta (route) dell'applicazione. Ad esempio, per gestire l'URL `/chat/abc123`, la directory sarà `app/chat/[chatId]/`, dove la porzione `[chatId]` indica un segmento dinamico, cioè un parametro che può cambiare a seconda della chat selezionata. Questo consente di mappare direttamente una rotta, permettendo così di caricare dinamicamente contenuti diversi in base al contesto.

L'utilizzo di Next.js permette anche un miglioramento delle performance e dell'indicizzazione da parte dei motori di ricerca (SEO), rendendolo una scelta ideale per applicazioni moderne e scalabili [19].

2.2.2 Backend

API REST

Le **API** (Application Programming Interfaces) sono interfacce che permettono a due componenti software di comunicare tra loro, rendendo disponibili alcune funzionalità o dati in modo controllato e standardizzato. In un'applicazione web moderna, le API rappresentano il canale attraverso cui il frontend può inviare richieste e ricevere risposte dal backend.

Nel caso specifico, si adottano API di tipo **REST** (REpresentational State Transfer), uno stile architetturale che sfrutta il protocollo HTTP per modellare risorse (es. messaggi, documenti, chat) come URL accessibili tramite operazioni standard (`GET`, `POST`, `PUT`, `DELETE`). Una caratteristica chiave delle API REST è la loro *statelessness*, ovvero l'assenza di stato lato server tra una richiesta e l'altra: ogni chiamata è autonoma e completa.

Nel contesto dell'applicazione, le API REST vengono implementate direttamente nella cartella `/api` di Next.js, sfruttando il supporto nativo del framework per definire *serverless function* che gestiscono richieste HTTP in modo modulare.

L'adozione delle API REST si è rivelata una scelta vantaggiosa in quanto:

- sono **stateless**, il che significa che ogni richiesta contiene tutte le informazioni necessarie per essere processata, semplificando la logica lato server;
- utilizzano convenzioni basate su **HTTP** standard (come `GET`, `POST`, `DELETE`), facilitando l'integrazione e il debug;
- sono **modulari** e facilmente estensibili, permettendo di aggiungere nuove funzionalità (es. `/api/chat`, `/api/file`, `/api/download`) in maniera scalabile e ordinata;
- sono supportate nativamente in Next.js, che consente di scrivere le route API come semplici file JavaScript/TypeScript, senza dover configurare un server separato.

SQLite

Per la persistenza dei dati viene utilizzato **SQLite**, un sistema di gestione di database relazionali leggero, self-contained e basato su file. A differenza di database client-server come MySQL o PostgreSQL, SQLite non richiede configurazione di rete né un server dedicato: tutti i dati sono memorizzati in un singolo file sul disco locale, rendendolo ideale per applicazioni embedded

o prototipi leggeri [20]. La comunicazione con il database avviene tramite la libreria `better-sqlite3`, che offre un'interfaccia sincrona ad alte prestazioni per ambienti Node.js.

File System (fs)

In combinazione con SQLite, si è fatto uso anche del modulo `fs` **(file system)** di Node.js, una libreria nativa che consente di interagire con il file system locale della macchina su cui gira l'applicazione. Questo modulo permette operazioni fondamentali come la lettura, scrittura e modifica di file da/su disco. Nel contesto di questo progetto, `fs` è impiegato per accedere direttamente ai file del database SQLite.

2.2.3 LLM e AI locale

Ollama e modelli open-source

Il nucleo intelligente del sistema è costituito da **Ollama**, un framework progettato per eseguire LLM in locale, senza necessità di inviare dati sensibili a server remoti. Ollama consente di caricare modelli AI open-source e interagire con essi tramite una semplice API HTTP locale, esposta di default all'indirizzo `localhost:11434` [21].

L'integrazione locale consente notevoli vantaggi in termini di privacy e controllo: le richieste dell'utente e i documenti caricati non lasciano mai il sistema locale, riducendo il rischio di violazioni o perdite di dati. Tra i modelli disponibili vi sono **Mistral**, **DeepSeek**, **Phi** e altri modelli efficienti sviluppati per essere eseguiti anche su hardware consumer (ossia hardware di pc destinati ad uso personale).

Particolare attenzione è rivolta al modello `nomic-embed-text`, utilizzato esclusivamente per la generazione di **embedding** (e dunque non a scopo conversazionale, a differenza degli altri), ovvero rappresentazioni numeriche dei testi necessarie per il recupero semantico dei contenuti.

Grazie ad Ollama è possibile interagire con questi modelli localmente, anche su leggeri dispositivi personali, senza dover ricorrere a servizi cloud esterni.

2.2.4 Strumenti ausiliari

PDF.js

Per l'elaborazione dei documenti PDF caricati dagli utenti viene utilizzata la libreria **PDF.js**, sviluppata da Mozilla. Essa consente di interpretare i file PDF e convertirli in una rappresentazione strutturata navigabile e analizzabile tramite JavaScript, fondamentale per consentire la successiva indicizzazione e interrogazione da parte del motore NLP. Inoltre, è indispensabile poiché permette l'utilizzo di apposite funzioni quali `getDocument`, `getPage`, `getTextContent` per leggere i PDF [22].

Busboy

Il caricamento dei file all'interno delle API Next.js è gestito dalla libreria **Busboy**, che analizza richieste `multipart/form-data` e consente di salvare file binari (es. PDF) in modo efficiente all'interno dell'infrastruttura backend.

Bootstrap

Per la costruzione dell'interfaccia grafica è stato adottato **Bootstrap 5**, un framework CSS open source che fornisce componenti UI pronti all'uso, uno stile responsivo e una griglia flessibile per l'organizzazione dei contenuti. Bootstrap è completato dall'utilizzo di **Bootstrap Icons**, una libreria di icone vettoriali leggere e integrate nativamente nel framework [23].

2.3 Architettura generale del sistema

2.3.1 Panoramica delle componenti

Dopo aver illustrato nel dettaglio le tecnologie adottate, in questa sezione si propone una visione d'insieme dell'architettura funzionale del sistema, evidenziando il ruolo e le responsabilità di ciascuna componente nella catena di elaborazione.

Il sistema è strutturato secondo un'architettura client/server ed è composto da quattro blocchi principali:

- **Frontend web**, realizzato in React e gestito tramite il framework Next.js, ha il compito di fornire all'utente un'interfaccia conversazionale intuitiva e interattiva, permettendo l'invio di messaggi e il caricamento di documenti PDF.

- **Backend applicativo**, implementato come API REST direttamente nel contesto Next.js, funge da ponte tra il frontend e le logiche di elaborazione. Gestisce le richieste dell'utente, coordina l'interazione con l'AI locale, salva le informazioni nel database e gestisce i file PDF caricati.

- **Motore AI locale**, basato su Ollama, consente l'esecuzione di modelli linguistici di grandi dimensioni (LLM) direttamente in locale. Due categorie di modelli vengono utilizzate: uno per la generazione conversazionale (es. DeepSeek, Mistral), l'altro per il calcolo degli embedding testuali (es. nomic-embed-text), impiegati nel processo di retrieval semantico.

- **Database SQLite**, responsabile della persistenza dei dati locali, memorizza i messaggi delle conversazioni, le informazioni sui documenti e le relazioni tra essi. Grazie alla sua natura file-based, si adatta bene a un contesto di esecuzione locale, mantenendo leggerezza e semplicità di gestione.

Queste componenti interagiscono tra loro in modo sinergico per offrire un'esperienza utente fluida e coerente. L'interfaccia utente funge da punto di accesso centrale, mentre il backend si occupa del coordinamento logico. L'AI locale consente l'elaborazione semantica dei contenuti, mantenendo i dati all'interno del sistema, e il database garantisce la tracciabilità delle interazioni con persistenza selettiva (le chat sono salvate solo in presenza di un documento caricato).

La figura 3.5 illustra graficamente questa suddivisione in componenti, evidenziandone i flussi principali.

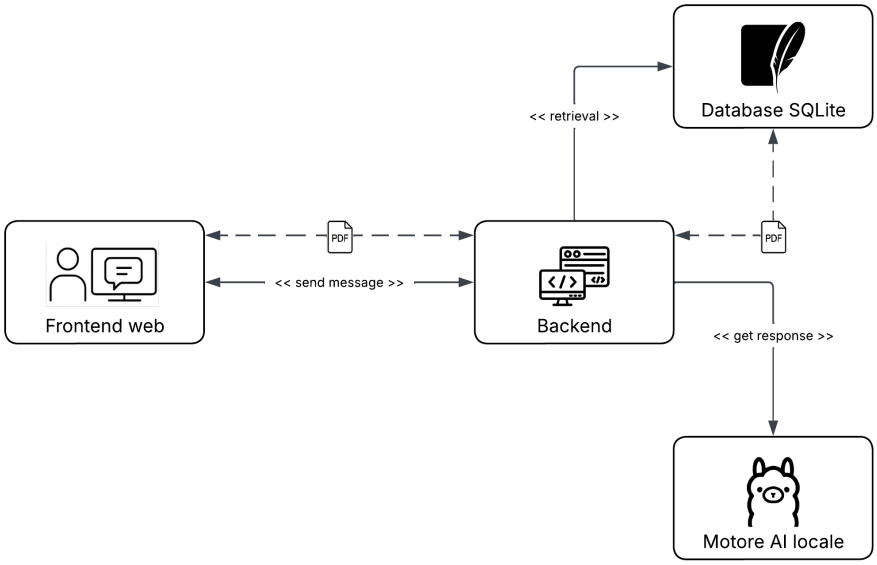

Figura 2.2: Architettura generale del sistema

2.3.2 Flusso e Casi d'Uso

Il funzionamento dell'applicazione segue un flusso ben definito che parte dalla creazione di una nuova chat e culmina nell'interazione conversazionale tra utente e modello AI, intensificata dall'eventuale caricamento di un documento PDF.

Il caso d'uso principale prevede che l'utente interagisca con il sistema attraverso un'interfaccia web, caricando un documento e ponendo domande ad esso relative. I passaggi fondamentali possono essere riassunti come segue:

1. **Avvio di una nuova sessione di chat:** l'utente seleziona il pulsante *"+ Nuova Chat"*, che genera sul backend un nuovo identificativo di chat.

2. **Caricamento documento:** all'interno della nuova chat, l'utente, oltre a poter conversare in maniera generica con il modello LLM scelto, ha la possibilità di caricare un documento PDF. In quest'ultimo caso, il sistema ne estrae il testo, lo divide in chunk e ne calcola gli embedding tramite l'apposito modello locale (`nomic-embed-text`), memorizzando il tutto nel database.

3. **Inserimento domanda:** l'utente scrive una domanda nel campo di input e la invia.

4. **Recupero semantico:** il sistema confronta la domanda con gli embedding delle porzioni di testo presenti nel documento, selezionando i chunk più rilevanti.

