Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il Management

SVILUPPO DI UN'APP ANDROID PER LO STUDIO BASATA SU SISTEMA RAG

(RETRIEVAL-AUGMENTED-GENERATION)

Relatore:
Prof.
MONTORI FEDERICO

Presentata da:
IELO
FRANCESCO
GIUSEPPE

Sessione I
Anno Accademico 2024/2025

Dedicato a...
tutti coloro che hanno
fatto parte del mio percorso
Grazie

Abstract

L'avvento dei Large Language Models (LLM) ha rivoluzionato il campo dell'intelligenza artificiale, mostrando capacità senza precedenti nella comprensione e generazione del linguaggio. Tuttavia, limiti intrinseci come la tendenza a produrre informazioni inaccurate (allucinazioni), la conoscenza statica e la mancanza di tracciabilità delle fonti ne ostacolano l'adozione in contesti critici. La letteratura scientifica ha proposto l'architettura Retrieval-Augmented Generation (RAG) come soluzione a tali problemi, integrando i modelli generativi con basi di conoscenza esterne per ancorare le risposte a dati verificabili.

Nonostante il progresso tecnologico, si è identificata una lacuna nel settore degli strumenti per l'apprendimento: l'assenza di una soluzione integrata che supporti lo studente in un ciclo di studio completo. Le applicazioni attuali sono spesso frammentate, consentendo di interagire con un singolo documento o offrendo funzionalità di IA generiche non ottimizzate per lo studio.

Il presente lavoro di tesi affronta questo problema attraverso la progettazione e lo sviluppo di "MyStudyApp", un'applicazione mobile nativa per Android. Sfruttando un'architettura RAG basata su servizi cloud Microsoft Azure, l'applicazione funge da compagno di studi intelligente, permettendo agli utenti di caricare i propri materiali, dialogare con essi tramite una chat contestuale, generare quiz personalizzati per l'autovalutazione e monitorare i progressi.

La validazione del sistema è stata condotta tramite una valutazione sperimentale comparativa tra l'architettura cloud implementata e un'alternativa basata su modelli eseguiti in locale. I risultati dimostrano la netta superiorità dell'approccio cloud in termini di qualità, accuratezza delle risposte e latenza, confermando la validità delle scelte progettuali per offrire uno strumento di studio efficace, affidabile e performante.

Introduzione

L'avvento dell'intelligenza artificiale generativa, e in particolare dei Large Language Models (LLM), ha segnato una delle più significative rivoluzioni tecnologiche dell'era moderna. Questi modelli hanno dimostrato capacità senza precedenti nella comprensione e nella produzione del linguaggio naturale, aprendo la strada a nuove applicazioni in innumerevoli settori. Tuttavia, nonostante il loro potenziale, gli LLM presentano delle limitazioni intrinseche ben note: la tendenza a generare informazioni non accurate o inventate (le cosiddette "allucinazioni"), una conoscenza statica e non aggiornata al momento del loro addestramento, e una mancanza di tracciabilità delle fonti che ne mina l'affidabilità in contesti critici.

Per superare queste sfide, la ricerca si è orientata verso architetture ibride, tra le quali spicca la Retrieval-Augmented Generation (RAG). Questo paradigma, che integra un modello generativo con una base di conoscenza esterna, permette di ancorare le risposte a un contesto specifico e verificabile, aumentando drasticamente l'affidabilità e la pertinenza delle informazioni prodotte. Il presente lavoro di tesi si inserisce in questo filone, con l'obiettivo di applicare l'architettura RAG per rispondere a un'esigenza concreta nel dominio dell'apprendimento: la mancanza di uno strumento di studio integrato che supporti lo studente in un ciclo di apprendimento completo.

A tal fine, è stata progettata e sviluppata "MyStudyApp", un'applicazione mobile nativa per Android che funge da compagno di studi intelligente. Sfruttando una robusta architettura client-server basata su servizi cloud Microsoft Azure, l'applicazione permette agli utenti di caricare i propri materiali

di studio, dialogare con essi attraverso una chat contestuale, generare quiz personalizzati per l'autovalutazione e monitorare i propri progressi. L'obiettivo è trasformare un processo di studio passivo in un'esperienza interattiva, efficace e personalizzata.

La presente tesi documenta l'intero percorso, dalla concettualizzazione alla realizzazione e validazione del sistema.

- Il **Capitolo 1** analizza lo stato dell'arte, introducendo l'architettura Transformer, la Document AI e i sistemi RAG, e definendo le motivazioni e il "gap" di mercato che hanno ispirato il progetto.

- Il **Capitolo 2** approfondisce il background tecnico, descrivendo nel dettaglio il funzionamento di una pipeline RAG, dalla suddivisione dei documenti alla ricerca vettoriale.

- Il **Capitolo 3** illustra il progetto e l'architettura ad alto livello di "MyStudyApp", descrivendo i servizi cloud impiegati e il flusso di navigazione dell'utente.

- Il **Capitolo 4** si addentra nei dettagli implementativi del backend, sviluppato in Python, e del frontend Android nativo, mostrando frammenti di codice significativi.

- Il **Capitolo 5** presenta la valutazione sperimentale condotta per validare il sistema, confrontando l'architettura cloud con un'alternativa locale attraverso metriche quantitative e qualitative.

- Infine, il **Capitolo 6** riassume i risultati ottenuti, discute le limitazioni del lavoro e delinea le possibili direzioni per gli sviluppi futuri del progetto.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Stato dell'arte

Il tema dell'intelligenza artificiale, è un tema più che mai ricorrente e presente in tutti gli ambiti della nostra vita. Sebbene il concetto di intelligenza artificiale esista da molto tempo, l'attenzione mediatica sull'argomento è aumentata esponenzialmente dopo l'avvento dell'IA generativa che ha profondamente segnato il panorama tecnologico moderno, inaugurando una nuova era nello sviluppo di sistemi intelligenti. A differenza dei modelli tradizionali, progettati principalmente per l'analisi e la classificazione di dati esistenti, l'IA generativa consente di generare dei contenuti come testi, immagini e audio apparantemente nuovi partendo da dei dati di addestramento [5]. In questo scenario in rapida evoluzione, i *Large Language Models* (LLM) si sono affermati come la manifestazione più potente e pervasiva di tale tecnologia. Questi modelli sono addestrati su quantità enormi di dataset e archivi di testo, hanno dimostrato capacità senza precedenti nella comprensione e nella produzione del linguaggio naturale umano, diventando in breve tempo una tecnologia onnipresente e trasformativa in moltissimi settori. Il loro successo e la loro rapida adozione sono direttamente attribuibili a un'innovazione architetturale chiave che ha ridefinito le fondamenta del Natural Language Processing: il modello Transformer.

1.1 L'Architettura Transformer: "Attention Is All You Need"

Per comprendere appieno le ragioni dietro l'eccezionale performance dei moderni LLM, è necessario analizzare l'architettura che ne costituisce le fondamenta: il Transformer. Proposto per la prima volta nel paper "Attention Is All You Need" da Vaswani et al. nel 2017 [17], questo modello ha rappresentato un punto di svolta nel campo del Natural Language Processing. Il Transformer è un'architettura di rete progettata per compiti di trasduzione di sequenze, questo modello si differenzia da quelli precedentemente utilizzati, ovvero reti neurali ricorrenti o convoluzionali (RNN o CNN) e si basa interamente su meccanismi di attenzione.

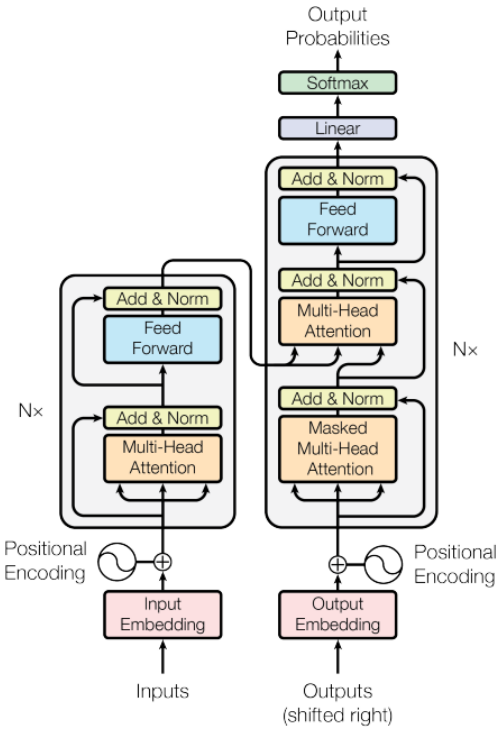


Figura 1.1: Schema dell'architettura del modello Transformer. La figura illustra le componenti dell'Encoder (a sinistra) e del Decoder (a destra). (Fonte: [17])

1.1.1 Le Innovazioni Chiave del Transformer

L'architettura proposta da Vaswani et al. introduce diversi concetti rivoluzionari che ne hanno decretato il successo. Le più significative sono:

Self-Attention È il cuore del modello e la sua innovazione più grande. A differenza delle architetture precedenti che processavano il testo parola per parola, la self-attention permette al modello di esaminare l'intera sequenza simultaneamente. Per ogni parola, questo meccanismo calcola un "punteggio di attenzione" che determina quanto "focus" dedicare a ogni altra parola presente nel testo. In questo modo, il modello costruisce una comprensione contestuale di un termine basandosi sulle relazioni che ha con tutte le altre parole, anche quelle molto distanti, catturando così le dipendenze globali in modo estremamente efficace.

Multi-Head Attention Invece di una singola funzione di attenzione, il Transformer impiega più "teste di attenzione" (attention heads) in parallelo. Ciò consente al modello di prestare attenzione congiuntamente alle informazioni provenienti da diversi sottospazi di rappresentazione in varie posizioni. Il modello utilizza 8 "head" parallele.

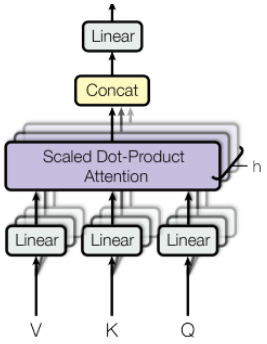


Figura 1.2: Rappresentazione della Multi-Head Attention. L'architettura calcola diverse proiezioni dell'input (teste di attenzione) in parallelo, permettendo al modello di catturare differenti tipi di relazioni contestuali. (Fonte: [17])

Positional Encoding Poiché il modello non ha una percezione intrinseca dell'ordine della sequenza, le informazioni posizionali vengono aggiunte agli embedding di input utilizzando funzioni seno e coseno. È interessante notare che gli embedding posizionali appresi hanno dato risultati quasi identici alla versione sinusoidale.

1.2 Document AI nell'era dei Transformer

Le innovazioni introdotte dall'architettura Transformer, ottenendo prestazioni superiori e riducendo i tempi di addestramento, non si sono limitate a migliorare i task classici del Natural Language Processing, ma hanno aperto la strada a una vera e propria rivoluzione in ambiti applicativi specifici. Uno dei settori che ha maggiormente beneficiato di queste nuove capacità è la Document AI.

1.2.1 Definizione e Compiti della Document AI

Cos'è: La Document AI, nota anche come Intelligent Document Processing (IDP) [3], è una branca dell'intelligenza artificiale che combina tecniche di Natural Language Processing (NLP) e Computer Vision per analizzare, comprendere ed elaborare automaticamente le informazioni contenute nei documenti [9]. L'obiettivo è trasformare dati non strutturati o semi-strutturati — presenti in formati come PDF, immagini, scansioni ed email [3] — in informazioni strutturate e immediatamente utilizzabili. Questo campo affronta la sfida di gestire la grande diversità di layout, formati e qualità dei documenti, che spaziano da fatture e ricevute a contratti legali e articoli scientifici [9].