5. **Risposta del modello AI:** la richiesta viene inoltrata al modello AI locale (es. LLaMA, Mistral, DeepSeek), insieme al prompt e ai chunk più pertinenti. Il modello restituisce una risposta contestuale.

6. **Visualizzazione e persistenza:** la risposta viene mostrata all'utente e salvata nel database (assieme al messaggio utente) solo se nella chat è presente un documento. In assenza di un documento, la chat non viene registrata nel sistema.

Oltre al flusso principale, si possono individuare ulteriori casi d'uso accessori, quali:

- **Eliminazione di una chat esistente** dalla home page tramite il tasto "X", con conferma da parte dell'utente.

- **Selezione del modello AI** da un menu a tendina, cliccabile in qualsiasi momento (anche nel bel mezzo della conversazione), in alto a sinistra nella navbar, che mostra la lista (filtrata opportunamente per non includere modelli non conversazionali) dei modelli LLM disponibili.

- **Download del documento** possibile tramite un apposito bottone che compare al posto di quello per l'upload dopo l'avvenuto caricamento.

Questo approccio modulare consente all'applicazione di adattarsi sia a un uso temporaneo (chat generiche, non persistenti, senza documento) sia a un uso più strutturato, basato su sessioni salvate e associate a uno specifico file PDF.

2.3.3 Scelte architetturali: client/server - locale

Il sistema progettato adotta un'architettura di tipo **client/server**, dove l'interfaccia utente (frontend) interagisce con un backend centralizzato che gestisce la logica applicativa, l'elaborazione dei file e la persistenza dei dati. Tuttavia, a differenza dei classici sistemi web distribuiti su infrastrutture cloud, l'intera applicazione è concepita per l'esecuzione in **ambiente locale**, mantenendo così un maggiore controllo sui dati e sulle risorse computazionali.

Il frontend è sviluppato in **Next.js**, un framework React-based che permette la generazione di interfacce dinamiche e componentizzate, in grado di comunicare con il backend tramite chiamate a **API REST** interne. Queste API sono implementate direttamente nel progetto, nella cartella `/api`, e vengono eseguite lato server all'interno del medesimo ambiente Next.js, evitando la necessità di un'infrastruttura separata.

La scelta di eseguire il sistema **in locale** è stata dettata da vari fattori progettuali:

- **Privacy**: i documenti caricati dall'utente non vengono mai inviati su server esterni. Tutte le operazioni, inclusa l'elaborazione semantica, avvengono esclusivamente sul dispositivo locale.

- **Semplicità di distribuzione**: il sistema è facilmente installabile su qualsiasi macchina senza configurazioni cloud complesse.

- **Sperimentazione prototipale**: l'ambiente locale consente un controllo maggiore durante la fase di sviluppo e testing, particolarmente utile in un contesto di ricerca;

- **Indipendenza dalla connessione Internet**: l'intero sistema può funzionare offline, senza dipendere da servizi esterni o infrastrutture cloud.

Il motore di intelligenza artificiale si basa su **Ollama**, un runtime che permette di eseguire **modelli linguistici open-source** (LLM) direttamente sulla macchina dell'utente. Questo approccio elimina la necessità di un servizio AI in cloud (come OpenAI o Google Cloud), mantenendo i dati sempre sotto il controllo dell'utente e rendendo il sistema autosufficiente.

In conclusione, l'architettura adottata rappresenta un compromesso efficace tra **modularità client/server** e **esecuzione locale**, ottimizzando privacy, performance e semplicità di utilizzo.

2.3.4 Progettazione Database

Il sistema utilizza un database relazionale **SQLite**, scelto per la sua leggerezza, portabilità e semplicità d'integrazione in ambienti locali. La struttura dati è stata progettata per gestire efficacemente le entità coinvolte nel processo conversazionale con documenti PDF, e supportare le funzionalità di persistenza delle chat, dei file utente e dei relativi contenuti elaborati.

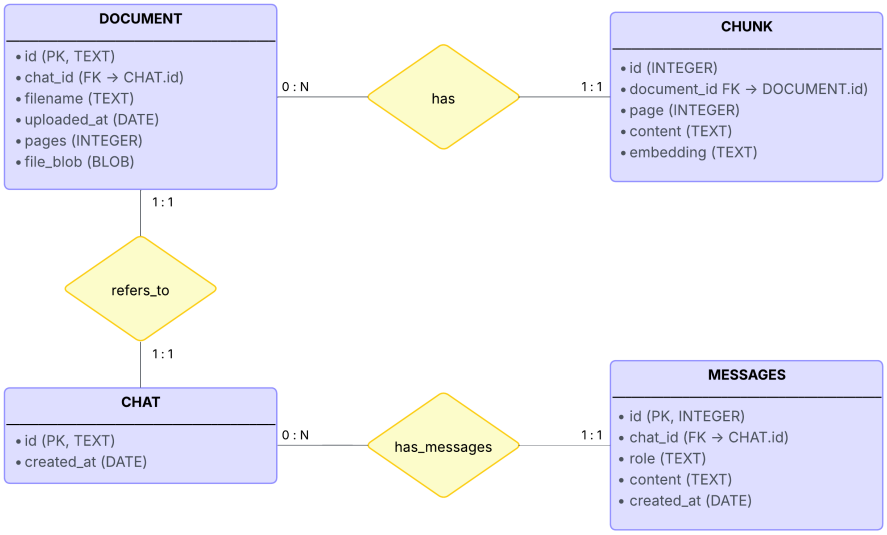

Figura 2.3: Struttura concettuale del database

Come illustrato in Figura 2.3, il database è articolato attorno a quattro entità principali:

- **CHAT**: rappresenta ogni conversazione avviata dall'utente. Ogni chat viene memorizzata solo se contiene un documento associato, seguendo la logica progettuale che considera come valide solo le interazioni documentali.
- **DOCUMENT**: contiene i metadati e il contenuto binario dei file PDF caricati. Ogni documento è associato a una singola chat (`chat_id`), instaurando una relazione 1:1. Sono memorizzati il nome del file, il numero di pagine, la data di caricamento e il file stesso in formato BLOB.
- **CHUNK**: ogni documento viene suddiviso in *spezzoni testuali* (chunk) contenenti anche l'embedding semantico del contenuto. Ogni chunk è associato a un'unica pagina e a un solo documento. Questa struttura consente un efficace retrieval semantico delle informazioni durante la fase di risposta dell'AI.

- **MESSAGES**: raccoglie tutti i messaggi scambiati nella conversazione, distinguendo tra messaggi dell'utente (`role = 'user'`) e dell'AI (`role = 'bot'`). Ogni messaggio è legato a una specifica chat tramite chiave esterna.

Dal punto di vista relazionale, si osservano i seguenti legami:

- Una **chat** può contenere da zero a molti messaggi (relazione 0:N).
- Ogni **documento** è associato esattamente a una chat (relazione 1:1) e viceversa.
- Ogni **documento** può essere suddiviso in più chunk (relazione 0:N).

La progettazione tiene conto anche dell'estensibilità futura: la struttura consente, ad esempio, l'aggiunta di nuovi metadati, la gestione di altri formati documentali o il collegamento a più documenti per singola sessione, se necessario.

Capitolo 3

Implementazione

3.1 Frontend

Il frontend dell'applicazione è stato sviluppato con **React** e **Next.js**, sfruttando la struttura a componenti offerta dal framework per organizzare le funzionalità in moduli riutilizzabili, mantenibili e reattivi.

3.1.1 Interfaccia Utente

L'interfaccia utente è progettata per essere **semplice**, **intuitiva** e completamente **responsiva**. Il layout è strutturato per adattarsi a dispositivi di diverse dimensioni, mantenendo una chiara gerarchia visiva.

Il punto di ingresso principale per l'utente è rappresentato dalla **homepage**, che mostra una griglia di "card" corrispondenti alle sessioni di chat precedentemente salvate, ciascuna associata a un documento PDF caricato.

Ogni card riporta il nome del file, un'icona identificativa e un bottone per l'eventuale eliminazione della chat, protetto da una richiesta di conferma per evitare cancellazioni accidentali. È inoltre presente una card speciale con simbolo "+" che consente di avviare una nuova sessione di chat.

La navigazione verso una chat esistente avviene invece tramite click sulla rispettiva card, che reindirizza alla pagina dinamica `/chat/[chatId]`. Al-

l'interno di queste pagine di chat si sviluppa l'interfaccia conversazionale vera e propria.

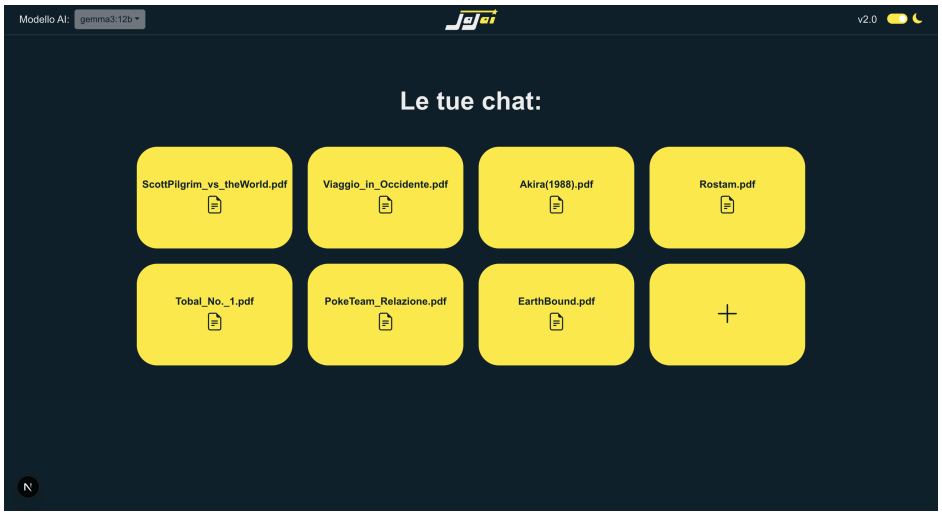

Figura 3.1: Schermata della homepage

Quando l'utente atterra sulla pagina di una nuova chat, e non ha ancora iniziato la conversazione, la barra di input testuale, assieme ai pulsanti di invio e di allegazione file che la affiancano, viene automaticamente centrata verticalmente per dare maggiore enfasi all'inizio dell'interazione; inoltre appare soprastante un messaggio di incoraggiamento a scrivere. Al contrario, in presenza di messaggi, la barra di input (insieme ai due relativi bottoni) rimane fissata in basso, come in una classica finestra di chat.