Compiti Principali: I compiti principali della Document AI includono diverse operazioni fondamentali:

- **Estrazione di Informazioni (Information Extraction)**: Questo processo mira a identificare e estrarre dati specifici da un docu-

mento, come l'importo totale da una fattura, le date di scadenza in un contratto o i nomi degli autori in un articolo. Spesso, questo task è strettamente legato all'analisi del layout del documento, che interpreta le relazioni spaziali tra gli elementi per strutturarne il contenuto.[9]

- **Classificazione di Documenti (Document Classification)**: Consiste nell'assegnare automaticamente una categoria a un'immagine di documento. Ad esempio, un sistema può imparare a distinguere tra "articoli scientifici", "fatture", "ricevute" e altri tipi di documenti, indirizzando ciascuno verso il corretto flusso di lavoro aziendale. [9]

- **Risposta a Domande (Document Question Answering)**: Questo task avanzato permette di porre domande in linguaggio naturale relative al contenuto di un documento e di ottenere risposte precise, basate sulla comprensione logica del testo e del suo contesto visivo. [9]

Approcci Tradizionali: Storicamente, gli approcci tradizionali a questi problemi si basavano su pipeline complesse e rigide. Questi sistemi utilizzavano metodi basati su regole fisse, che richiedevano un notevole sforzo umano per essere creati e mantenuti [9, 3], o tecniche di machine learning classiche, come alberi decisionali e support vector machines [9]. Tali metodi, tuttavia, mostravano risultati insoddisfacenti, specialmente con documenti visivamente ricchi e complessi [9]. Tali metodi, tuttavia, mostravano risultati insoddisfacenti, specialmente con documenti visivamente ricchi e complessi [9]. L'analisi basata esclusivamente sull'informazione testuale, tipica anche dei primi modelli linguistici, si rivelava infatti insufficiente per una comprensione completa del documento, che dipende in modo cruciale anche da elementi come layout e immagini [9].

1.2.2 La Rivoluzione degli LLM nella Document AI

L'introduzione dei Large Language Models (LLM), basati sull'architettura Transformer [17], ha rappresentato un punto di svolta per il campo della Document AI, superando molti dei limiti degli approcci precedenti. Se i metodi basati su regole euristiche erano rigidi e costosi da mantenere [2], e quelli basati sul machine learning statistico mancavano di generalizzabilità [2, 3], gli LLM hanno introdotto un nuovo paradigma basato sulla comprensione profonda del contenuto e della struttura dei documenti.

- **Comprensione Semantica Profonda**: A differenza degli approcci tradizionali, che si basavano spesso su parole chiave o feature ingegnerizzate manualmente, i modelli basati su Transformer sono in grado di cogliere le relazioni contestuali e le sfumature semantiche del testo [2]. Tuttavia, l'applicazione dei primi LLM (come BERT [4]) all'analisi di documenti ha mostrato presto i suoi limiti; pur eccellendo nella comprensione del testo puro, la loro performance era insoddisfacente perché si basavano esclusivamente sull'informazione testuale [9], ignorando elementi cruciali come il layout, la formattazione e le immagini, che sono fondamentali per l'interpretazione completa di un documento [9, 2].

- **Flessibilità e Adattabilità**: La vera rivoluzione è avvenuta con l'introduzione di modelli multimodali pre-addestrati, con LayoutLM [2] come capostipite. Questi modelli estendono l'architettura dei Transformer per integrare, oltre agli embedding testuali, anche le informazioni sulla posizione 2D (layout) e visive (immagini) [2]. Questa fusione di modalità permette al modello di apprendere delle invarianze di layout (ad esempio, il fatto che una chiave e un valore sono spesso vicini) che sono comuni a diverse tipologie di documenti [2]. Di conseguenza, questi modelli sono intrinsecamente più flessibili e richiedono meno addestramento specifico per adattarsi a nuovi formati di documenti. L'evoluzione successiva, con LayoutLMv2 [2] e LayoutLMv3

[9], ha ulteriormente potenziato questa capacità integrando in modo ancora più profondo le feature visive e ottimizzando gli obiettivi di pre-addestramento per migliorare l'allineamento tra testo e immagine [2].

- **Semplificazione delle Pipeline**: In precedenza, un sistema di Document AI era tipicamente costituito da una pipeline complessa e multi-stadio che includeva pre-processing dell'immagine, Optical Character Recognition (OCR), classificazione del documento, estrazione delle entità e post-processing [3, 2]. Ogni passo era spesso gestito da un sistema separato, con conseguente aumento della complessità e dei punti di potenziale errore. L'avvento di modelli multimodali pre-addestrati e general-purpose ha permesso di semplificare drasticamente questo processo [2]. Un singolo modello come LayoutLMv3 può infatti gestire simultaneamente compiti che prima richiedevano sistemi distinti, come il riconoscimento di entità a livello di token (un task text-centric) e l'individuazione di aree del documento come tabelle o figure (un task image-centric) [9]. Questa unificazione non solo riduce la complessità architetturale e i costi di manutenzione, ma migliora anche le performance complessive, poiché il modello impara a sfruttare le sinergie tra le diverse modalità in un unico framework [2].

1.2.3 Nuove Sfide: Affidabilità e Specificità dei LLM

L'integrazione di testo, layout e informazioni visive ha dotato i modelli multimodali di una capacità di comprensione documentale senza precedenti. Tuttavia il loro utilizzo in task che vanno oltre la semplice estrazione di informazioni, come il loro utilizzo come fonte di conoscenza interattiva presenta un serie di sfide da affrontare legate a temi come affidabilità, specificità e sicurezza delle informazioni generate, limitandone quindi l'applicazione pratica in contesti sensibili. I problemi principali possono essere così sintetizzati:

- **Allucinazioni**: Le allucinazioni, nel contesto degli LLM, si riferiscono a situazioni nelle quali il modello genera dei contenuti basati su informazioni non accurate o errate. Le allucinazioni si manifestano quando il modello produce testo che include dettagli, fatti o informazioni fuorvianti o completamente inventati. Il problema deriva dalle istruzioni fornite ai modelli, ovvero fornire all'utente una risposta plausibile, basandosi su diversi pattern che ha imparato durante la fase di training, anche nei casi in cui la risposta generata non corrisponda al vero. Le allucinazioni non sono intenzionali, ma il risultato di una combinazioni di vari fattori, tra i quali mancanza di informazioni per rispondere correttamente, bias nei dati di addestramento e limitazioni del modello nel comprendere ciò che gli è stato richiesto. [15]

- **Conoscenza non Aggiornata e non Specifica**: I modelli linguistici pre-addestrati hanno una limitazione intrinseca, ovvero la loro conoscenza si basa esclusivamente sulle informazioni del loro addestramento, immagazinate nei parametri del modello e difficili da ampliare o revisionare. Questa limitazione rende i modelli statici e la loro conoscenza limitata alle informazioni disponibili al momento dell'addestramento, senza poter quindi accedere a informazioni più recenti[10]. Un altro problema derivante da tale memoria parametrica è la mancanza di specificità, il modello infatti non può accedere a dati personali e specifici, come ad esempio dei documenti di un'azienda o di un singolo individuo.

- **Mancanza di Tracciabilità**: I modelli parametrici non forniscono spiegazioni chiare per le risposte fornite all'utente, rendendo complicata la verifica delle fonti e richiedendo quindi uno sforzo considerevole per convalidare la risposta fornita dal modello. Questa limitazione assume particolare importanze in contesti in cui l'affidabilità e la precisione è fondamentale, come ad esempio in un contesto aziendale, legale o giornalistico. Architetture che utilizzano una memoria esterna basata su testo (memoria non-parametrica) offrono una forma di interpretabilità,

poiché la conoscenza a cui si accede può essere direttamente ispezionata dall'utente.

1.3 I Sistemi RAG: una Soluzione per l'Affidabilità e la Specificità

Una soluzione ai limiti della memoria puramente parametrica è stata proposta nel paper "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" di Lewis et al. (2020) [10]. Gli autori introducono un'architettura ibrida, denominata Retrieval-Augmented Generation (RAG), che integra la potenza dei modelli generativi con l'affidabilità di una base di conoscenza esterna.

1.3.1 Funzionamento Concettuale: l'Esame a Libro Aperto

Per comprendere il principio alla base dei RAG system, si può ricorrere a un'efficace analogia: quella dell'esame a libro aperto. Un LLM tradizionale è come uno studente che deve rispondere basandosi unicamente sulla propria memoria; per quanto preparato, può dimenticare dettagli, confondere concetti o, messo alle strette, "inventare" una risposta (l'allucinazione).

Il sistema RAG, invece, trasforma questo scenario. Fornisce al modello (lo studente) l'accesso a una fonte di conoscenza esterna e autorevole (il libro di testo, in questo caso un insieme di documenti specifici). L'obiettivo non è più testare la memoria, ma la capacità di trovare l'informazione pertinente all'interno della fonte e di utilizzarla per formulare una risposta coerente e fattuale. In questo modo, il RAG non si limita a combinare conoscenza pregressa e specifica, ma istituisce un processo dinamico di recupero e generazione che mitiga direttamente i problemi di affidabilità e attualità discussi in precedenza.

1.3.2 Recupero e Generazione: le Due Fasi del RAG

Il funzionamento di un sistema RAG si articola in due stadi principali che lavorano in sequenza per rispondere a una richiesta dell'utente:

- **Fase 1: Recupero (Retrieval)** Quando l'utente fa una domanda, il sistema non la invia direttamente modello linguistico, ma cerca prima all'interno della propria collezione di documenti le informazioni e i passaggi rilevanti per poter rispondere correttamente.

- **Fase 2: Generazione Aumentata (Augmented Generation)** Il sistema prende i frammenti di documento che ha trovato, li unisce alla domanda e chiede al modello di generare una risposta basata esclusivamente sulle informazioni fornite.

1.3.3 Benefici Chiave: Affidabilità, Specificità e Tracciabilità

L'adozione di un'architettura ibrida come quella RAG, che disaccoppia il recupero dell'informazione dalla generazione del testo, si traduce in una serie di vantaggi diretti che ne definiscono il valore applicativo:

- **Affidabilità:** Le risposte sono ancorate a un contesto reale fornito dall'utente, riducendo le allucinazioni.

- **Specificità:** Il sistema può rispondere su documenti che non ha mai visto durante il suo addestramento, ad esempio se applicato in un contesto aziendale potrebbe rispondere a delle domande inerenti a dei documenti specifici dell'azienda non reperibili online.

- **Tracciabilità:** È possibile mostrare all'utente quali fonti sono state usate per la risposta, risolvendo il problema della mancanza di tracciabilità.

1.4 Analisi della Letteratura e Motivazioni della Tesi

L'analisi dello stato dell'arte evidenzia una traiettoria chiara: la tecnologia RAG si è affermata come la soluzione de facto per rendere i Large Language Models affidabili e contestualmente specifici, specialmente nel dominio della Document AI. Questo ha portato alla nascita di applicazioni commerciali che applicano tale principio, spaziando da strumenti di produttività generica a soluzioni di nicchia.

Tuttavia, applicando questa analisi al contesto specifico dello studio e dell'apprendimento, emerge un "gap" significativo. Le soluzioni esistenti per studenti tendono a essere frammentate: da un lato, ci sono eccellenti strumenti RAG che permettono di "dialogare" con un singolo documento (es. ChatPDF[14]); dall'altro, piattaforme di produttività (es. Notion AI[13]) che offrono funzionalità di IA su un intero workspace, ma senza un flusso di lavoro ottimizzato per lo studio. Mancano, di fatto, soluzioni che propongano un **anello di studio integrato**: un ecosistema in cui lo studente possa non solo interrogare i propri materiali organizzati per materia, ma anche generare dinamicamente test di autovalutazione, ricevere feedback attivi sulle proprie risposte (anche quelle aperte) e monitorare i progressi in modo mirato.

È proprio in questo spazio che si inserisce il presente lavoro di tesi. La motivazione principale è quella di progettare e sviluppare un'applicazione mobile nativa, "MyStudyApp", che colmi tale lacuna. L'obiettivo è creare un compagno di studi che sfrutti un'architettura RAG non come singola funzionalità, ma come motore di un ciclo di apprendimento completo: dalla gestione focalizzata dei documenti, alla chat contestuale, fino alla generazione di quiz avanzati e alla valutazione attiva delle performance. La tesi intende dimostrare come tale approccio integrato possa offrire un'esperienza di studio più efficace, personalizzata e coinvolgente rispetto agli strumenti frammentati attualmente disponibili sul mercato.