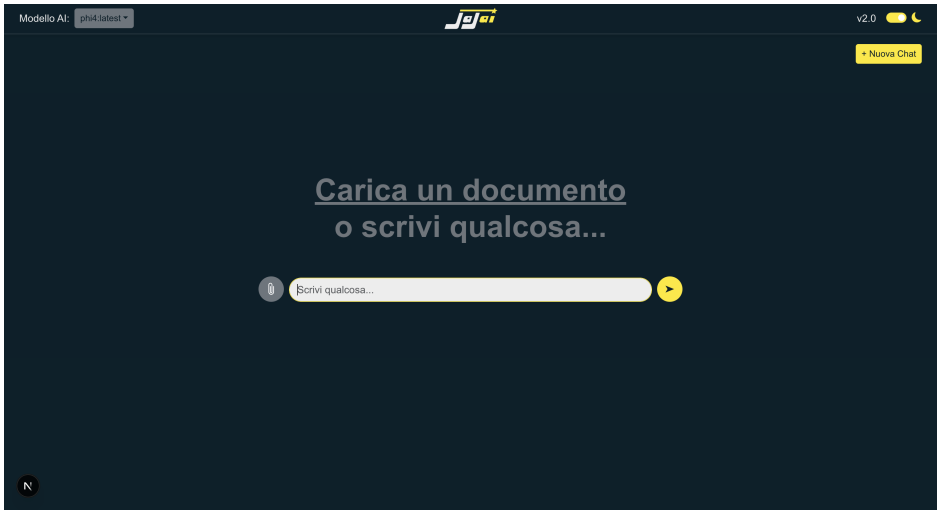

Figura 3.2: Come appare una nuova chat vuota

L'invio dei messaggi avviene sia tramite l'apposito bottone a destra della casella di testo, sia tramite pressione del tasto `Enter` su tastiera, con supporto al ritorno a capo tramite `Shift + Enter`. Dopo l'invio, il messaggio dell'utente viene visualizzato e immediatamente seguito da un indicatore di caricamento (un cursore lampeggiante "|") per segnalare che l'AI sta generando una risposta. Durante la generazione della risposta da parte dell'AI, il bottone di invio viene disabilitato per impedire all'utente di inviare un secondo messaggio prima di aver ricevuto la risposta al primo.

Una volta ricevuta la risposta, l'interfaccia effettua automaticamente lo **scroll verso il basso**, scorrendo quindi, fino in fondo, i messaggi fermandosi al più recente. Tutti i messaggi sono stilizzati con colori e dimensioni diverse per distinguere chiaramente i ruoli dell'utente e dell'AI.

Particolare attenzione è stata posta anche nella **gestione dei file PDF**: se non è ancora stato caricato alcun documento, viene mostrato, affianco alla barra di input, un bottone per allegare file. Durante l'elavborazione del documento da parte del backend, viene mostrato un messaggio temporaneo, analogo a quello che appare durante la generazione di una risposta da parte dell'AI, per rappresentare il caricamento. Durante questa fase, il

bottone per allegare file e il bottone d'invio vengono disabilitati. Una volta andata a buon fine l'elaborazione del PDF da parte del backend, questo viene associato alla chat e il bottone per allegare file si trasforma automaticamente in un bottone, con nuova icona, per il download del file caricato (in caso l'utente avesse, in futuro, bisogno di riscaricarlo), evitando così ambiguità e semplificando l'esperienza utente rendendola accessibile a chiunque.

Figura 3.3: Schermata di una chat con documento caricato

Un elemento centrale dell'esperienza utente è la **navbar** superiore, visibile in ogni pagina. Al suo interno sono stati integrati due componenti interattivi:

- Un **menù a tendina per selezione del modello AI**, che consente all'utente di scegliere quale modello linguistico utilizzare tra quelli disponibili in locale (es. `deepseek`, `gemma`, `phi`, ecc.). Il menù è stato programmato in modo da mostrare soltanto i modelli conversazionali, escludendo altre tipologie quali modelli per l'embedding. La selezione avviene dinamicamente, e viene mantenuta nel contesto dell'applicazione fino a nuova modifica. In questo modo, l'utente può sperimentare rapidamente l'efficacia di modelli diversi su uno stesso documento.

- Uno **switch per l'attivazione del tema scuro/chiaro**, implementato per migliorare l'accessibilità e l'ergonomia visiva. Lo switch sfrutta le preferenze utente e il sistema CSS nativo `prefers-color-scheme`, adattando i colori dell'interfaccia alle condizioni ambientali e alle preferenze di chi utilizza il sistema.

Entrambi questi componenti sono stati realizzati in modo minimale e integrati armoniosamente nell'estetica generale dell'applicazione, mantenendo l'interfaccia pulita e concentrata sul contenuto conversazionale.

Figura 3.4: Schermata che mostra la selezione del modello di AI tramite l'apposito menu a tendina della navbar

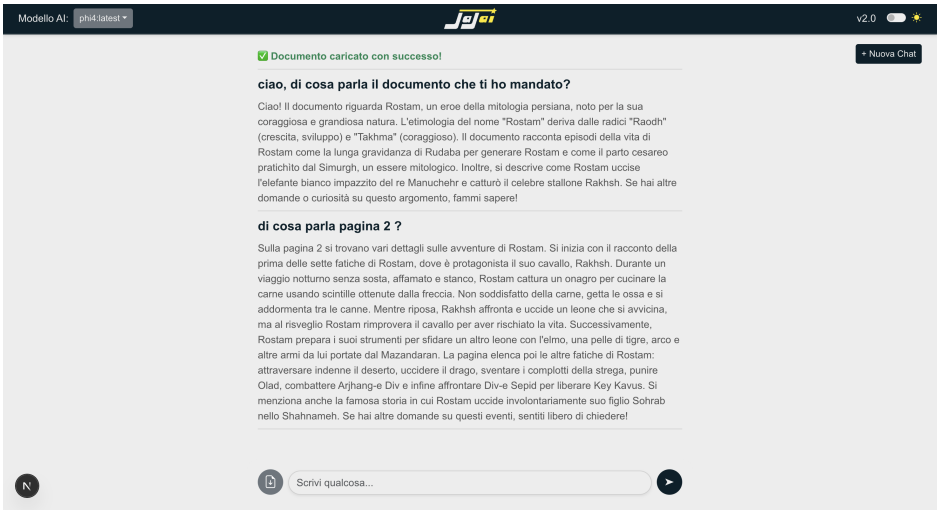

Figura 3.5: Schermata che mostra il tema-chiaro, attivato tramite l'apposito bottone a switch dalla navbar

3.1.2 Gestione delle Chat e Sessioni

Ogni nuova interazione con il sistema può essere avviata premendo il pulsante "`+ Nuova Chat`" (o anche la card "`+`" dalla homepage), che genera un nuovo identificatore univoco e reindirizza l'utente a una nuova pagina di chat. Questo comportamento è fondamentale per isolare le conversazioni e mantenere la coerenza tra i messaggi e i documenti allegati.

I messaggi della conversazione corrente, così come il modello di AI attualmente in uso (selezionato dalla tendina), vengono temporaneamente memorizzati in **sessionStorage**, in modo da preservare lo stato anche in caso di refresh della pagina. In parallelo, se è stato allegato un documento, la chat viene anche salvata nel database tramite chiamate API, insieme a tutti i messaggi generati.

In particolare:

- se l'utente interagisce senza allegare un file, la sessione non viene salvata nel database;

- se invece viene allegato un documento, la sessione viene persistita, e ogni messaggio successivo viene sincronizzato con il backend.

Questo meccanismo ibrido consente di ridurre la quantità di dati salvati inutilmente e ottimizzare le risorse, pur garantendo la possibilità di riprendere le chat documentate in qualsiasi momento.

Inoltre, dalla schermata principale (homepage), l'utente ha la possibilità di **eliminare una chat salvata** tramite un'apposita icona "X" presente su ogni card. Quando questa viene cliccata, viene mostrato un messaggio di conferma per evitare cancellazioni accidentali. Se l'utente conferma, la chat e il documento associato vengono rimossi dal database in modo atomico, grazie a una route API dedicata. Questo garantisce che nessuna informazione residua venga mantenuta dopo l'eliminazione, contribuendo a una gestione ordinata e trasparente delle sessioni.

3.2 Backend

3.2.1 Upload, Parsing e Indicizzazione dei Documenti

Una delle funzionalità centrali del sistema consiste nella possibilità per l'utente di allegare un documento PDF all'interno di una sessione di chat. Questo documento viene successivamente analizzato ed indicizzato al fine di renderlo interrogabile mediante linguaggio naturale.

Caricamento e gestione lato client

Dal punto di vista dell'interfaccia, l'utente può caricare un file PDF tramite un pulsante visibile nella barra inferiore della chat. Il file viene gestito da un componente React che sfrutta un input HTML di tipo `file`, il quale, una volta selezionato il documento, lo invia al backend tramite una chiamata HTTP POST verso l'endpoint `/api/file`.

Durante questa fase viene mostrato un messaggio temporaneo nella chat (un cursore lampeggiante "`|`") e il pulsante di upload viene disabilitato, fornendo un riscontro visivo immediato all'utente.

Parsing del contenuto con PDF.js

Il backend riceve il file utilizzando il modulo `Busboy`, che consente di gestire file multipart in modo efficiente. Una volta completato l'upload, il documento viene analizzato tramite `PDF.js`, una libreria open-source per l'estrazione del testo dai file PDF.

Tale estrazione restituisce una concatenazione ordinata del contenuto testuale delle varie pagine, permettendo al sistema di disporre di una rappresentazione lineare del documento, utilizzabile nei passaggi successivi.

Suddivisione in chunks

Una volta estratto il contenuto testuale del PDF, il sistema procede alla sua suddivisione in **frammenti** (*chunks*), ovvero blocchi di testo di dimensione controllata (generalmente di 500 caratteri). Per evitare che parti importanti del contenuto vengano tagliate bruscamente, viene applicata una **sovrapposizione** tra i blocchi, così da conservare parte del contesto testuale da un chunk al successivo.

Tale tecnica consente un'analisi più granulare del documento e migliora la capacità del sistema di individuare le porzioni di testo più rilevanti durante la fase di risposta, riducendo la possibilità che informazioni significative vengano trascurate a causa della segmentazione.

Inoltre, ogni chunk viene arricchito da metadati relativi alla **pagina di origine** e alla posizione nel documento, rendendo possibile un successivo tracciamento preciso delle informazioni.

Questa fase costituisce un prerequisito essenziale per la generazione degli embedding e per il recupero semantico, che vengono trattati nella sezione seguente.

Calcolo degli embeddings

Ogni frammento viene trasformato in un vettore numerico (embedding) tramite il modello `nomic-embed-text`, eseguito in locale grazie al framework `Ollama`. Gli embeddings sono rappresentazioni densi del significato semantico del testo, utili per misurare similarità tra porzioni di testo e query dell'utente.

Il sistema interroga Ollama inviando una richiesta HTTP contenente i chunk testuali, e riceve in risposta un array di vettori multidimensionali.

Salvataggio nel database

Una volta calcolati, i chunks embeddizzati vengono memorizzati nel database SQLite insieme ai metadati associati:

- `document_id` (riferimento al documento di origine),
- `text` (contenuto del chunk),
- `embedding` (rappresentazione numerica codificata).

Queste informazioni vengono poi utilizzate nelle fasi successive per eseguire operazioni di retrieval semantico mirato, basato su similarità vettoriale.

3.2.2 Retrieval Semantico con Cosine Similarity

Una delle funzionalità chiave del sistema consiste nella capacità di rispondere a domande dell'utente facendo riferimento al contenuto dei documenti PDF caricati. Per ottenere questo comportamento, l'applicazione implementa un meccanismo di **retrieval semantico**, ovvero un sistema di recupero delle informazioni che non si basa su semplici parole chiave, ma sul *significato* delle frasi.

Il funzionamento si articola in tre fasi principali:

1. **Embedding della domanda:** al momento della richiesta da parte dell'utente, la frase inserita viene trasformata in un vettore numerico (embedding) che ne cattura il significato semantico. Questo processo

viene svolto localmente tramite il modello `nomic-embed-text`, eseguito all'interno del runtime `Ollama`.

2. **Calcolo della similarità:** una volta ottenuto l'embedding della domanda, esso viene confrontato con tutti gli embedding dei chunks memorizzati nel database (ottenuti durante la fase di indicizzazione del documento).

   Per misurare il grado di somiglianza tra due vettori numerici, viene utilizzata la **cosine similarity**, una metrica che calcola il *coseno dell'angolo* tra due vettori nello spazio n-dimensionale. La cosine similarity valuta l'orientamento dei vettori, cioè quanto sono "puntati" nella stessa direzione e questo la rende particolarmente adatta al confronto di rappresentazioni semantiche, poiché in questi casi non è importante la lunghezza assoluta dei vettori, ma la loro direzione nello spazio: due testi con significato simile generano vettori che puntano nella stessa direzione, anche se di lunghezza diversa.

   Matematicamente, il valore della similarità coseno tra due vettori $\vec{a}$ e $\vec{b}$ si ottiene dalla formula [14] :

$$\text{similarity}(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \cdot \|\vec{b}\|}$$

   Questa metrica restituisce un valore compreso tra -1 e 1:

   - 1 indica che i due vettori sono perfettamente allineati (massima affinità),
   - 0 indica che sono ortogonali (nessuna correlazione),
   - -1 indica che puntano in direzioni opposte (massima dissimilarità).

   La cosine similarity è ampiamente utilizzata in ambito di *information retrieval* e *natural language processing* per confrontare vettori di embedding, poiché si dimostra efficace nell'identificare similarità

semantiche tra testi indipendentemente dalla lunghezza assoluta dei vettori.

3. **Recupero dei chunks più rilevanti:** il sistema seleziona i frammenti (chunks) con il punteggio di similarità più alto e li utilizza per costruire il *contesto* da fornire al modello linguistico nella fase successiva di generazione della risposta.

Questo approccio permette al modello AI di rispondere tenendo conto delle porzioni più pertinenti del documento, anche nel caso in cui la domanda non contenga esattamente le stesse parole del testo originale. Si tratta quindi di una forma avanzata di **recupero basato sul significato**, che migliora l'efficacia dell'interazione e riduce il rischio di risposte generiche o fuori contesto.

Dal punto di vista implementativo, l'intero processo avviene localmente:

- gli embedding dei chunks e delle domande sono calcolati con modelli AI open-source;
- la similarità è calcolata in JavaScript tramite una funzione personalizzata basata sulla formula del coseno;
- i dati necessari sono recuperati dal database `SQLite` e ordinati in base alla similarità.

In questo modo, si ottiene un sistema autonomo, performante e rispettoso della privacy, in cui l'intero ciclo di retrieval semantico è confinato al dispositivo dell'utente.

Esempio di Implementazione

La funzione `cosineSimilarity`, responsabile del calcolo del punteggio tra due vettori, è definita come segue:

Listing 3.1: Funzione per il calcolo della cosine similarity

```
function cosineSimilarity(a, b) {
  const dotProduct = a.reduce((sum, val, i) => sum + val
      * b[i], 0);
  const normA = Math.sqrt(a.reduce((sum, val) => sum +
     val * val, 0));
  const normB = Math.sqrt(b.reduce((sum, val) => sum +
     val * val, 0));
  return dotProduct / (normA * normB);
}
```

La funzione `cosineSimilarity` implementa il calcolo della similarità coseno tra due vettori numerici, rappresentati dagli array `a` e `b`. Essa procede in tre passaggi fondamentali:

1. **Prodotto scalare:** calcola la somma dei prodotti elemento per elemento tra i due vettori, ossia $\sum_i a_i \cdot b_i$. Questo valore rappresenta quanto i due vettori "puntano" nella stessa direzione.

2. **Norma euclidea:** determina la lunghezza (o modulo) di ciascun vettore utilizzando la radice quadrata della somma dei quadrati dei suoi elementi, ovvero $\|\vec{a}\| = \sqrt{\sum_i a_i^2}$.

3. **Divisione normalizzata:** divide il prodotto scalare per il prodotto delle due norme, ottenendo così un valore compreso tra -1 e 1. Valori vicini a 1 indicano una forte somiglianza direzionale, mentre valori vicini a 0 indicano dissimilarità.

L'uso della similarità coseno consente quindi di stimare l'affinità semantica tra una domanda e le porzioni di testo indicizzate, migliorando il recupero delle informazioni più pertinenti.

Il seguente frammento mostra invece l'algoritmo principale utilizzato per il recupero dei chunks più rilevanti a partire dalla domanda dell'utente:

Listing 3.2: Retrieval semantico via cosine similarity

```
const scoredChunks = chunks.map(chunk => {
  const chunkEmbedding = JSON.parse(chunk.embedding);
  const score = cosineSimilarity(queryEmbedding,
     chunkEmbedding);
  return { ...chunk, score };
});

return scoredChunks
  .sort((a, b) => b.score - a.score)
  .slice(0, topK);
```

Qui il sistema:

- confronta l'embedding della domanda con ciascun embedding dei chunk;
- calcola un punteggio di similarità;
- ordina i risultati;
- seleziona i **top-k** frammenti semanticamente più vicini alla query dell'utente.

I chunk selezionati vengono successivamente concatenati e forniti al modello linguistico come *contesto*, migliorando la qualità e la pertinenza delle risposte generate. Questa procedura consente al sistema di generare risposte contestuali e precise, anche a partire da richieste formulate in modo semantico diverso dal testo originale.

3.2.3 Gestione delle richieste per pagina specifica

Un caso particolare di retrieval riguarda le domande dell'utente che contengono riferimenti espliciti a una **pagina specifica** del documento PDF, come ad esempio: *"Cosa dice pagina 5?"* oppure *"Spiegami la pagina 2"*. Per affrontare questa esigenza, sono state sperimentate tre soluzioni distinte:

Strategia 1 – Inserire il numero di pagina nei chunks

In questa ipotesi, il numero di pagina veniva inserito direttamente all'interno del contenuto testuale di ciascun chunk, come nel seguente esempio:

```
const pageChunks = chunkText(pageText, 500, 100).map(
   chunk => ({
            id: chunkId++,
            page: pageNum,
            content: `[pagina ${pageNum}] ${chunk}`,
          }));
```

L'idea era che l'embedding generato contenesse informazioni sul numero di pagina. Tuttavia, i modelli non sembrano interpretare efficacemente tali riferimenti numerici, e i risultati ottenuti erano spesso poco coerenti.

Strategia 2 – Parsing testuale della domanda

Questa strategia, che si è dimostrata la più efficace, prevede il parsing della domanda con una semplice espressione regolare. Se viene rilevata una struttura del tipo `pagina N`, la funzione bypassa completamente il retrieval semantico, restituendo direttamente i chunk della pagina:

```
const match = userQuery.match(/pagina\s+(\d+)/i);
if (match) {
  const pageNumber = parseInt(match[1]);
  return retrieveChunksByPage(documentId, pageNumber);
}
```

Questo metodo, per quanto semplice, ha fornito risultati affidabili e con un'ottima reattività.

Strategia 3 – Classificazione AI della domanda

Una soluzione più ambiziosa ha previsto di chiedere al modello stesso se la domanda facesse riferimento a una pagina, inviando un messaggio strutturato prima del retrieval:

```
{
  role: "system",
  content: `If the question refers to a page, reply ONLY
      with its number. Else reply "semantic".`
}
```

Il modello rispondeva con un numero o con la parola `semantic`, e il sistema agiva di conseguenza. Questo approccio, oltre a non essere risultato pienamente efficace nella classificazione, ha avuto un impatto negativo sui tempi di risposta, rallentando notevolmente l'interazione utente.

Considerazioni finali

Le tre strategie sono state confrontate in base a **efficacia** e **performance**. Alla fine, è stata adottata la strategia 2, rivelatasi la più **semplice, stabile ed efficiente**. La combinazione di parsing regolare e retrieval diretto si è dimostrata sufficiente per coprire tutti i casi d'uso realistici, senza complicazioni né impatti sulla UX.

3.2.4 Integrazione con LLM locali tramite Ollama

Il sistema utilizza **Ollama** per eseguire modelli linguistici di grandi dimensioni (*Large Language Models* - LLM) in locale, direttamente sulla macchina dell'utente. Questo approccio elimina la necessità di inviare dati a ser-

ver esterni, garantendo pieno controllo sulla privacy e inoltre rende il sistema completamente **autosufficiente** anche in assenza di connessione internet..

Selezione e gestione dei modelli

All'avvio dell'applicazione, il frontend interroga l'endpoint `/api/models` per ottenere la lista dei modelli installati e disponibili tramite Ollama. Il primo modello disponibile (tra quelli scaricati in locale dall'utente) viene selezionato automaticamente, ma l'utente può modificarlo in qualsiasi momento tramite l'apposito selettore nella navbar. Il modello scelto viene salvato nel `sessionStorage` del browser, assicurando persistenza tra ricaricamenti di pagina. La gestione di questo stato è centralizzata nel file `ModelContext.js`, tramite React Context API.

Invio della richiesta e struttura del prompt

Quando l'utente invia un messaggio, il componente `ChatClient`, dopo aver salvato il messaggio dell'utente del database (solo se presente un documento associato alla chat), costruisce un array di messaggi comprensivo della cronologia (importante per fornire all'AI il contesto della conversazione corrente) e lo invia all'endpoint `/api/chat`, specificando anche:

- il modello attualmente selezionato;
- l'ID del documento, se presente;
- l'ID della sessione di chat.