Capitolo 2

Background Tecnico sui Sistemi Retrieval-Augmented Generation (RAG)

Dopo aver introdotto nel capitolo precedente i sistemi RAG come soluzione efficace alle sfide di affidabilità e specificità dei LLM, questo capitolo ne approfondisce l'architettura e il funzionamento. Verranno analizzati in dettaglio i componenti e i processi che costituiscono una pipeline RAG, fornendo le basi tecniche necessarie a comprendere le scelte implementative della presente tesi.

2.1 L'Architettura di una Pipeline RAG

Per comprendere il funzionamento di un sistema RAG, è essenziale analizzarne l'architettura, che può essere suddivisa in due macro-fasi logiche e operative. Un'efficace rappresentazione visiva di questa struttura è fornita da un diagramma che illustra una pipeline RAG moderna (Figura 2.1), evidenziando il flusso dei dati sia nella fase di preparazione della conoscenza sia in quella di interrogazione da parte dell'utente.

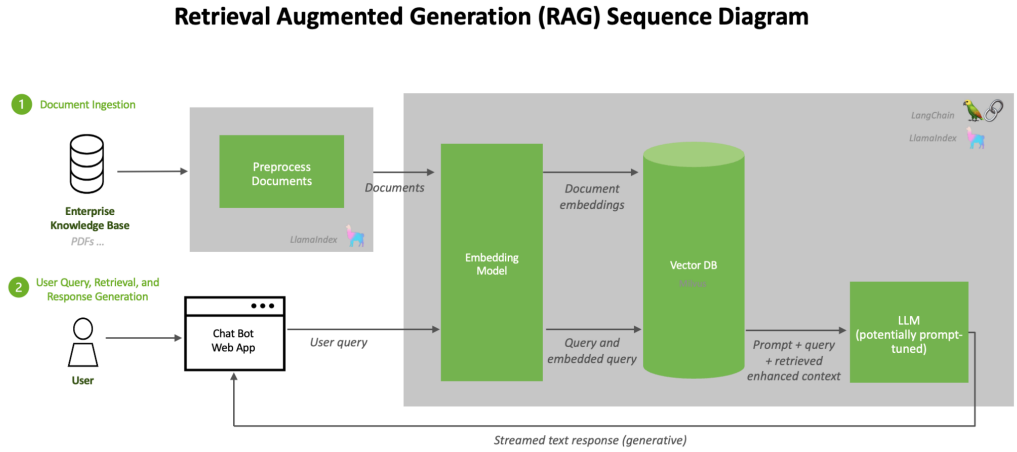


Figura 2.1: Schema di una pipeline RAG moderna che evidenzia le fasi di ingestione dei documenti e di interrogazione dell'utente. (Fonte: Adattato da [18])

Come illustra lo schema, la pipeline è nettamente separata in due flussi operativi principali:

Fase di Ingestione dei Documenti (Document Ingestion) Questa fase, eseguita *offline*, ha lo scopo di preparare la base di conoscenza esterna. I documenti (es. PDF, testi) vengono prima pre-processati e poi trasformati in rappresentazioni numeriche (embedding) tramite un apposito modello. Questi embedding vengono infine memorizzati e indicizzati in un database vettoriale (Vector DB), rendendoli pronti per una ricerca semantica efficiente. Questa fase corrisponde a ciò che definiremo **Fase di Indicizzazione**.

Fase di Interrogazione (User Query, Retrieval, and Response Generation) Questo processo, eseguito *online*, si attiva quando un utente interagisce con il sistema. La domanda dell'utente viene anch'essa trasformata in un embedding e usata per interrogare il Vector DB, che restituisce i frammenti di testo (contesto) più pertinenti. Questo contesto, unito alla domanda originale, viene inserito in un prompt e inviato al Large

Language Model (LLM), che genera la risposta finale basandosi sulle informazioni fornite. Questa sequenza di operazioni corrisponde a ciò che definiremo **Fase di Interrogazione**.

Le sezioni seguenti analizzeranno in dettaglio i passaggi tecnici che compongono ciascuna di queste due fasi.

2.2 Fase 1: Indicizzazione della Conoscenza

I documenti caricati come conoscenza esterna, non possono essere interrogati direttamente dal modello, ma devono prima essere processati e memorizzati in un formato ottimizzato per il recupero delle informazioni.

2.2.1 Caricamento e Suddivisione dei Documenti (Document Loading and Chunking)

Inizialmente i dati originali vengono ripuliti e ne viene estratto il contenuto, diversi formati di file come PDF, HTML, Word ecc. vengono convertiti in del testo semplice. Il testo ottenuto dalla fase di pulizia verrà poi suddiviso in dei "chunk" più piccoli, ciò avviene perchè gli LLM hanno una lunghezza di contesto limitata, per tale motivo è necessario creare dei chunk il più brevi possibile, in modo da passare al modello solo le informazioni necessarie e non sovraccaricarlo con informazioni non utili allo svolgimento della task che gli è stata assegnata[6]. La dimensione dei chunk è una scelta fondamentale durante lo sviluppo dei sistemi RAG, chunk troppo lunghi possono contenere troppe informazioni e causare del "rumore" che degrada le prestazioni del modello, mentre chunk troppo corti possono invece risultare troppo specifici e quindi difficili da correlare con altre informazioni. Esistono diverse strategie di chunking, ognuna con i propri vantaggi e svantaggi, la cui scelta dipende dalle caratteristiche dei dati, dal modello di embedding e dall'applicazione finale. Le più comuni includono:

Suddivisione a Dimensione Fissa (Fixed-Size Chunking) È l'approccio più semplice, che consiste nel dividere il testo in blocchi di una dimensione predefinita (es. un certo numero di token [16] o caratteri). Per mitigare la perdita di contesto ai bordi di ogni chunk, si può utilizzare una tecnica di **sovrapposizione (overlap)**, dove una porzione di testo viene condivisa tra la fine di un chunk e l'inizio del successivo [6]. Un esempio pratico di questa tecnica è la suddivisione di un testo in passaggi di 100 parole, avendo cura di non spezzare le frasi a metà [1].

Suddivisione per Frase (Sentence Splitting) Questa strategia utilizza i confini delle frasi come punto di divisione naturale. Questo approccio garantisce che ogni chunk abbia un senso compiuto, a differenza della suddivisione a dimensione fissa che può spezzare una frase a metà. Tuttavia, le frasi possono avere lunghezze molto variabili e, a volte, non essere auto-contenute, mancando del contesto necessario per interpretare pronomi o riferimenti [1].

Suddivisione Ricorsiva (Recursive Chunking) È una tecnica più avanzata che tenta di mantenere la coerenza semantica. Funziona dividendo il testo in modo gerarchico, utilizzando una lista predefinita di separatori (ad esempio, prima cerca di dividere per paragrafo, poi per frase, poi per spazio). Il processo si ripete ricorsivamente sui pezzi ottenuti finché non raggiungono la dimensione desiderata. Questo aiuta a mantenere uniti i blocchi di testo semanticamente correlati.

Suddivisione Semantica per Proposizioni Un approccio innovativo mira a superare i limiti delle strategie precedenti, suddividendo il testo in "proposizioni". Una proposizione è definita come un'**espressione atomica** all'interno del testo, che incapsula un singolo fatto in un formato conciso e auto-contenuto [1]. L'obiettivo è creare chunk che abbiano un'alta densità di informazione rilevante, eliminando dettagli superflui che potrebbero "distrarre" sia il retriever che il LLM [1]. Gli esperi-

menti dimostrano che indicizzare un corpus per proposizioni migliora significativamente le performance sia del recupero di informazioni che dei task di Question Answering a valle, rispetto all'uso di passaggi o frasi [1].

La scelta della giusta granularità per il chunking è quindi un compromesso fondamentale: unità più piccole come le proposizioni offrono maggiore precisione, mentre unità più grandi come i passaggi forniscono un contesto più ampio, ma a rischio di introdurre rumore.[1]

2.2.2 Creazione degli Embedding (Embedding Generation)

Una volta che il testo è stato suddiviso in chunk, è necessario trasformare questi frammenti di testo in un formato che la macchina possa comprendere e confrontare. Questo processo è al **cuore della ricerca semantica** e si realizza tramite la creazione di **embedding**.

Un embedding di testo è una **rappresentazione vettoriale numerica** di un pezzo di testo (in questo caso, un chunk) in uno spazio multidimensionale [6]. L'obiettivo di un buon modello di embedding è fare in modo che chunk con significati semanticamente simili abbiano vettori vicini in questo spazio, mentre chunk con significati diversi siano distanti. Questa proprietà è ciò che permette al sistema di "capire" quale frammento di documento risponde meglio a una domanda, semplicemente cercando i vettori dei chunk più vicini al vettore della domanda tramite il calcolo della similarità [6].

Per ottenere queste rappresentazioni, si utilizzano modelli di embedding pre-addestrati su vasta scala [6]. Sebbene esistano molti modelli, alcuni sono diventati dei punti di riferimento nel campo dei sistemi RAG grazie alla loro efficacia:

- **Modelli basati su architettura Transformer**: Molti dei retriever più performanti si basano su modelli Transformer, come BERT, su cui

viene effettuato del fine-tuning per il compito specifico di produrre embedding di alta qualità per la ricerca di similarità. Esempi noti includono **DPR (Dense Passage Retriever)**, **Contriever** e **GTR (Generalizable T5-based Retriever)** [1]. Questi modelli sono progettati per codificare sia le domande (query) sia i passaggi di testo (document) in vettori densi che possono essere confrontati efficacemente.

- **Modelli commerciali ad alte prestazioni**: Aziende come OpenAI offrono modelli di embedding molto potenti, come `text-embedding-ada-002`, che sono ampiamente utilizzati per la loro capacità di catturare sfumature semantiche complesse [6]. Questi modelli, basati anch'essi su architetture Transformer, sono una scelta comune per la loro facilità d'uso e le performance elevate.

- **Modelli open-source ottimizzati**: La comunità open-source ha prodotto modelli estremamente competitivi, spesso ottimizzati per specifici domini o task. Un esempio è il modello **BGE (BAAI General Embedding)**, noto per le sue eccellenti performance e per la possibilità di essere ulteriormente fine-tuned su dati specifici per migliorare la rappresentazione semantica in contesti di nicchia [6].

La scelta del modello di embedding è cruciale, poiché la qualità dei vettori generati determina direttamente l'efficacia della fase di recupero delle informazioni [6].

2.2.3 Memorizzazione in un Vector Store

Dopo aver generato gli embedding numerici per ogni chunk di testo, il passo finale della fase di indicizzazione è la loro memorizzazione e organizzazione in una struttura ottimizzata per la ricerca: un **Vector Store**, noto anche come **database vettoriale**.

Un Vector Store è un tipo di database specializzato, progettato per archiviare e gestire in modo efficiente vettori ad alta dimensionalità [7]. Le sue

due funzioni principali sono l'archiviazione dei vettori e, soprattutto, il loro recupero tramite ricerca per similarità [7].

Eseguire una ricerca per similarità esatta (exhaustive search), che calcola la distanza tra la query e ogni singolo vettore nel database, è impraticabile su dataset di grandi dimensioni a causa della "maledizione della dimensionalità" [8, 11]. Per questo motivo, i vector store moderni si affidano a tecniche di **Ricerca Approssimata del Vicino più Prossimo (Approximate Nearest Neighbor - ANN)** [7]. Questi algoritmi sacrificano una piccolissima e spesso trascurabile percentuale di accuratezza per ottenere un'enorme accelerazione nei tempi di ricerca [11, 7].

Le strategie ANN più efficaci, implementate nelle moderne tecnologie di indicizzazione, includono:

- **HNSW (Hierarchical Navigable Small World)**: È un approccio basato su grafi che si è affermato come uno dei più performanti [11]. Costruisce una struttura gerarchica multi-livello composta da grafi di prossimità, dove i livelli superiori contengono i link a lunga distanza e i livelli inferiori quelli a breve distanza [11]. La ricerca inizia dallo strato più alto, attraversando velocemente lo spazio vettoriale, per poi scendere progressivamente ai livelli più dettagliati, raggiungendo una complessità di ricerca logaritmica [11].