Nel backend, la route `/api/chat` costruisce un prompt da fornire al modello. Se è presente un documento, viene prima invocato il meccanismo di *retrieval semantico* per estrarre i chunk più rilevanti, che vengono anteposti al messaggio dell'utente per arricchirne il contesto. L'intera sequenza viene poi inviata in POST all'API locale di Ollama, raggiungibile all'indirizzo `http://localhost:11434/api/chat`.

Recupero dello stato della chat al caricamento

Quando un utente accede a una sessione salvata, ad esempio tramite la rotta `/chat/[chatId]`, Next.js esegue una funzione server-side definita in `page.js`. Questo modulo interroga il database per recuperare:

- tutti i messaggi associati alla chat specifica (`chatId`);
- il documento PDF allegato, se presente.

Entrambi i dati vengono quindi passati come proprietà iniziali al componente `ChatClient`, che li utilizza per ricostruire correttamente l'interfaccia dell'utente e abilitare il download del file caricato. Questo meccanismo consente di ripristinare facilmente lo stato delle conversazioni passate, senza necessità di ulteriore sincronizzazione client-server.

Parsing e formattazione della risposta

Una volta ricevuta la risposta dal modello, questa viene prima salvata nel database (solo se la chat è associata ad un documento) e poi sottoposta a un processo di **pulizia e formattazione**. Tale passaggio è fondamentale per garantire un'esperienza utente chiara e leggibile: molte AI generano il testo secondo convenzioni Markdown (es. `**grassetto**`, `*corsivo*`, `<tag>`), che se non interpretate correttamente possono risultare confuse o antiestetiche.

Il sistema rimuove quindi simboli superflui come asterischi, backtick e sequenze speciali, e inserisce spaziature e andate a capo per migliorare la leggibilità. Ad esempio, per alcuni modelli come **DeepSeek**, viene applicato un parsing dedicato per eliminare porzioni di testo racchiuse tra tag non rilevanti (ad esempio `<think>...</think>`), che rappresentano istruzioni interne del modello ma non sono destinate all'utente.
La risposta formattata viene poi quindi visualizzata nella chat (con eventuali stili per intestazioni, grassetti, errori ecc.).

Questa architettura consente di ottenere risposte contestualizzate e coerenti, con la possibilità di modificare dinamicamente il modello AI, tutto mantenendo una gestione completamente locale dell'inferenza.

3.2.5 API Backend

L'architettura del sistema prevede una serie di API REST che mediano la comunicazione tra frontend e backend. Queste API sono implementate come `serverless functions` all'interno della directory `/api` di Next.js e gestiscono funzionalità essenziali come la creazione di sessioni di chat, l'elaborazione dei documenti, la comunicazione con il modello AI e l'accesso ai file. Tutte le route sono pensate per garantire modularità, chiarezza e robustezza nella gestione degli errori.

/api/chat – Interazione con il modello LLM

Questa API funge da ponte tra il frontend dell'applicazione e l'intelligenza artificiale locale gestita da **Ollama**. Riceve un messaggio dell'utente dal client e restituisce una risposta generata dal modello linguistico scelto. Se disponibile, include nel prompt anche un contesto informativo derivato da un documento PDF caricato in precedenza.

Funzionamento L'API accetta una richiesta `POST` contenente:

- `model`: il nome del modello AI selezionato (es. `deepseek-coder`);
- `messages`: un array di messaggi strutturati secondo il formato `[role: 'user', content: '...']`;
- `documentId`: opzionale, identifica il documento PDF caricato e associato alla chat.

Se è presente un documento, viene attivato il modulo di **retrieval semantico**, che estrae i chunk più rilevanti in base alla domanda dell'utente. Questi frammenti vengono poi inseriti come messaggio di sistema all'inizio della conversazione, così da fornire al modello un contesto utile.

Listing 3.3: Recupero dei chunk e costruzione del prompt

```
const chunks = await retrieveRelevantChunks(documentId,
   userMessage.content);
contextText = chunks
  .map((chunk) => `- (p.${chunk.page}) ${chunk.content
     }`)
  .join('\n');

const messagesWithContext = [
  ...(contextText ? [{
    role: 'system',
    content: `Questi sono degli estratti rilevanti dal
       documento caricato:\n\n${contextText}\n\nRispondi
        tenendo conto di queste informazioni se
       pertinenti.`,
  }] : []),
  ...messages
];
```

Invocazione del modello La richiesta finale viene inviata tramite `fetch` a `localhost:11434/api/chat`, l'endpoint locale esposto da Ollama. Il parametro `stream` è impostato su `false`, quindi la risposta è ricevuta in blocco.

Listing 3.4: Chiamata all'API del modello

```
const response = await fetch("http://localhost:11434/api
   /chat", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
```

```
  body: JSON.stringify({
    model,
    messages: messagesWithContext,
    stream: false
  })
});
```

Risposta Il backend restituisce al frontend la risposta del modello sotto forma di JSON:

```
return Response.json({ response: data.message.content })
   ;
```

Error handling Il codice include anche un blocco `try/catch` per gestire eventuali errori durante il retrieval dei chunk, con logging esplicito:

```
try {
  const chunks = await retrieveRelevantChunks(...);
  // ...
} catch (err) {
  console.error("Errore durante il retrieval dei chunk:"
     , err);
}
```

Conclusioni L'endpoint `/api/chat` è centrale nel flusso dell'applicazione, in quanto gestisce:

- il collegamento tra messaggi utente e intelligenza artificiale;
- l'integrazione tra AI e contenuti PDF attraverso retrieval semantico;
- la formattazione e trasmissione sicura dei prompt.

Grazie alla sua struttura modulare, permette grande flessibilità e controllo sul comportamento del sistema.

/api/documents – Recupero dei documenti caricati

L'endpoint `/api/documents` fornisce al frontend la lista completa dei documenti PDF caricati nel sistema e memorizzati nel database. Viene principalmente invocato nella schermata `Home`, per mostrare all'utente le chat che includono un file allegato.

Funzionamento Questa route supporta esclusivamente richieste `GET`. Quando invocata, esegue una query SQL sulla tabella `documents` per recuperare:

- l'`id` del documento;
- il `filename` (nome originale del file PDF);
- il `chat_id` a cui è associato il documento.

Listing 3.5: Implementazione della GET in /api/documents

```
export async function GET() {
  try {
    const rows = db.prepare(`
      SELECT id, filename, chat_id
      FROM documents
      ORDER BY uploaded_at DESC
    `).all();

    return Response.json(rows);

  } catch (err) {
    console.error("Errore␣DB:", err);
    return new Response('Internal␣Server␣Error', {
       status: 500 });
  }
}
```

Comportamento I documenti vengono ordinati in ordine decrescente di `uploaded_at`, così da mostrare prima i file più recenti. Questo comportamento è utile per migliorare l'esperienza utente nella schermata principale dell'applicazione.

Risposta attesa La risposta è un array JSON del tipo:

```
[
  {
    "id": "abc123",
    "filename": "tesi_latex.pdf",
    "chat_id": "xyz789"
  },
  ...
]
```

Gestione errori In caso di errori nella connessione al database o durante l'esecuzione della query, l'API risponde con codice HTTP `500` e logga l'errore lato server.

Utilità Questa API è fondamentale per consentire:

- la visualizzazione dei documenti disponibili;
- il collegamento diretto a una specifica chat attraverso l'`chat_id` associato;
- il recupero rapido delle informazioni senza esporre i contenuti testuali o i file binari.

Essendo una **read-only API**, si presta anche a una eventuale futura implementazione di caching o indicizzazione dei metadati.

/api/download – Scaricamento del file PDF originale

L'endpoint `/api/download` permette agli utenti di scaricare il file PDF originale precedentemente caricato, mantenendone il nome e il contenuto intatto. Questa API è particolarmente utile quando si desidera recuperare il documento originale dopo aver interagito con esso tramite l'interfaccia conversazionale.

Funzionamento La route supporta richieste `GET` contenenti il parametro `documentId` all'interno della query string. Questo identificativo viene utilizzato per cercare nel database il corrispondente file binario (`file_blob`) e restituirlo come allegato HTTP.

Listing 3.6: Implementazione di /api/download

```
export async function GET(req) {
  try {
    const { searchParams } = new URL(req.url);
    const documentId = searchParams.get('documentId');

    if (!documentId) {
      return new Response('Missing␣documentId', { status
         : 400 });
    }

    const row = db.prepare(`
      SELECT filename, file_blob FROM documents WHERE id
          = ?
    `).get(documentId);

    if (!row || !row.file_blob) {
      return new Response('File␣not␣found', { status:
         404 });
    }
```

```
    return new Response(row.file_blob, {
      status: 200,
      headers: {
        'Content-Type': 'application/pdf',
        'Content-Disposition': `attachment; filename="${
            row.filename}"`,
      }
    });

  } catch (err) {
    console.error("Download route error:", err);
    return new Response("Internal server error", {
       status: 500 });
  }
}
```

Meccanismo di risposta Il file PDF viene restituito nel corpo della risposta come blob binario. Vengono inoltre aggiunte intestazioni HTTP per:

- impostare il tipo MIME corretto (`application/pdf`);
- forzare il download del file (`Content-Disposition: attachment`).

Esempio di richiesta

```
GET /api/download?documentId=abc123
```

Gestione errori Sono previsti i seguenti casi di errore:

- Se manca il parametro `documentId` → errore `400 Bad Request`.
- Se il documento non viene trovato nel database → errore `404 Not Found`.
- Per altri problemi interni → errore `500 Internal Server Error`.

Considerazioni Questa API offre una funzionalità cruciale per garantire all'utente la possibilità di recuperare il documento sorgente, contribuendo alla trasparenza e reversibilità delle operazioni svolte nel sistema. Il file viene scaricato nella sua forma originale, senza alterazioni dovute all'elaborazione successiva (chunking, embedding, ecc.).

/api/createChat – Creazione di una nuova sessione di chat

Questa API permette di generare una nuova sessione di chat, producendo un identificatore univoco che verrà poi utilizzato per associare messaggi e documenti.

Metodo supportato `POST`

Funzionamento Viene generato un `UUID` (Universal Unique Identifier) tramite la libreria `crypto`, e una nuova riga viene inserita nella tabella `chats` del database SQLite.

Listing 3.7: Implementazione di /api/createChat

```
import db from '@/db/init';
import { randomUUID } from 'crypto';

export async function POST(req) {
  try {
    const chatId = randomUUID(); //Genera un ID univoco

    db.prepare(`
      INSERT INTO chats (id)
      VALUES (?)
    `).run(chatId); //Inserisce nel DB

    return new Response(JSON.stringify({ chatId }), {
      status: 200,
      headers: { 'Content-Type': 'application/json' }
```

```
    });

  } catch (err) {
    console.error("
␣␣␣␣Errore␣nella␣creazione␣chat:", err);
    return new Response("Errore␣interno", { status: 500
       });
  }
}
```

Risposta

```
{
  "chatId": "abc123-def456"
}
```

Scopo Separare logicamente ogni interazione dell'utente con il sistema, facilitando il salvataggio e il recupero delle conversazioni e dei documenti allegati.

/api/deleteChat – Eliminazione completa di una sessione di chat

Questa API consente di cancellare in modo ricorsivo una chat esistente, rimuovendo anche:

- i messaggi associati;
- eventuali documenti caricati;
- i chunk vettoriali derivati dai documenti.

Metodo supportato `POST`

Funzionamento Riceve in input il campo `chatId`, e procede con l'eliminazione in cascata di tutte le entità correlate (documenti, chunks e messaggi) prima di cancellare la riga della chat.

Listing 3.8: Implementazione di /api/deleteChat

```
import db from '@/db/init';

export async function POST(req) {
  try {
    const { chatId } = await req.json();

    const documents = db.prepare(`
      SELECT id FROM documents WHERE chat_id = ?
    `).all(chatId);

    for (const doc of documents) {
      db.prepare(`DELETE FROM chunks WHERE document_id =
          ?`).run(doc.id);
    }

    db.prepare(`DELETE FROM documents WHERE chat_id =
       ?`).run(chatId);
    db.prepare(`DELETE FROM messages WHERE chat_id = ?`)
       .run(chatId);
    db.prepare(`DELETE FROM chats WHERE id = ?`).run(
       chatId);

    return new Response('Chat eliminata con successo', {
        status: 200 });

  } catch (err) {
    console.error("Errore eliminazione chat:", err);
    return new Response('Errore interno', { status: 500
       });
```

```
  }
}
```

Considerazioni Questa API è cruciale per garantire la corretta gestione dello spazio e della privacy dell'utente. Eliminando non solo la chat, ma anche tutti i dati associati, si evita l'accumulo di informazioni inutili nel database e si assicura che ogni conversazione sia effettivamente cancellabile su richiesta dell'utente.

Esempio di richiesta

```
{
  "chatId": "abc123"
}
```

pages/api/file.js – Upload, Parsing e Indicizzazione dei Documenti

Questa API rappresenta uno degli snodi fondamentali dell'intero sistema, in quanto si occupa della ricezione e indicizzazione dei file PDF caricati dall'utente.

Metodo supportato `POST`

Funzionalità principali

- Ricezione e salvataggio del file PDF in formato binario.
- Parsing del contenuto testuale tramite `pdfjs`.
- Suddivisione del testo in *chunk* sovrapposti.
- Calcolo degli **embedding** semantici tramite modello locale.
- Salvataggio nel database: file, chunk e metadati.

Parsing con PDF.js L'API utilizza una build compatibile con Node.js di `pdfjs-dist`, disabilitando il `worker` in quanto non necessario in ambiente server.

Suddivisione in chunk Ogni pagina del PDF viene processata e suddivisa in blocchi di massimo 500 caratteri, con una sovrapposizione di 100 caratteri tra un chunk e l'altro:

```
function chunkText(text, maxLength = 500, overlap = 100)
    {
  const chunks = [];
  let start = 0;
  while (start < text.length) {
    const end = Math.min(start + maxLength, text.length)
       ;
    const chunk = text.slice(start, end);
    chunks.push(chunk.trim());
    start += maxLength - overlap;  // sovrapposizione
  }
  return chunks;
}
```

Esempio: Embedding dei chunk e salvataggio nel database Ogni chunk viene passato alla funzione `getEmbedding`, che restituisce un vettore numerico rappresentativo. I chunk e i loro embedding vengono poi memorizzati nella tabella `chunks`, mentre il file originale viene salvato nella tabella `documents`.

```
const insertChunk = db.prepare(`
  INSERT INTO chunks (document_id, page, content,
     embedding)
  VALUES (?, ?, ?, ?)
`);
for (const chunk of embeddedChunks) {
```

```
  insertChunk.run(
    documentId,
    chunk.page,
    chunk.content,
    JSON.stringify(chunk.embedding)
  );
}
```

Risposta La risposta restituisce al frontend l'ID del documento appena creato, il nome del file e i chunk indicizzati:

```
{
  "documentId": "abc123",
  "filename": "documento.pdf",
  "chunks": [...]
}
```

Considerazioni L'API `file.js` è particolarmente articolata poiché svolge sia la funzione di caricamento file che quella di preprocessing. La separazione in chunk e l'embedding locale garantiscono efficienza e privacy, eliminando la necessità di servizi esterni.

pages/api/retrieve.js – Recupero semantico dei contenuti

L'API `retrieve.js` espone un endpoint dedicato al recupero dei frammenti (*chunk*) più pertinenti di un documento PDF in base ad una query dell'utente. Tale processo costituisce il cuore della componente di **retrieval semantico**.

Metodo supportato `POST`

Payload richiesto

```
{
```

```
  "documentId": "abc123",
  "question": "Quali␣sono␣le␣conclusioni␣del␣documento?"
     ,
  "topK": 5
}
```

Funzionamento Dopo aver ricevuto la richiesta, l'API:

1. Verifica che siano presenti i parametri `documentId` e `question`.
2. Calcola l'**embedding** della domanda tramite la funzione `getEmbedding`.
3. Recupera dal database tutti i chunk associati al documento.
4. Calcola la similarità coseno tra la domanda e ciascun chunk.
5. Ordina i risultati per rilevanza e restituisce i migliori `topK`.

Snippet semplificato del codice

```
import { retrieveRelevantChunks } from '@/utils/
   retrieval';

export default async function handler(req, res) {
  if (req.method !== 'POST') {
    return res.status(405).json({ error: 'Metodo␣non␣
       consentito' });
  }

  const { documentId, question, topK } = req.body;

  if (!documentId || !question) {
    return res.status(400).json({ error: 'documentId␣e␣
       question␣sono␣obbligatori' });
  }
```

```
  const chunks = await retrieveRelevantChunks(documentId
     , question, topK || 5);
  return res.status(200).json({ chunks });
}
```

Esempio di risposta

```
{
  "chunks": [
    {
      "content": "Conclusione␣del␣documento...",
      "page": 12,
      "score": 0.84
    },
    ...
  ]
}
```

Utilità Questa API è utile:

- per eseguire interrogazioni mirate su documenti caricati;
- per visualizzare nel frontend (es. nella modalità "highlight") i chunk più rilevanti;
- come componente stand-alone di ricerca testuale avanzata.

Nota tecnica L'intero processo sfrutta il modello `nomic-embed-text` per generare gli embedding vettoriali, mantenendo il tutto **in locale**, a garanzia della privacy e dell'efficienza.

api/models – Elenco dei modelli AI disponibili

L'endpoint `/api/models` è responsabile del recupero dinamico dei **modelli linguistici** installati localmente tramite **Ollama**, escludendo quelli non adatti all'interazione conversazionale.

Metodo supportato `GET`

Funzionalità Questa API:

- invia una richiesta all'endpoint locale `http://localhost:11434/api/tags`, fornito da Ollama;
- filtra i modelli restituiti, rimuovendo eventuali modelli non utili (es. `nomic-embed-text`);
- restituisce un elenco JSON dei modelli utilizzabili nel frontend, dove l'utente può selezionarli tramite menu a tendina.

Estratto del codice

```
export async function GET(req) {
  const res = await fetch("http://localhost:11434/api/
     tags", {
    method: "GET"
  });

  if (!res.ok) {
    return Response.json({ error: 'Impossibile␣
       recuperare␣i␣modelli.' }, { status: 500 });
  }

  const data = await res.json();

  // Esclude il modello per embedding
  const filteredModels = data.models.filter(
    model => model.name !== "nomic-embed-text:latest"
  );

  return Response.json({ models: filteredModels });
}
```

Esempio di risposta

```
{
  "models": [
    { "name": "deepseek-coder:latest" },
    { "name": "llama2:7b" },
    { "name": "mistral:instruct" }
  ]
}
```

Utilità Questa API consente di rendere l'interfaccia flessibile, permettendo agli utenti di **scegliere il modello AI** più adatto alle loro esigenze (es. precisione, velocità, dimensione del modello), migliorando l'esperienza e l'adattabilità del sistema.

Gestione degli errori e sicurezza

Durante la progettazione delle API backend, particolare attenzione è stata dedicata alla gestione degli errori e alla robustezza del sistema. Ogni route è stata strutturata per:

- intercettare **richieste non valide** (es. metodi HTTP errati, parametri mancanti);
- restituire **codici di stato HTTP** coerenti con la tipologia dell'errore (400, 404, 405, 500, ecc.);
- loggare gli errori lato server tramite `console.error` per facilitare il debugging.

Esempio: gestione errori in `/api/file.js`

```
if (req.method !== 'POST') {
  return res.status(405).json({ error: 'Metodo non
     consentito' });
}
```

Validazione degli input Tutte le API che ricevono dati in ingresso (es. `/saveMessage`, `/retrieve`, `/chat`) eseguono controlli espliciti sui campi richiesti:

```
if (!chatId || !role || !content) {
  return new Response("Missing␣fields", { status: 400 })
     ;
}
```

Protezione da accessi imprevisti Anche se il sistema è progettato per funzionare in **ambiente locale**, l'adozione di controlli e messaggi di errore espliciti rende il backend più solido e pronto per future evoluzioni in ambienti multiutente o distribuiti.

Questi accorgimenti contribuiscono alla costruzione di un backend affidabile, chiaro nei messaggi di errore e resistente a comportamenti imprevisti, migliorando l'esperienza dell'utente finale e facilitando la manutenzione del codice.

Conclusioni

Il lavoro descritto in questa tesi ha portato alla realizzazione di un'applicazione prototipale che consente l'analisi semantica di documenti PDF attraverso un'interfaccia conversazionale basata su modelli linguistici di grandi dimensioni (LLM) eseguiti in locale. L'obiettivo principale, ovvero fornire uno strumento pratico, user-friendly e rispettoso della privacy per facilitare l'interazione con testi complessi, può dirsi raggiunto.

A partire dall'analisi dei requisiti, il progetto ha affrontato in maniera strutturata le fasi di progettazione architetturale, selezione delle tecnologie, sviluppo frontend e backend, nonché l'integrazione dei modelli AI tramite il sistema Ollama. Particolare attenzione è stata posta sulla conservazione dei dati in locale e sull'esperienza d'uso, elementi cruciali in ambito aziendale e professionale, specialmente alla luce delle recenti normative europee in materia di protezione dei dati.