- **Product Quantization (PQ) e FAISS**: La quantizzazione del prodotto (PQ) è una tecnica per comprimere vettori ad alta dimensionalità in codici più corti, riducendo drasticamente l'uso della memoria e accelerando il calcolo delle distanze [8]. La libreria open-source **FAISS (Facebook AI Similarity Search)**, sviluppata da Meta, è uno strumento di riferimento che implementa in modo estremamente efficiente la ricerca per similarità basata su PQ [8]. Spesso la PQ viene combinata con una tecnica di partizionamento chiamata **Inverted File (IVF)**, dove i vettori vengono prima raggruppati in cluster e la ricerca viene limitata solo ai cluster più vicini alla query, riducendo ulteriormente lo spazio di ricerca [8].

Questi algoritmi e librerie costituiscono le fondamenta di molti vector database moderni. Soluzioni open-source come **ChromaDB** o servizi cloud gestiti come **Pinecone** utilizzano implementazioni di HNSW, FAISS o algoritmi simili per offrire i loro servizi di archiviazione e ricerca vettoriale scalabile [7].

2.3 Fase 2: Interrogazione (Recupero e Generazione)

Questa fase descrive il flusso di lavoro che si attiva in tempo reale nel momento in cui un utente interagisce con il sistema ponendo una domanda. Si articola in due passaggi sequenziali: il recupero delle informazioni pertinenti (Retrieval) e la generazione della risposta basata su di esse (Augmented Generation) [6].

2.3.1 Recupero (Retrieval)

Lo scopo di questa fase è identificare e recuperare i frammenti di conoscenza più rilevanti per la richiesta dell'utente dalla base di conoscenza indicizzata. Il processo si svolge come segue:

1. **Creazione dell'Embedding della Query**: L'input dell'utente (query) viene processato attraverso lo stesso modello di embedding utilizzato durante la fase di indicizzazione [6]. Questo passaggio è fondamentale per garantire che la domanda e i documenti siano rappresentati nello stesso spazio semantico, rendendo possibile un confronto significativo.

2. **Interrogazione del Vector Store**: Il vettore della query viene inviato al Vector Store, che esegue una ricerca per similarità per identificare i "k" chunk di testo i cui embedding sono più vicini a quello della query [10]. La "vicinanza" viene misurata tramite una funzione di distanza, come la similarità cosenica o il prodotto scalare (Maximum

Inner Product Search - MIPS) [10]. Per garantire una risposta rapida anche su miliardi di vettori, questa ricerca non è esaustiva, ma si affida ad algoritmi ANN come HNSW [11].

3. **Costruzione del Contesto**: I k chunk di testo originali, corrispondenti ai vettori più simili, vengono recuperati. L'insieme di questi frammenti costituisce il **contesto** ("context") che verrà fornito come conoscenza supplementare al Large Language Model nella fase successiva [10].

2.3.2 Generazione Aumentata (Augmented Generation)

Una volta recuperato il contesto pertinente, il sistema passa alla generazione della risposta finale [6].

1. **Costruzione del Prompt**: Il contesto recuperato non viene utilizzato da solo. Viene strategicamente assemblato insieme alla domanda originale dell'utente all'interno di un **prompt** strutturato. Questo processo, noto come *prompt engineering*[12], guida il modello a utilizzare esclusivamente le informazioni fornite. Un template di prompt comune ha la seguente forma:

   Basandoti sul seguente contesto, rispondi alla domanda. Non utilizzare alcuna conoscenza pregressa.

   Contesto: "[Contesto recuperato dal Vector Store]"

   Domanda: "[Domanda originale dell'utente]"

2. **Generazione della Risposta Finale**: Il prompt completo e "aumentato" viene inviato al Large Language Model[10]. Il modello esegue il suo compito di generazione del linguaggio, ma con un vincolo fondamentale: deve formulare la risposta basandosi sulle informazioni

presenti nel contesto fornito. Questo approccio, definito *grounding* (ancoraggio), riduce drasticamente il rischio di allucinazioni, migliora l'accuratezza fattuale delle risposte [6] e garantisce la tracciabilità, poiché ogni affermazione può essere ricondotta a uno specifico frammento di testo della fonte originale [6].

2.3.3 Conclusione del Capitolo

In questo capitolo abbiamo analizzato nel dettaglio la meccanica interna di un sistema di Retrieval-Augmented Generation. Abbiamo visto come la pipeline si articoli in due fasi distinte: una prima fase di **Indicizzazione**, durante la quale la conoscenza esterna viene processata tramite *chunking*, trasformata in vettori numerici attraverso modelli di *embedding* e infine archiviata in un *Vector Store* ottimizzato per la ricerca. A questa segue la fase di **Interrogazione**, che comprende il *recupero* (retrieval) dei chunk più pertinenti in risposta a una query dell'utente e la *generazione* (generation) di una risposta finale da parte del Large Language Model, aumentata con il contesto recuperato [6].

Questa architettura ibrida, che combina la memoria parametrica del LLM con una memoria non-parametrica esterna [10], risponde direttamente alle sfide teoriche delineate nel Capitolo 1. Ancorando la generazione a un contesto specifico e fattuale, si mitiga drasticamente il problema delle **allucinazioni** [6]. La natura esterna e modulare della base di conoscenza permette di affrontare il limite della **conoscenza non aggiornata**, poiché i documenti possono essere aggiornati o sostituiti senza la necessità di riaddestrare l'intero modello linguistico [10]. Infine, la possibilità di risalire ai frammenti di testo specifici utilizzati per formulare una risposta risolve il problema della **mancanza di tracciabilità**, aumentando l'interpretabilità e la fiducia nel sistema [6].

Stabilite le fondamenta tecniche dei sistemi RAG, l'analisi può ora spostarsi dalla teoria alla pratica. Il capitolo seguente sarà dedicato all'applicazione di questi concetti nel progetto e nell'architettura di "MyStudyApp".

Capitolo 3

Progetto e Architettura dell'Applicazione

Dopo aver analizzato nel capitolo precedente i fondamenti teorici e tecnici dei sistemi di Retrieval-Augmented Generation, questo capitolo descrive come tali principi siano stati applicati per progettare e realizzare l'applicazione mobile "MyStudyApp". Verrà presentata la visione architetturale d'insieme del sistema, illustrando i componenti principali, le tecnologie scelte e l'infrastruttura cloud su cui si basa. Successivamente, verrà descritto il flusso di navigazione che caratterizza l'esperienza utente. L'analisi si manterrà a un livello alto, rimandando i dettagli implementativi specifici al capitolo successivo.

Per fornire una visione chiara e immediata delle funzionalità offerte dall'applicazione, si introduce un diagramma dei casi d'uso (Figura 3.1). Questo diagramma illustra le interazioni principali tra l'attore (l'Utente) e il sistema, evidenziando gli obiettivi che l'utente può raggiungere attraverso l'app.

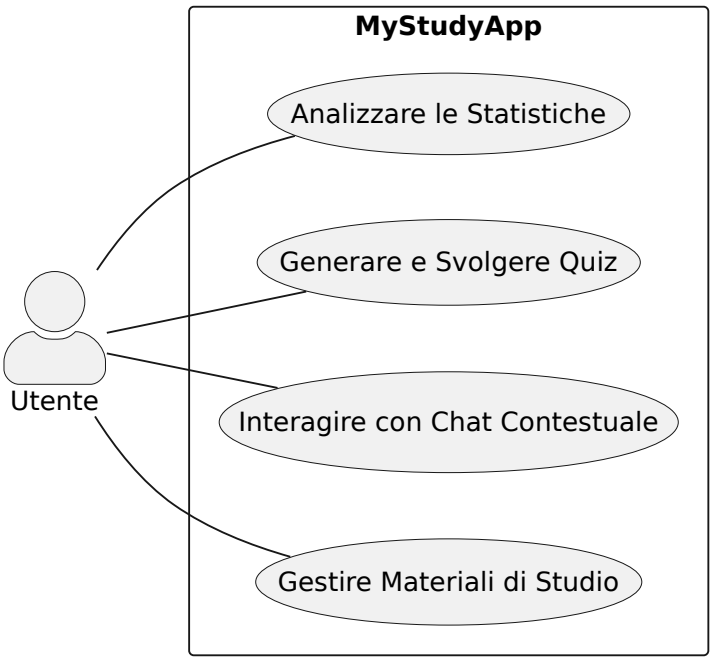


Figura 3.1: Diagramma dei casi d'uso principali per l'applicazione MyStudyApp. Vengono mostrate le interazioni chiave dell'Utente con il sistema.

Come mostrato nel diagramma, l'Utente è l'attore centrale che può eseguire le seguenti azioni principali:

- **Gestire Materiali di Studio**: L'utente può organizzare i propri documenti creando categorie tematiche e caricando al loro interno i file di studio (PDF, DOCX, TXT).

- **Interagire con Chat Contestuale**: L'utente può avviare una conversazione con un chatbot basato su una specifica categoria, ricevendo risposte basate esclusivamente su quel contesto.

- **Generare e Svolgere Quiz**: L'utente ha la facoltà di creare test di autovalutazione personalizzati, scegliendo la materia, il numero e la tipologia di domande.

- **Analizzare le Statistiche**: L'utente può monitorare i propri progressi attraverso una dashboard che mostra il punteggio medio e l'andamento delle performance.

3.1 Visione Architetturale

L'architettura di MyStudyApp è stata progettata per essere robusta, scalabile e sicura, seguendo un moderno approccio basato su servizi cloud. La struttura si fonda su un modello client-server che disaccoppia nettamente l'interfaccia utente dalla logica di elaborazione dei dati.

3.1.1 Filosofia di Progetto e Componenti Principali

L'architettura di MyStudyApp si basa su un modello client-server, questa scelta progettuale consente di separare la logica di presentazione, gestita dal client, da quella di elaborazione, delegata al server. Questo approccio garantisce che l'applicazione rimanga snella e reattiva, affidando le operazione computazionalmente più complesse, come l'esecuzione della pipeline RAG, a un backend più potente. La comunicazione tra le due componenti avviene tra un'interfaccia basata su API REST.

Frontend Il client è un'applicazione nativa per Android, sviluppata in Kotlin. È responsabile dell'intera esperienza utente, gestendo l'interfaccia grafica e la navigazione. Per garantire una struttura del codice robusta, il frontend è stato progettato seguendo il pattern architetturale MVVM (Model-View-ViewModel).

Backend Il server agisce come il nucleo computazionale del sistema. È stato implementato in Python utilizzando il framework FastAPI. Gestisce tutta la logica di business, esponendo una serie di endpoint REST e orchestrando l'intera pipeline RAG.

3.1.2 Architettura Cloud e Servizi Distribuiti

La funzionalità del backend è realizzata attraverso un'architettura a servizi distribuiti, basata interamente sulla piattaforma cloud Microsoft Azure. Anziché implementare ogni componente da zero, sono stati scelti specifici servizi PaaS (Platform-as-a-Service) per ogni compito, consentendo di concentrarsi sulla logica applicativa e di beneficiare della scalabilità, sicurezza e manutenzione offerte dalla piattaforma. I servizi chiave che compongono l'architettura sono i seguenti:

Azure Container Apps Questo servizio serverless è stato scelto per l'hosting e l'esecuzione dell'applicazione backend containerizzata (Docker). La sua capacità di scalare le risorse in base al carico, inclusa la possibilità di scalare a zero per ottimizzare i costi, lo ha reso ideale per il contesto del progetto.

Azure Blob Storage Costituisce la soluzione di archiviazione per tutti i file documentali (PDF, DOCX, TXT) caricati dagli utenti. Offre un sistema di storage di oggetti sicuro, duraturo e altamente scalabile, disaccoppiando l'archiviazione dei file dalla logica computazionale dell'applicazione.

Azure AI Search Rappresenta il cuore della fase di *retrieval* della pipeline RAG. Questo servizio viene utilizzato per creare un indice di ricerca avanzato che memorizza sia il testo dei "chunk" estratti dai documenti, sia i loro corrispondenti embedding vettoriali. La sua funzione è quella di eseguire ricerche semantiche e vettoriali ad alte prestazioni per recuperare le informazioni più pertinenti in risposta a una query dell'utente.