Durante lo sviluppo, sono state affrontate sfide significative, tra cui la gestione dei documenti binari in database SQLite, la frammentazione semantica (chunking), l'embedding dei contenuti testuali e l'individuazione dei passaggi più rilevanti da sottoporre all'LLM in risposta alle domande dell'utente. L'interfaccia è stata progettata per risultare intuitiva anche per utenti non tecnici, offrendo funzionalità come il caricamento e il download dei file, la persistenza delle chat e un flusso conversazionale naturale.

Possibili sviluppi futuri

Nonostante il progetto abbia raggiunto gli obiettivi prefissati, esistono numerose direzioni lungo le quali l'applicazione potrebbe evolvere ulteriormente:

- **Ottimizzazione delle prestazioni**: Sebbene l'uso locale dei LLM offra vantaggi in termini di privacy, comporta anche requisiti hardware rilevanti. Un'area di miglioramento potrebbe essere l'adozione di tecniche di quantizzazione o modelli più leggeri e performanti per dispositivi meno potenti. Un'altra direzione interessante potrebbe consistere nel predisporre un server dedicato, il quale permetterebbe, ad esempio, di far girare un modello AI più potente e in tempi di attesa ridotti.

- **Estensione ai formati multipli**: Al momento il sistema è pensato principalmente per PDF, ma potrebbe essere esteso a supportare formati testuali aggiuntivi, come DOCX, HTML o addirittura pagine web.

- **Funzionalità avanzate di interazione**: Potrebbero essere aggiunte funzionalità come la generazione automatica di sommari, l'estrazione di dati strutturati (tabelle, grafici), o la creazione di "mappe concettuali" dinamiche del contenuto.

- **Addestramento su documenti personalizzati**: Un'ulteriore evoluzione potrebbe consistere nell'adattare il comportamento dell'LLM ai documenti più frequentemente usati da una determinata azienda, creando modelli specializzati tramite tecniche di fine-tuning o retraining parziale.

- **Gestione utenti e autenticazione**: Un'evoluzione naturale dell'applicativo potrebbe prevedere l'introduzione di funzionalità di login e registrazione, con la possibilità di creare profili utente personalizzati, gestire le conversazioni in modo privato e sicuro, e differenziare i livelli

di accesso. Questo garantirebbe una maggiore protezione dei dati e una personalizzazione dell'esperienza d'uso.

- **Collaborazione tra utenti e condivisione**: Una volta introdotta la gestione degli utenti, si potrebbero sviluppare funzionalità collaborative come la condivisione di chat, documenti o modelli tra membri di uno stesso team, la creazione di gruppi di lavoro, o l'integrazione di funzionalità di commento e modifica condivisa.

- **Integrazione con sistemi aziendali esistenti**: Infine, un'eventuale integrazione dell'applicativo con CRM, ERP o altri sistemi informativi aziendali potrebbe renderlo parte integrante del workflow professionale, aumentando notevolmente la sua utilità operativa.

In conclusione, il progetto si pone come un punto di partenza concreto verso l'integrazione dell'intelligenza artificiale nel supporto all'analisi documentale, dimostrando come sia possibile costruire soluzioni potenti, accessibili e rispettose della privacy anche in ambito locale. La flessibilità dell'architettura e la modularità delle componenti implementate aprono ampie possibilità di estensione e perfezionamento, rendendo questa applicazione non solo un valido prototipo, ma anche un potenziale prodotto da evolvere in contesti professionali reali.

Bibliografia

[1] Erik Brynjolfsson and Andrew McAfee. *The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies*. W. W. Norton & Company, 2014.

[2] Michael Haenlein and Andreas Kaplan. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California Management Review*, 61(4):5–14, 2019.

[3] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education Limited, 3 edition, 2016. ISBN 978-0136042594.

[4] Thomas H Davenport and Rajeev Ronanki. Artificial intelligence for the real world. *Harvard Business Review*, 96(1):108–116, 2018.

[5] Eric Topol. *Deep Medicine: How Artificial Intelligence Can Make Healthcare Human Again*. Basic Books, New York, 2019.

[6] Daniel Jurafsky and James H Martin. *Speech and language processing*. Pearson Prentice Hall, 2 edition, 2009. ISBN 9780131873216.

[7] Andre W Kushniruk and Elizabeth M Borycki. Natural language processing and the representation of clinical documents. *Yearbook of Medical Informatics*, 29(01):70–77, 2020.

[8] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, et al. Language mo-

dels are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[10] Rishi Bommasani, Jack Hudson, Ehsan Adeli, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021. URL `https://arxiv.org/abs/2108.07258`.

[11] Kate Saenko. A computer scientist breaks down generative ai's hefty carbon footprint. *Scientific American*, 2023. URL `https://www.scientificamerican.com/article/a-computer-scientist-breaks-down-generative-ais-hefty-carbon-footprint/`.

[12] UK Government. Microsoft 365 copilot experiment: Cross-government findings report. `https://www.gov.uk/government/publications/microsoft-365-copilot-experiment-cross-government-findings-report`, 2024. Accessed: 2025-06-04.

[13] Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Jianfeng Gao, and Bill Dolan. Dialogpt: Large-scale generative pretraining for conversational response generation. *arXiv preprint arXiv:1911.00536*, 2020.

[14] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[15] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kulkarni, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-

intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[16] Margot E Kaminski and Gianclaudio Malgieri. Algorithmic impact assessments under the gdpr. *International Data Privacy Law*, 9(1):11–36, 2019.

[17] Paul Voigt and Axel Von dem Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Springer International Publishing, 2017. ISBN 9783319579580.

[18] Meta Platforms, Inc. React – a javascript library for building user interfaces. `https://react.dev`, 2024. Accessed: 2025-06-04.

[19] Guillermo Rauch. Next.js documentation. `https://nextjs.org/docs`, 2021. Accessed: 2025-06-04.

[20] D. Richard Hipp. Sqlite: past, present, and future. *IEEE Software*, 27 (6):118–120, 2010.

[21] Ollama. Ollama: Run large language models locally. `https://ollama.com`, 2024. Accessed: 2025-06-04.

[22] Mozilla Foundation. Pdf.js – a general-purpose, web standards-based platform for parsing and rendering pdfs. `https://mozilla.github.io/pdf.js/`, 2024. Accessed: 2025-06-04.

[23] The Bootstrap Team. Bootstrap 5 documentation. `https://getbootstrap.com/docs/5.0/getting-started/introduction/`, 2021. Accessed: 2025-06-04.

[24] Latex.

Ringraziamenti

Premetto che mi scuso se rimarrete sorpresi nel vedermi adottare un tono strettamente colloquiale in quest' ultima sezione, ma ci tengo a far suonare le parole che seguono nella maniera più umana e sincera possibile, distaccandomi dalla freddezza del linguaggio formale utilizzato finora.

Innanzitutto, ci tengo a ringraziare la mia **relatrice Catia Prandi**, che ha accolto da subito la mia proposta di progetto con entusiasmo e ha poi seguito con interesse la stesura della relativa tesi.

Ringrazio poi **Stefano Cemin**, il quale mi ha proposto e permesso di sviluppare questo progetto ambizioso, e con lui tutto il resto del meraviglioso **team di Brainy Labs**. Non ho mai capito perché abbiate deciso di prendermi come tirocinante nonostante fin dal colloquio io avessi detto chiaramente di non esser tagliato per fare l'ingegnere e di non aver esperienza né con i backend né con i database; e nonostante ciò, mi avete trattato dal primo giorno come fossi un vero membro del team, facendomi sentire tutt'altro che fuori posto e permettendomi così di crescere e migliorare. Sono stati quattro mesi fantastici che ricorderò sempre con affetto, tuttavia sappiate che se in futuro dovessi trovarmi male in un'azienda, sarà solo colpa vostra che mi avete viziato troppo!

Vorrei ringraziare anche tutto l'accoglientissimo e gentilissimo staff di **Volume**, ho passato molte più ore al vostro bar che in qualsiasi aula di questo

Campus. In particolare ringrazio **Dario**, una persona estroversa e di gran cuore, che fin da subito si è rivelato per me molto più di un semplice barista, diventando un amico a tutti gli effetti con cui ho sempre piacere a conversare.

Ci tengo anche a ringraziare tutti **gli amici di JMP** per aver reso questo percorso tortuoso una sorta di esperienza di gruppo, rendendo così il viaggio molto più piacevole e leggero. Uno dei miei ricordi preferiti sarà sempre il primo appello di Algebra e Geometria nell'estate del primo anno: un caldo afoso, l'aula stracolma, un'ansia della madonna e noi, tipo Avengers, carichissimi, che ci facciamo forza ed entriamo tutti insieme come se il fatto di essere un gruppo unito ci permettesse di passare l'esame pur non avendo studiato a dovere (così non fu, lol). In particolare, del gruppo JMP, vorrei ringraziare **Piso** e **Giò**, assieme ai quali ho affrontato le prime difficoltà e assieme ai quali ho riso di più in assoluto, e poi la **Mati**, senza la quale non mi sarebbe mai riuscito di preparare la maggior parte degli esami in tempo record; sei stata un'amica fondamentale, non so come farò in futuro a studiare su dei fogli che non sono dei tuoi appunti magici super riassuntivi. Un grazie speciale però va anche alla **Gaiuz**, la persona con la quale ho legato di più in questi anni di università. Sono veramente felice di averti incontrata, sei un'amica preziosa e una persona dolcissima; spero vivamente che continueremo a giocare a Pokémon insieme, in qualsiasi parte del globo ci troveremo. Ti ringrazio per tutto il supporto, soprattutto per quello fornitomi in questo mio ultimo periodo così difficile; sappi che non do affatto per scontato tutto quello che hai fatto per me.

Ringrazio **i miei genitori**, per il supporto fornitomi in questi anni di studio. Mi dispiace se non sono esattamente come avreste voluto, tuttavia spero, alla luce di questo traguardo, di avervi reso almeno un po' fieri di me.
Ringrazio **Kurosh**, il più grande dei miei fratelli maggiori, per avermi dato fin dai miei primi giorni di università dei consigli sul come approcciarmi ad essa e vivermela al meglio. Ti ringrazio per avermi fatto spesso rivalutare

delle situazioni che per me erano tragiche, ricordandomi quanto piccole in realtà fossero se paragonate al lungo e personalissimo viaggio che è la vita vera. In realtà sono sempre stato parecchio influenzato dal tuo pensiero in quanto ti ho sempre rispettato molto per esser stato il primo fratello ad esser stato gentile con me e ad avermi compreso. Per me sei sempre stato una sorta di amico-mentore, anche se a guardarmi non si direbbe visto che purtroppo non ho minimamente ereditato la tua enorme cultura invece.