Azure OpenAI Service È il motore della fase di *generation* e di vettorizzazione del testo. Fornisce l'accesso via API a modelli linguistici di grandi dimensioni per due compiti fondamentali:

- La generazione di embedding vettoriali dal testo, utilizzando modelli come `text-embedding-ada-002`.
- La generazione di testo complesso e coerente per le risposte della chat e la creazione delle domande dei quiz, utilizzando modelli potenti come `gpt-4o`.

L'interazione tra questi servizi è illustrata schematicamente nella Figura 3.2, che mostra il flusso di dati e comunicazioni dall'applicazione client fino ai singoli componenti dell'infrastruttura cloud.

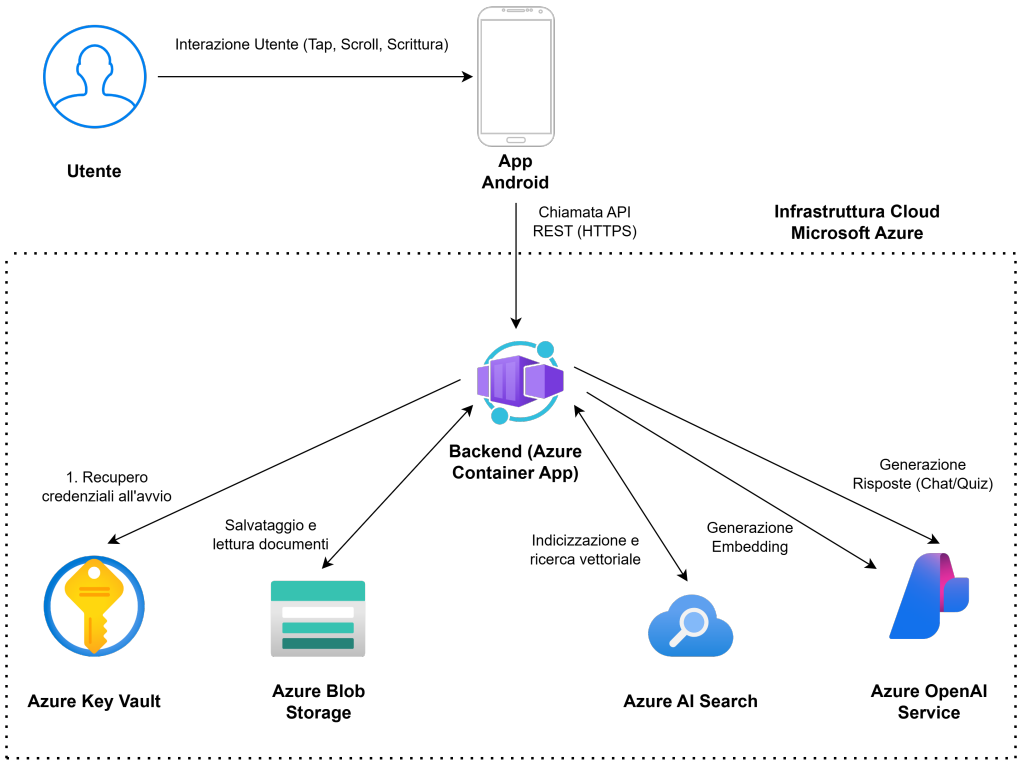


Figura 3.2: Schema dell'architettura cloud su Microsoft Azure. Vengono mostrati i flussi di comunicazione tra il client Android, l'applicazione backend su Container Apps e i servizi PaaS di supporto.

3.1.3 Sicurezza del Backend: Gestione delle Credenziali

Un aspetto fondamentale nella progettazione di qualsiasi applicazione connessa a servizi esterni è la gestione sicura delle credenziali. Seguendo le best practice di sicurezza, che sconsigliano di memorizzare informazioni sensibili come le chiavi API direttamente nel codice sorgente, è stata implementata una soluzione robusta basata su due servizi chiave della piattaforma Azure: Azure Key Vault e le Identità Gestite (Managed Identities).

Il Ruolo di Azure Key Vault: La Cassaforte dei Segreti Azure Key Vault funge da repository sicuro e centralizzato per tutte le informazioni sensibili del progetto. Invece di essere disperse in file di configurazione o variabili d'ambiente, le credenziali necessarie per comunicare con i servizi di IA e di archiviazione sono custodite all'interno del Key Vault, protette da policy di accesso granulari.

Il Ruolo dell'Identità Gestita: L'Autenticazione senza Password Per permettere all'applicazione backend di accedere in modo sicuro al Key Vault, si sfrutta la funzionalità delle Identità Gestite. All'applicazione ospitata su Azure Container Apps viene assegnata un'identità univoca all'interno di Azure Active Directory. In fase di configurazione, a questa identità viene concesso il permesso esplicito di leggere i segreti dal Key Vault. Questo meccanismo elimina la necessità di gestire credenziali per l'applicazione stessa, poiché la sua autenticazione è gestita nativamente dalla piattaforma.

Il flusso di autenticazione all'avvio del backend è quindi il seguente: l'applicazione, tramite la classe DefaultAzureCredential dell'SDK di Azure, utilizza la sua Identità Gestita per stabilire una connessione sicura con Azure Key Vault. Successivamente, richiede e ottiene i segreti necessari per operare. Solo dopo aver recuperato queste credenziali in memoria, inizializza i client per gli altri servizi esterni. Questo approccio garantisce che nessuna chiave

API o altra informazione sensibile sia mai esposta nel codice sorgente o nelle variabili d'ambiente del container, aderendo pienamente al principio della gestione sicura dei segreti.

3.1.4 Sicurezza sul Client: Iniezione Dinamica della API Key

Oltre alla sicurezza lato server, è stata posta attenzione anche alla gestione sicura dell'autenticazione dal client verso il backend. Poiché ogni chiamata API richiede un'intestazione di autenticazione (X-API-Key) per essere accettata dal server, si è reso necessario un meccanismo per aggiungere questa informazione in modo automatico e centralizzato. Per raggiungere questo obiettivo, è stato impiegato il pattern Interceptor, una funzionalità della libreria OkHttp su cui si basa Retrofit.

Per centralizzare la gestione della chiave API, è stato aggiunto un interceptor direttamente nella configurazione del client HTTP. Lo scopo di questo interceptor è intercettare ogni richiesta prima che venga eseguita per aggiungere dinamicamente l'intestazione X-API-Key con la relativa chiave segreta. Grazie a questo approccio, il resto dell'applicazione (ad esempio, i Repository) non deve essere a conoscenza dei dettagli di autenticazione. La chiave API stessa è gestita al di fuori del codice sorgente, memorizzata in file di configurazione sicuri di Gradle e acceduta tramite la classe BuildConfig, prevenendone l'esposizione accidentale.

Questo approccio offre due vantaggi significativi. In primo luogo, centralizza la logica di autenticazione: qualsiasi modifica alla chiave o al meccanismo di autorizzazione richiede un intervento in un solo punto del codice. In secondo luogo, promuove un codice più pulito e disaccoppiato, poiché i componenti responsabili del recupero dei dati non devono preoccuparsi dei dettagli implementativi dell'autenticazione, ma solo di eseguire le chiamate di rete necessarie.

3.1.5 Stack Tecnologico

Di seguito viene presentato un elenco riassuntivo delle principali tecnologie, librerie e servizi che compongono lo stack tecnologico dell'applicazione MyStudyApp.

- **Frontend (Android)**
  - Linguaggio: **Kotlin**
  - Architettura: **MVVM (Model-View-ViewModel)**
  - Componenti Jetpack: **Room** (database locale), **Navigation Component** (navigazione), **ViewModel**, **Lifecycle**, **WorkManager** (task in background)
  - Networking: **Retrofit** e **Gson** per la comunicazione API REST
  - UI: **Material Components**, **ViewBinding**
  - Grafici: **MPAndroidChart** per la visualizzazione delle statistiche
- **Backend**
  - Linguaggio e Framework: **Python** con **FastAPI**
  - Server: **Uvicorn**
  - Validazione Dati: **Pydantic**
  - Librerie Chiave: SDK di Azure, PyPDF2, python-docx, Tiktoken
- **Piattaforma Cloud**
  - Provider: **Microsoft Azure**
  - Servizi Utilizzati: Azure Container Apps, Azure Blob Storage, Azure AI Search, Azure OpenAI Service, Azure Key Vault

3.2 Flusso di Navigazione e User Experience

Dopo aver descritto l'architettura tecnica del sistema, questa sezione sposta l'attenzione sull'applicazione dal punto di vista dell'utente finale. Verrà illustrato il flusso di navigazione attraverso le funzionalità principali, seguendo un percorso logico che un utente compirebbe per utilizzare l'app. Questo percorso guidato (walkthrough), supportato da schermate dell'interfaccia, mira a fornire una comprensione pratica del flusso di interazione, mostrando come le scelte architetturali descritte in precedenza si concretizzino nell'esperienza utente finale.

3.2.1 Gestione dei Materiali di Studio

Per poter interagire con le varie funzionalità dell'applicazione, l'utente deve prima caricare e organizzare i propri documenti di studio in "contenitori" logici che nell'app vengono definiti "Categorie". Questa organizzazione è fondamentale per poi poter interagire con i documenti in modo contestuale.

Creazione di una nuova categoria

L'utente può accedere alla sezione "Categorie" dalla barra di navigazione principale, qui verranno visualizzate le categorie già esistenti e un Floating Action Button (FAB) per la creazione di una nuova categoria. Al click dell'utente sul FAB si aprirà una schermata di dialogo dove inserire il nome della categoria, una volta confermata la creazione della categoria sarà poi possibile interagire con essa.

Caricamento di un Documento

Dopo aver creato una categoria, l'utente vi accede con un tap dalla schermata precedentemente menzionata, entrando nella pagina di dettaglio di una singola categoria. Inizialmente la pagina mostrerà un indicatore di caricamento per mostrare un feedback visivo mentre vengono recuperati i documenti associati alla categoria dal backend. Al termine del caricamento verranno

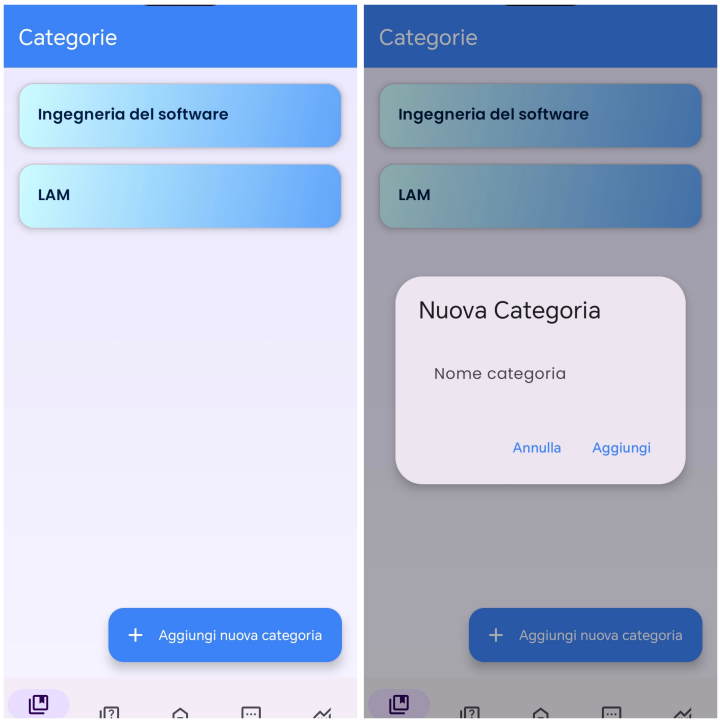


Figura 3.3: Flusso utente per la creazione di una nuova categoria. L'immagine mostra sia la schermata con l'elenco delle categorie sia la finestra di dialogo per l'inserimento del nome.

mostrati i documenti già caricati all'interno della categoria con un'icona specifica per indicare il formato del file e un'opzione per rimuovere uno specifico file dalla categoria. Sul fondo della pagina è presente un FAB per l'upload di un nuovo documento, cliccando sul pulsante verrà aperto il FilePicker di sistema che consentirà di selezionare dei documenti in formato PDF, TXT, DOCX da aggiungere alla categoria.