Grazie **Nader** per avermi ospitato tutto questo ultimo periodo in casa tua a Bologna. Se dicessi al me di tre anni fa che sono riuscito a convivere pacificamente sotto al tuo stesso tetto senza mai litigare, non mi crederebbe mai. Sono infatti felice di avere finalmente un vero legame con te e di riuscire a comprenderci nonostante le nostre diversità. Sei ostinato e hai una corazza che non rende facile capire come prenderti, ma sotto di essa c'è una persona che ama aprirsi, estremamente emotiva, e molto più riflessiva di quello che si può pensare. Mi mancherà intrattenerci a vicenda con i nostri racconti e guardare Hunter x Hunter mentre ceniamo (mi mancheranno un po' meno le pulizie invece ihih).

Grazie **Keivan** per avermi aiutato nelle primissime sfide di questa facoltà; ricordo ancora quella lunghissima videochiamata Fano-Torino in cui mi spiegasti cosa fossero le flag e come utilizzarle, mi fece sentire fortissimo. So che, vista la tua esperienza in materia, in questi anni avrei potuto chiederti aiuto innumerevoli volte per i miei esami, ma se non l'ho mai fatto è un po' perché temevo potessi sentirti usato, ma soprattutto perché sono cocciuto: ho sempre ammirato la facilità e rapidità con la quale hai finito il mio stesso percorso, e siccome tu non avevi avuto un fratello maggiore che ti spiegava tutto all'occorrenza, io non volevo essere da meno.

Grazie **Vida** per esserci stata quando avevo bisogno che qualcuno credesse in me. Francamente, non ho mai capito cosa tu ci veda in me per ammirarmi come tanto fai. Tuttavia, proprio per questo motivo, certe volte mi sono sentito particolarmente spinto a non arrendermi per paura di tradire le speranze che avevi riposto in me o perché mi sentivo in dovere di darti il buon

esempio. Quale fratello maggiore vorrebbe mai perdere davanti agli occhi della sua sorellina? Mi dispiace se ogni tanto sono un po' duro con te, ma è frutto di questo stesso sentimento che mi fa sentire così tanto responsabile nei tuoi confronti. Al di là delle discordanze, amo vederti fiorire, e non vedo l'ora di poter vedere i tuoi splendidi frutti. Sappi che per me ormai, tu non sei solo parte della mia famiglia, ma anche una mia amica a tutti gli effetti. E ovviamente, grazie **nonno**, per avermi permesso di poter conseguire i miei studi senza problemi. È solo merito tuo se ho potuto permettermi di scegliere il corso a piacimento e poi di prendermi i miei tempi per finirlo. Te ne sarò per sempre molto grato; sono fortunato ad avere un nonno così, e per tanto, vedrò sempre di essere un nipote degno di te, indipendentemente da quanto tu possa, involontariamente, vedermi offuscato.

Infine, ringrazio anche **tutti i miei amici di Fano** che mi hanno da sempre sostenuto, non solo in questo percorso universitario, ma nella vita in generale. Mi avete sempre dimostrato di poter contare su di voi: avete riparato danni di ogni tipo per me, sebbene non ne foste i responsabili, mi avete consolato e irradiato, aiutandomi ad uscire vittorioso dalle situazioni più oscure. Siete l'unico motivo per il quale io mi senta così realizzato nella vita. Ogni tanto ripenso a quando ero bambino e abitavo a Pergola: esprimevo spesso, nel mio diario segreto, la preoccupazione che provavo nel non riuscir a formare amicizie (ero persino arrivato ad inventarmi un amico immaginario!). Io non sognavo di diventare un ingegnere, non sognavo di diventare un artista; sognavo, più di ogni altra cosa, di trovare dei veri amici. E credo sia per questo che adesso, nella vita, mi sento sempre così agiato e non troppo motivato: dopotutto, io il mio sogno l'ho già realizzato.

Ci tengo a menzionare nello specifico qualche nome...

Angelica, la mia cara amichetta fin dalle medie. Sei una persona impeccabile, al contrario mio, e per questo ti sono grato per essere comunque rimasta al mio fianco in tutti questi anni di amicizia. Ti sei sempre dimostrata una persona di grande valore e sulla quale poter contare nei momenti più bassi

che ho avuto. Nonostante non riesca a vederti spesso, rimani una delle mie amiche più fondamentali e un punto di riferimento per me.

Edoardo, col quale sono amico solo da pochi anni, ma col quale ho formato fin da subito una complicità unica. Sei un amico, Edo. Sei una persona che, a mio parere, chiunque necessiterebbe affianco. A primo impatto sembri solo un idiota ma in realtà sei un ragazzo empatico, affettuoso, (brutalmente) sincero, tenace e capace di adattarsi ad un sacco di situazioni e persone.... e poi sei anche un idiota. Mi trovo davvero bene con te e spero che il futuro ci riservi tante altre avventure insieme.

Lorenzo, una delle persone più talentuose che io conosca. Ironico come quelle spigolose corna da diavolo, vengano indossate da un ragazzo tanto allegro, sensibile, aperto e disponibile. Sai tirarmi su il morale e dimostrarmi affetto in un modo tutto tuo. Non mi stancherò mai di fare le nostre seratine al tuo attico. Vado molto fiero della nostra amicizia, sei una persona super capace, piena di idee e con tanta voglia di mettersi in gioco. Sono sicuro che arriverai a fare grandi cose e io non mancherò a sostenerti.

Maddalena, la ragazza più solare che io conosca. Sei fortissima, nonostante le fatiche che affronti sai sempre scaldarci tutti con i tuoi sorrisi, e per questo ti stimo così tanto. Sei una persona alla quale non si può che voler bene. Non smetterò mai di ringraziarti per avermi insegnato a guidare nonostante i rischi a cui andavi incontro, sei davvero tosta. Non so come tu faccia a bombardarmi puntualmente di affetto e complimenti anche quando non ne meriterei nemmeno la metà. Sono felicissimo di averti al mio fianco, e sono tranquillo perché so che, per come siamo fatti noi due, non potremmo mai perderci.

Martina, con la quale ho condiviso, oltre alle varie belle esperienze, tutto il mio ultimo anno di studio alla MeMo. La tua presenza così gentile e piacevole mi ha aiutato tanto in quei periodi bui che sanno essere le sessioni. Sei una persona sulla quale si può davvero fare affidamento, ma paradossalmente ogni tanto mi sembrava avessi più fiducia in me di quanto non ne avessi in te stessa. Magari mi sbaglio ma, secondo me, sei troppo umile per vedere

quanto tu sia veramente forte. Spero, dovunque saremo nei prossimi anni, di non perderti né come compagna di studi né (soprattutto) come amica.

Marco, una delle mie persone preferite in assoluto. Non so nemmeno cosa dire su di te. Di una cosa però sono certo: la tua sola esistenza mi rallegra. Seriamente, spesso mi capita che, quando mi perdo in profondi tunnel oscuri, io a una certa pensi: "sono sicuro che però, se ora fossi con Marco, non starei così". Amo il modo in cui siamo così simili e allo stesso tempo così diversi. Amo il modo in cui mi stimi e allo stesso tempo non mi sopporti. Sei indispensabile nella mia vita, io a te mi sento connesso nel profondo. Il tuo modo di contemplare la vita, di affrontare i problemi, di approcciarti a me... ecco, se dovessi racchiuderti in una frase, direi questo: io non potrei vivere senza poter parlare con te, davvero. Sei una persona davvero unica, sei un ragazzo formidabile, quindi ti prego, quei tuoi sogni non lasciarli nel cassetto, hai veramente tanto da dare a questo mondo. Non vedo l'ora di scoprire dove finiranno due sbandati come noi: è molto probabile che, distratti dalle nostre chiacchiere, sbaglieremo sicuramente strada una o due volte, ma so per certo che, seppur col nostro discutibile modo di fare (e col nostro tipico ritardo), arriveremo in fondo, insieme.

Sveva, che effettivamente, oltre ad avere ogni preziosa qualità per la quale mi piace definirla anche un'amica, è prima di tutto la mia ragazza. Sono fortunato ad aver trovato una persona come te. Cosa sarebbe successo se l'ultimo giorno di liceo non ti avessi tirato addosso quella secchiata d'acqua? Sai che non esagero se dico che, in tal caso, ora probabilmente non sarei qui. Sono orgoglioso di avere al mio fianco una ragazza con una personalità così forte e decisa. Ti ringrazio per credere così tanto in me e ti ringrazio per guardarmi sempre con quello sguardo che risplende di un sentimento sincero. Negli anni ho sviluppato con te una complicità speciale, che va ben oltre il legame della relazione. Sei forse la persona che mi conosce meglio in assoluto e non riesco ad immaginare una vita della quale tu non faccia assolutamente parte in nessun modo. Sveva, la verità è che mi è tremendamente facile smarrirmi quando i tuoi fulgenti occhi non puntano la mia fosca pelle.

Per ultimo, vorrei ringraziare colui senza il quale non sarei davvero mai arrivato in fondo a questo percorso, né a innumerevoli altri. Parlo di te, **Diotta**. Ti devo davvero tanto, mi sei accanto da prima di chiunque altro. Sei il mio vanto più grande e non ti si può non ammirare. Da piccoli eravamo identici, te lo ricordi? Due personalità totalmente sovrapponibili. Tuttavia, crescendo, stiamo assumendo entrambi una forma sempre più personale e... io vorrei tanto essere forte quanto te, vorrei tanto essere determinato quanto te. Io nemmeno me lo spiego cosa ti spinga a far parte, in maniera così impegnativa, della mia vita. Molto spesso, le persone mi paragonano ad un Sole; mi reputano una persona solare, in grado di irradiare gli altri con la propria luce. Ma quando penso a te, io mi sento un satellite. Io temo di essere al massimo una Luna, la quale può anche saper illuminare, ma non brilla certo di luce propria. La Luna non fa altro che riflettere la luce del Sole. Io mi sento così, senza di te sono sicuro che non sarei in grado di irradiare nessuno. Mi dispiace, vorrei tanto poterti restituire, almeno in parte, questa luce che mi trasmetti da ormai tredici anni. Per il momento posso soltanto limitarmi a ringraziarti. Ti aspetti forse che adesso io mi disperi perché, dopo questo così lungo cammino condiviso, le nostre strade si separeranno? Ti sbagli, ora potremmo anche intraprendere due strade completamente diverse che portano a due galassie opposte: io non ti lascerò andare. Non perché io abbia paura di rimanere al buio, ma perché non potrei mai permettere che ci rimanga tu. Quella luce, quella luce che mi hai donato, io te la restituirò tutta quanta. È una promessa, amico.

Concludo dicendo che, di base, ringrazio chiunque sarebbe disposto ad alzare le proprie mani al cielo per donarmi la sua energia nel caso io stessi caricando la Sfera Genkidama.