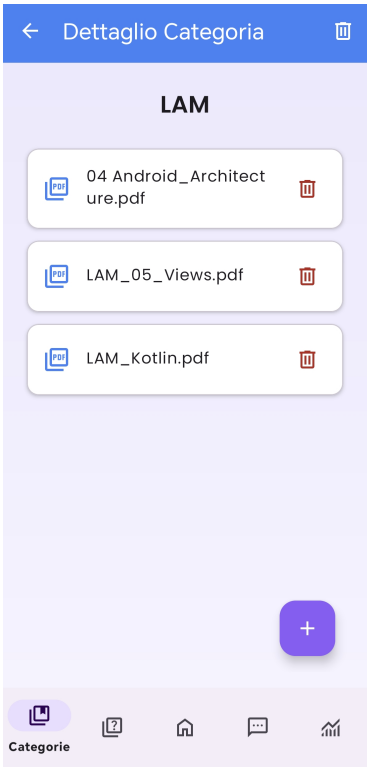


Figura 3.4: Schermata di dettaglio di una singola categoria.

3.2.2 Anello di Studio: Chat Contestuale

Dopo aver caricato i propri documenti, l'utente può sfruttare la funzonalità di Q&A. Attraverso questa funzionalità sarà possibile interrogare il chatbot interloquendo in linguaggio naturale, il bot risponderà basandosi sui documenti specificati per quella determinata categoria, sfruttando appieno le funzionalità del sistema RAG.

Flusso di Avvio di una Conversazione

Attraverso il menù di navigazione è possibile accedere alla sezione "Chat Q&A" dove sarà mostrata la lista delle chat esistenti e un pulsante per avviare una nuova conversazione. Al tap dell'utente sul FAB non verrà avviata

direttamente la conversazione ma si arriverà in una schermata intermedia dove l'utente selezionerà la materia sulla quale avviare la conversazione.

Figura 3.5: Schermata delle conversazioni esistenti e Floating Action Button per avviare una nuova conversazione.

Interfaccia di Chat e Interazione

Una volta avviata la conversazione, l'utente atterra nella schermata di chat. Il layout della chat presenta un campo di testo per poter inserire le domande e un'area per visualizzare la cronologia dei messaggi. I messaggi del bot e quelli dell'utente hanno rappresentazioni grafiche differenti per essere facilmente distinguibili. Le domande dell'utente appaiono immediatamente a schermo, invece per la risposta del bot appare prima un indicatore di caricamento per fornire all'utente un input visivo mentre viene generata la risposta.

Tracciabilità e Verifica delle Fonti

La risposta del chatbot sarà basata esclusivamente sui contenuti associati alla categoria sulla quale sta avvenendo la conversazione, ogni messaggio del bot inoltre sarà correlato dalla fonte da cui ha estrapolato l'intera risposta o il frammento di risposta nel caso in cui il bot abbia bisogno di più fonti per rispondere alla domanda posta dall'utente. Ogni fonte indicherà il documento dal quale è stata dedotta la risposta e il chunk a cui si fa riferimento, in appendice al messaggio sono inoltre disponibili dei link ai vari chunk citati nella risposta. Cliccando su tali link si apre un pannello a comparsa dal basso che permette la visualizzazione per intero del chunk originale, permettendo una rapida verifica delle fonti. Questo meccanismo garantisce piena tracciabilità, permettendo all'utente di verificare l'accuratezza della risposta e di mitigare il rischio di "allucinazioni" da parte dell'IA.

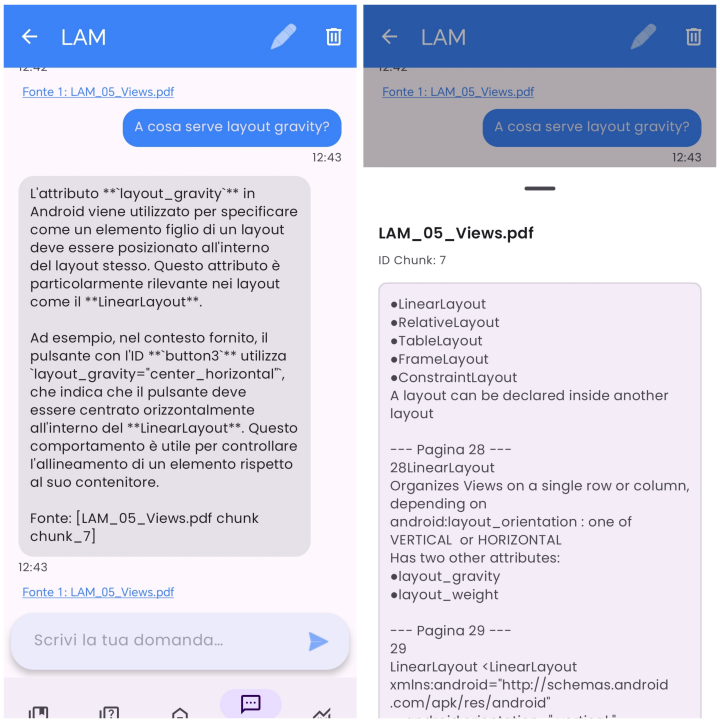


Figura 3.6: Interazione con il chatbot. L'immagine mostra sia la risposta del chatbot sia la visualizzazione di una fonte specifica.

3.2.3 Anello di Studio: Autovalutazione tramite Quiz

Dopo aver esplorato un argomento tramite la funzionalità di chat, l'utente può consolidare il proprio apprendimento attraverso l'autovalutazione. Attraverso la barra di navigazione principale è possibile accedere alla sezione "Quiz" per testare attivamente la propria preparazione. La schermata dedicata ai quiz è suddivisa in due sottosezioni, che mostrano rispettivamente i quiz generati e i tentativi effettuati, è infatti possibile ripetere un quiz già svolto e avere più tentativi per un singolo quiz.

Configurazione e Generazione di un Quiz

Attraverso la pagina dedicata ai quiz è possibile accedere alla schermata di configurazione di un nuovo quiz, progettata per offrire un elevato grado di personalizzazione, permettendo all'utente di definire con precisione le caratteristiche del test che intende generare. Le opzioni a disposizione sono le seguenti:

Selezione della Materia Come primo passo obbligatorio, l'utente deve scegliere da un elenco a discesa la materia su cui verrà basato il quiz. Questa selezione è di fondamentale importanza, poiché definisce il contesto informativo da cui il sistema RAG attingerà per formulare le domande, garantendo la loro pertinenza.

Numero di Domande L'utente può specificare il numero di domande che comporranno il test, consentendo di creare sia brevi verifiche rapide sia sessioni di autovalutazione più lunghe e approfondite.

Tipologia delle Domande Per rendere il test più vario e stimolante, l'applicazione offre la possibilità di scegliere una o più tipologie di domande da includere. Le opzioni, presentate tramite un gruppo di *chip* selezionabili, includono domande a scelta multipla, vero/falso e a risposta aperta.

Focus Tematico (Opzionale) Come funzionalità avanzata, l'utente può inserire una parola chiave o una breve frase in un campo di testo. Questa informazione viene passata al modello di linguaggio durante la generazione per guidarlo a formulare domande concentrate su un argomento specifico all'interno dei documenti della materia scelta.

Una volta impostate le proprie preferenze, l'utente preme il pulsante "Genera Quiz". L'applicazione raccoglie quindi la configurazione, la invia al backend tramite una chiamata API e mostra un indicatore di caricamento, in attesa che il sistema di intelligenza artificiale generi il test personalizzato.

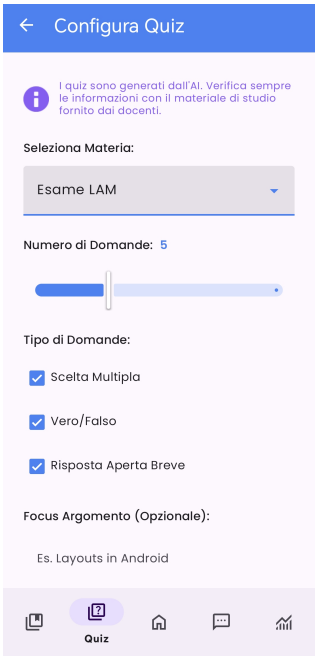


Figura 3.7: La schermata di configurazione del quiz, in cui sono visibili tutte le opzioni di personalizzazione a disposizione dell'utente.

Svolgimento del Quiz

Al termine della generazione del quiz, l'utente accede a una schermata dedicata al suo svolgimento. L'interfaccia è stata progettata per essere dinamica e in grado di adattarsi alle diverse tipologie di domande configurate.

L'interazione è semplice e intuitiva: l'utente risponde a ciascuna domanda utilizzando i controlli appropriati (selezionando un'opzione o digitando del testo) e può navigare tra le domande tramite pulsanti "Precedente" e "Successiva". Giunto all'ultima domanda, un pulsante "Concludi e Valuta" permette di inviare il quiz per la valutazione finale.

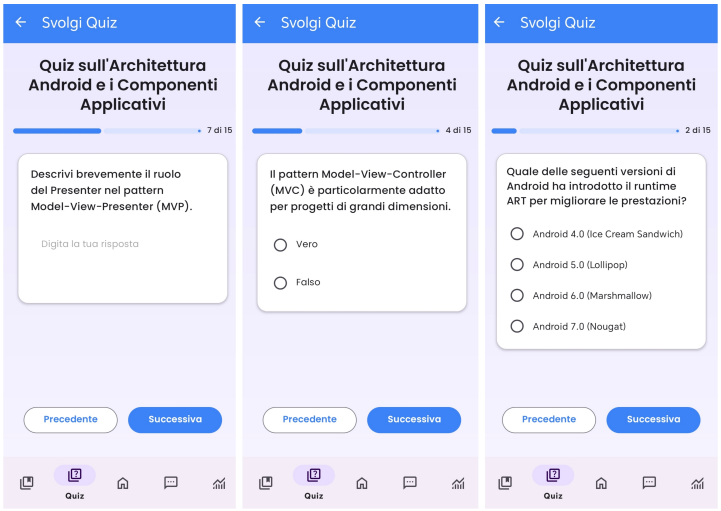


Figura 3.8: Panoramica dell'interfaccia di svolgimento del quiz. L'immagine collage mostra esempi delle diverse tipologie di domande (a scelta multipla, vero/falso, a risposta aperta) presentate all'utente.

Visualizzazione dei Risultati e Feedback

Al termine dello svolgimento, dopo aver premuto il pulsante di invio, l'utente viene indirizzato a una schermata di riepilogo dedicata alla visualizzazione dei risultati. Questa interfaccia è progettata per fornire un feedback chiaro e immediato sulla performance.

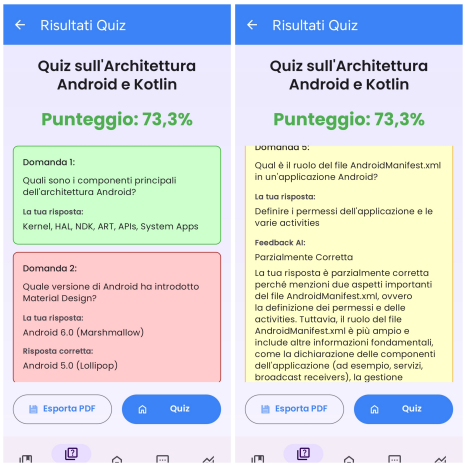


Figura 3.9: La schermata di revisione dei risultati, che mostra il punteggio complessivo e il feedback dettagliato per le singole domande.

La parte superiore della schermata presenta un sommario della prestazione, includendo il punteggio totale ottenuto in formato percentuale. Subito sotto, viene mostrato un elenco dettagliato che ripercorre ogni singola domanda del quiz per una revisione approfondita. Ogni elemento della lista mette a confronto la risposta fornita dall'utente con quella corretta, utilizzando degli indicatori visivi, come un diverso colore di sfondo, per comunicare istantaneamente l'esito (corretto, errato o parziale). Un aspetto qualificante di questa sezione è il feedback fornito per le domande a risposta aperta. In questo caso, la valutazione non si limita a un semplice giudizio binario, ma include una motivazione testuale generata dal modello di linguaggio, che spiega perché la risposta dell'utente è stata considerata corretta o meno. Infine, per consentire una revisione offline o l'archiviazione dei propri progressi, la schermata offre una funzionalità di esportazione. Tramite un'opzione nel menu, l'utente può generare un documento PDF contenente il riepilogo completo del quiz, che può essere salvato localmente o condiviso.

3.2.4 Monitoraggio dei Progressi e Gamification

L'ultima fase dell'anello di studio implementato in MyStudyApp è dedicata al monitoraggio delle performance. Dopo aver esplorato gli argomenti con la chat e aver verificato la propria conoscenza con i quiz, l'utente ha a disposizione una sezione dedicata alle "Statistiche" per analizzare i propri progressi nel tempo e rimanere motivato.

La Dashboard di Riepilogo

Accedendo alla sezione delle statistiche, l'utente viene accolto da una dashboard che offre una panoramica immediata della sua performance complessiva. Questa schermata è stata progettata per presentare i dati chiave in modo visivamente accattivante e di facile interpretazione. I componenti principali sono:

Punteggio Medio Complessivo Un grafico a torta domina la parte superiore della schermata, mostrando la media ponderata di tutti i punteggi ottenuti nei quiz. Questo indicatore fornisce una misura sintetica del livello di preparazione generale dell'utente.

Gamification: La Streak di Studio Per incentivare la costanza e la motivazione, è stato introdotto un elemento di *gamification*: la "streak" di studio. Un contatore ben visibile mostra il numero di giorni consecutivi in cui l'utente ha completato almeno una sessione di studio (un quiz). Toccando questo indicatore, l'utente può visualizzare un calendario dettagliato che evidenzia i giorni di attività, rinforzando l'abitudine positiva. Un sistema di notifiche opzionale ricorda inoltre all'utente di non interrompere la propria serie positiva.

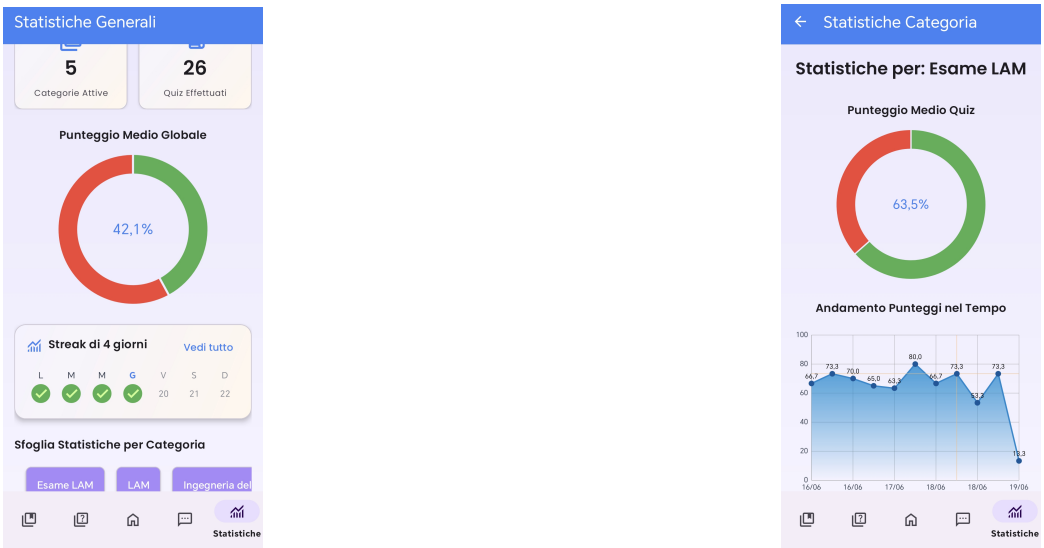
Performance per Materia Sotto gli indicatori principali, un elenco o un carosello mostra il punteggio medio per ogni singola materia, permettendo all'utente di identificare a colpo d'occhio le aree di forza e quelle che richiedono maggiore attenzione.

Analisi di Dettaglio per Materia

Dalla dashboard, l'utente può selezionare una specifica materia per accedere a un'analisi più approfondita. Questa schermata di dettaglio è fondamentale per tracciare i miglioramenti nel tempo.

L'elemento visivo chiave di questa sezione è un grafico a linee che mostra l'andamento cronologico dei punteggi ottenuti in tutti i quiz svolti per quella determinata materia. Questo grafico permette all'utente di rispondere in modo intuitivo alla domanda "Sto migliorando in questo argomento?", visualizzando una tendenza che può essere crescente, stabile o decrescente, e fornendo così uno strumento potente per la regolazione del proprio metodo di studio.

L'esperienza utente per la sezione statistiche è riassunta visivamente nella figura seguente.



(a) Dashboard di riepilogo.

(b) Dettaglio per una materia.

Figura 3.10: Schermate della sezione Statistiche. A sinistra (a) la dashboard principale con il riepilogo delle performance; a destra (b) la vista di dettaglio per una singola materia, che mostra l'andamento dei punteggi nel tempo.

3.3 Conclusione del Capitolo

In questo capitolo è stato presentato il progetto dell'applicazione MyStudyApp, delineandone l'architettura e l'esperienza utente. È stato descritto il sistema client-server, l'infrastruttura basata su servizi cloud di Microsoft Azure e le strategie di sicurezza adottate. Successivamente, attraverso un walkthrough guidato, è stato illustrato come queste fondamenta tecniche si traducano in un flusso di interazione coeso, che realizza un ciclo di studio completo. Dopo aver definito il "cosa" è stato costruito e il "come" a livello concettuale, il capitolo seguente si addentrerà nei dettagli implementativi dei moduli software chiave.

Capitolo 4

Dettagli di Implementazione

Dopo aver delineato nel capitolo precedente l'architettura generale di "MyStudyApp" e il suo flusso utente, questo capitolo si addentra negli aspetti più tecnici del progetto. L'obiettivo è analizzare in dettaglio le scelte implementative e i componenti software chiave che costituiscono sia il backend, sviluppato in Python, sia il frontend nativo per Android. Spostando l'attenzione dall'architettura concettuale al codice sorgente, verranno presentati frammenti di codice significativi per illustrare le soluzioni adottate per risolvere specifiche sfide di sviluppo, motivando le decisioni prese. Si inizierà con l'analisi del backend, il "cervello" del sistema, per poi passare al frontend, l'interfaccia con cui l'utente interagisce direttamente.

4.1 Implementazione del Backend (Python con FastAPI)

Il backend, sviluppato in Python utilizzando il framework FastAPI, rappresenta il nucleo computazionale dell'applicazione. Gestisce l'intera logica di business, dall'orchestrazione della pipeline RAG all'esposizione delle funzionalità tramite un'interfaccia API RESTful. La sua implementazione è stata guidata dalla necessità di creare un servizio performante, scalabile e manutenibile.

4.1.1 Struttura del Progetto e Dipendenze Chiave

Per garantire chiarezza e manutenibilità, il codice del backend è stato organizzato in una struttura logica che, sebbene contenuta in un singolo file principale per semplicità di deployment su Azure Container Apps, segue una chiara separazione delle responsabilità. Le funzionalità sono raggruppate per dominio: gestione delle categorie, indicizzazione dei documenti, interazione con la chat e generazione dei quiz.

Il funzionamento del backend si basa su un ecosistema di librerie Python specializzate. Di seguito sono elencate le dipendenze chiave e il loro ruolo specifico all'interno del progetto:

FastAPI È il framework web scelto per la costruzione delle API. La sua sintassi moderna basata sui type hint di Python, le performance elevate e la generazione automatica della documentazione interattiva (tramite Swagger UI) lo hanno reso la scelta ideale per sviluppare rapidamente un'API robusta.

Pydantic Utilizzata in sinergia con FastAPI, questa libreria gestisce la validazione dei dati. Permette di definire modelli di dati (come `QuizRequest` o `QAResponse`) tramite semplici classi Python, garantendo che i dati in ingresso e in uscita dalle API rispettino sempre la struttura attesa.

Uvicorn È il server ASGI (Asynchronous Server Gateway Interface) che si occupa di eseguire l'applicazione FastAPI, gestendo le connessioni e le richieste HTTP in modo asincrono ed efficiente.

SDK di Azure (‘azure-identity‘, ‘azure-storage-blob‘, ‘azure-search-documents‘) Questa suite di librerie è fondamentale per l'interazione con l'infrastruttura cloud. ‘azure-identity‘ gestisce l'autenticazione sicura tramite Managed Identities; ‘azure-storage-blob‘ permette di caricare e gestire i file degli utenti su Blob Storage; ‘azure-search-documents‘ è utilizzata per indicizzare e interrogare i chunk di testo e i loro vettori su Azure AI Search.

OpenAI (‘openai‘) La libreria ufficiale di OpenAI, utilizzata per comunicare con il servizio Azure OpenAI. È responsabile sia della generazione degli embedding testuali (con ‘text-embedding-ada-002‘) sia della generazione di risposte complesse (con ‘gpt-4o‘).

Tiktoken Libreria sviluppata da OpenAI per contare in modo efficiente il numero di token in un testo secondo le codifiche usate dai loro modelli. È utilizzata per garantire che i chunk di testo non superino i limiti di contesto dei modelli di embedding.

PyPDF2 e python-docx Queste due librerie sono essenziali per la fase di estrazione del testo. ‘PyPDF2‘ viene utilizzata per leggere il contenuto dei file PDF, mentre ‘python-docx‘ gestisce l’estrazione dai file in formato Word (.docx).

4.1.2 La Pipeline di Indicizzazione: dall’Upload all’Indicizzazione

Il cuore della fase di ingestione dei dati è rappresentato dall’endpoint `/uploadfile/`, che orchestra l’intero processo di trasformazione di un documento grezzo, caricato dall’utente, in una serie di frammenti di informazione vettorializzati e ricercabili. Questa pipeline è fondamentale poiché prepara la base di conoscenza su cui si fondano tutte le capacità di interrogazione dell’applicazione. Il processo, attivato da una richiesta POST a questo endpoint, si articola in quattro passaggi sequenziali e automatizzati.

Passo 1: Estrazione del Testo

La prima operazione consiste nell’estrarre il contenuto testuale puro dal file caricato. Il backend è progettato per gestire dinamicamente diversi formati di documento, tra cui PDF, Microsoft Word (.docx) e testo semplice (.txt). Per fare ciò, una funzione di smistamento ispeziona l’estensione del

file e invoca il parser appropriato. Per i file PDF si utilizza la libreria `PyPDF2`, mentre per i documenti Word si fa affidamento su `python-docx`.

```python
# Funzioni per l'estrazione del testo

def extract_text_from_pdf(file_path: str) -> str:
    with open(file_path, "rb") as f:
        reader = PdfReader(f)
        text = ""
        for i, page in enumerate(reader.pages):
            page_text = page.extract_text()
            if page_text:
                text += f"\n--- Pagina {i+1} ---\n{page_text}"
        return text

def extract_text_from_docx(file_path: str) -> str:
    doc = docx.Document(file_path)
    text = "\n".join([paragraph.text for paragraph in
    ↪  doc.paragraphs])
    return text
```

Listing 1: Logica per l'estrazione del testo da diversi formati di file.

Passo 2: Suddivisione in Chunk (Chunking)

Una volta ottenuto il testo completo, questo viene suddiviso in frammenti più piccoli ("chunk"). A differenza di approcci basati su librerie esterne, è stata implementata una funzione custom, `chunk_text_by_tokens`, che opera direttamente a livello di token. Utilizzando il tokenizer `tiktoken`, il testo viene prima codificato in una lista di token. Successivamente, la lista viene suddivisa in blocchi di una dimensione massima definita, gestendo una sovrapposizione (overlap) tra chunk consecutivi per preservare il contesto

semantico. Questo controllo a basso livello garantisce che ogni chunk sia ottimizzato per i modelli di embedding.

```python
# Funzione custom per il chunking basato su token

import tiktoken

def chunk_text_by_tokens(text: str, model_encoding: str = "cl100k_base", max_tokens: int = 512, overlap: int = 100) -> List[str]:
    try:
        tokenizer = tiktoken.get_encoding(model_encoding)
    except Exception as e:
        tokenizer = tiktoken.get_encoding("cl100k_base")

    all_tokens = tokenizer.encode(text, allowed_special='all')
    total_tokens = len(all_tokens)
    chunks = []
    start_token = 0
    while start_token < total_tokens:
        end_token = min(start_token + max_tokens, total_tokens)
        current_chunk_tokens = all_tokens[start_token:end_token]
        chunk_text = tokenizer.decode(current_chunk_tokens)
        chunks.append(chunk_text)
        if end_token == total_tokens:
            break
        # Gestione overlap
        start_token += (max_tokens - overlap)
    return chunks
```

Listing 2: Funzione custom per la suddivisione del testo in chunk basati su token.

Passo 3: Generazione degli Embedding

Ogni chunk di testo viene trasformato in una rappresentazione vettoriale numerica, o "embedding". Questo vettore cattura il significato semantico del testo in uno spazio multidimensionale. Per questo compito, il backend invia ogni chunk al modello `text-embedding-ada-002` tramite il servizio Azure OpenAI.

```
# Funzione per la generazione di embedding

def generate_embeddings(texts: List[str]) -> List[List[float]]:
    try:
        embeddings_response = openai_client.embeddings.create(
            input=texts, model=AZURE_OPENAI_EMBEDDING_DEPLOYMENT
        )
        embs = [emb.embedding for emb in embeddings_response.data]
        return embs
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Errore: {e}")
```

Listing 3: Funzione per la generazione di embedding da una lista di chunk di testo.

Passo 4: Indicizzazione su Azure AI Search

L'ultimo passo consiste nel memorizzare i chunk e i loro corrispondenti embedding in un indice di ricerca specializzato. Si utilizza Azure AI Search, che combina ricerca full-text e vettoriale. Per ogni chunk, viene creato un documento JSON contenente il testo, l'embedding e metadati utili come un UUID, il nome del file di origine e la categoria. Questo insieme di documenti viene poi caricato nell'indice, rendendo la conoscenza immediatamente disponibile.

```python
# Preparazione dei documenti per l'indicizzazione su Azure AI Search

# ... all'interno dell'endpoint /uploadfile/ ...
chunk_embeddings = generate_embeddings(chunks)

documents_to_index = [
    {
        "id": f"{uuid.uuid4()}",
        "content": chunk,
        "embedding": chunk_embeddings[i],
        "sourcefile": file.filename,
        "sourcepage": f"chunk_{i+1}",
        "storageUrl": blob_url or "",
        "category": category
    }
    for i, chunk in enumerate(chunks)
]

search_client.upload_documents(documents=documents_to_index)
```

Listing 4: Struttura dati dei documenti preparati per l'indicizzazione.

4.1.3 La Pipeline di Interrogazione: Chat e Quiz

Se la fase di indicizzazione rappresenta la preparazione della conoscenza, la pipeline di interrogazione è il processo dinamico che si attiva quando l'utente interagisce con l'applicazione. Questa pipeline gestisce la logica per fornire risposte contestualizzate, sia durante una conversazione nella chat, sia durante la generazione di un quiz. Sebbene gli endpoint `/query` e `/quiz` abbiano finalità diverse, condividono la stessa architettura di base, che si articola in due fasi principali: il recupero del contesto (Retrieval) e la generazione

aumentata (Augmented Generation).

Fase 1: Recupero del Contesto (Retrieval)

Indipendentemente dalla richiesta, il primo passo è sempre quello di recuperare i frammenti di testo più pertinenti dalla base di conoscenza indicizzata su Azure AI Search. La funzione `retrieve_chunks_from_search` si occupa di questa operazione, eseguendo una ricerca vettoriale filtrata per categoria.

```
# Funzione per il recupero dei chunk da Azure AI Search

def retrieve_chunks_from_search(query_embedding: List[float],
    index_name: str, top: int, filter_string: Optional[str] = None)
    -> List[Dict[str, any]]:
    try:
        vector_query = VectorizedQuery(
            vector=query_embedding,
            k_nearest_neighbors=top,
            fields="embedding"
        )
        results = search_client.search(
            search_text=None,
            vector_queries=[vector_query],
            select=["id", "content", "sourcefile", ...],
            filter=filter_string,
            top=top
        )
        # ... (costruzione della lista di documenti dai risultati)
        return docs
    except Exception as e:
        # ... (gestione eccezioni) ...
```

Listing 5: Funzione per eseguire una ricerca vettoriale filtrata sull'indice.

Fase 2: Generazione Aumentata (Augmented Generation)

Con i chunk pertinenti a disposizione, il sistema passa alla generazione della risposta. I chunk recuperati vengono assemblati in un **contesto** che viene inserito in un prompt strutturato, insieme alla richiesta originale, per istruire il Large Language Model.

Generazione Risposta Chat (`/query`) Per la chat, il contesto recuperato viene formattato e inserito in un prompt di sistema che istruisce l'LLM a rispondere basandosi esclusivamente sulle fonti fornite e a citarle.

```
# Logica di generazione della risposta nella chat

# ... (recupero dei chunk in retrieved_docs) ...

# Formattazione del contesto per il prompt
context = "\n\n".join([f"Fonte [{doc.get('sourcefile', 'N/A')}
    ...]:\n{doc.get('content', '')}" for doc in retrieved_docs])

messages = [
    {"role": "system", "content": "Sei un assistente AI utile.
        Rispondi alla domanda basandoti SOLO sul contesto fornito.
        Cita le fonti usando [nomefile chunk_numero]..."},
    {"role": "user", "content": f"Contesto:\n{context}\n\nDomanda:
        {request.question}"}
]
response = openai_client.chat.completions.create(
    # ... (parametri come modello, messages) ...
)
```

Listing 6: Costruzione del prompt e chiamata all'LLM per la funzionalità di chat.

Generazione Domande Quiz (/quiz) Per la generazione dei quiz, il prompt è più complesso e istruisce l'LLM a restituire l'output in un formato JSON rigoroso.

```
# Estratto dal prompt di sistema per la generazione dei quiz
json_schema_description = """
{
  "quiz_title": "...",
  "questions": [
    {"question_type": "multiple_choice",  ... },
    ...
  ]
}"""
system_prompt = f"""Sei un assistente AI esperto... generare un
↪  quiz di {num_questions} domande basandoti ESCLUSIVAMENTE sul
↪  'Contesto'.
Analizza l'INTERO 'Contesto' e crea domande...
Devi generare domande UTILIZZANDO ESCLUSIVAMENTE i seguenti tipi:
↪  {desired_types_str}.
Devi restituire ESATTAMENTE un oggetto JSON...:
{json_schema_description}
"""
messages = [{"role": "system", "content": system_prompt},
            {"role": "user", "content":
             ↪  f"Contesto:\n{context}\n\nGenera il quiz..."}]
response = openai_client.chat.completions.create(
    # ... parametri ...
    response_format={"type": "json_object"} # Richiede output JSON
)
```

Listing 7: Definizione del prompt e dello schema JSON per la generazione dei quiz.

4.1.4 Valutazione delle Risposte Aperte

Una delle funzionalità più avanzate di "MyStudyApp" è la capacità di fornire un feedback qualitativo e automatizzato per le domande a risposta aperta dei quiz. Questa logica è incapsulata nell'endpoint `/evaluate_answer/`, che trasforma un semplice giudizio binario (corretto/errato) in un'opportunità di apprendimento mirato. Il processo sfrutta la stessa pipeline RAG vista in precedenza, ma con un obiettivo diverso: non generare una risposta, ma valutarne una.

Fase 1: Recupero del Contesto Pertinente

Quando l'utente invia la propria risposta a una domanda aperta, il sistema non la valuta nel vuoto. Come primo passo, esegue una ricerca semantica utilizzando il testo della *domanda originale* come query. Questo permette di recuperare i "k" chunk di testo più pertinenti dalla categoria specificata, che costituiscono il contesto fattuale su cui si baserà la valutazione. Questa operazione viene eseguita dalla stessa funzione `retrieve_chunks_from_search` utilizzata in precedenza.

Fase 2: Generazione della Valutazione tramite LLM

Il contesto recuperato, insieme alla domanda originale e alla risposta dell'utente, viene assemblato in un prompt strutturato. Questo prompt istruisce l'LLM a comportarsi come un "tutor esperto" e a restituire la sua valutazione in un formato JSON predefinito. La funzione `evaluate_user_answer_with_llm` gestisce questa interazione.

Il prompt di sistema è cruciale: definisce il ruolo del modello, lo vincola a usare solo il contesto fornito e specifica la struttura esatta dell'output JSON, che include un punteggio qualitativo, un feedback testuale, suggerimenti per migliorare e, se necessario, un esempio di risposta corretta.

```python
# Funzione per la valutazione della risposta utente
async def evaluate_user_answer_with_llm(question: str, user_answer: str, context: str) -> EvaluationResponse:
    json_schema_description = """
    Restituisci ESATTAMENTE un oggetto JSON con la seguente struttura:
    {
      "evaluation_score": "Valutazione qualitativa (es. 'Corretta', 'Parzialmente Corretta', 'Errata', ...)",
      "feedback_text": "Una spiegazione dettagliata della valutazione...",
      "suggestions": "Suggerimenti specifici e costruttivi...",
      "example_correct_answer": "Un esempio di risposta completa e corretta..."
    }
    """
    system_prompt = f"""Sei un tutor AI esperto... Il tuo compito è valutare la 'Risposta Utente' ... basandoti ESCLUSIVAMENTE sul 'Contesto Originale'.
    ... (istruzioni su come rivolgersi all'utente e formato JSON) ...
    {json_schema_description}"""
    user_prompt = f"""Contesto Originale:\n{context}\n\nDomanda:\n{question}\n\nRisposta Utente:\n{user_answer}\n\nGenera la valutazione in formato JSON..."""
    messages = [{"role": "system", "content": system_prompt},
                {"role": "user", "content": user_prompt}]
    response = openai_client.chat.completions.create(
        #... parametri...
        response_format={"type": "json_object"}
    )
    evaluation_data = EvaluationResponse.model_validate_json(response.choices[0].message.content)
    return evaluation_data
```

Listing 8: Logica di costruzione del prompt e chiamata all'LLM per la valutazione delle risposte aperte.

Questo meccanismo permette di andare oltre la semplice correzione, fornendo all'utente un feedback ricco e argomentato che lo aiuta a comprendere i propri errori e a migliorare la propria preparazione.

4.2 Implementazione del Frontend (Android Nativo in Kotlin)

Il frontend dell'applicazione "MyStudyApp" è un'applicazione nativa per Android, sviluppata interamente in Kotlin. Agisce come l'interfaccia diretta con l'utente, gestendo la presentazione dei dati, la navigazione e l'interazione con il backend. Per garantire un'architettura robusta, scalabile e facilmente manutenibile, il progetto è stato strutturato seguendo il pattern architetturale MVVM (Model-View-ViewModel).

4.2.1 Architettura MVVM e Gestione Reattiva dello Stato

Il pattern MVVM è stato applicato per separare la logica di business e la gestione dei dati (ViewModel) dall'interfaccia utente (View). Il `ViewModel` espone uno stato alla View tramite `StateFlow`, permettendo un aggiornamento reattivo dell'UI.

Un esempio concreto è lo `StatisticsViewModel`, che aggrega dati da più sorgenti. Espone un unico oggetto di stato, `StatisticsUiState`, che racchiude tutte le informazioni necessarie per la schermata delle statistiche, come il punteggio medio, la streak di studio e i dettagli per categoria. Invece di usare librerie di dependency injection, la sua istanziazione è gestita tramite una factory custom, `StatisticsViewModelFactory`, che inietta manualmente le dipendenze necessarie (i Repository).

```kotlin
// Estratto da StatisticsViewModel.kt

// Data class che aggrega lo stato della UI per le schermate delle statistiche.
data class StatisticsUiState(
    val isLoadingInitialData: Boolean = true,
    val overallAverageScore: Double = 0.0,
    // ... altri campi dello stato
)

// ViewModel responsabile della logica per la visualizzazione delle statistiche.
class StatisticsViewModel(
    private val quizRepository: QuizRepository,
    private val categoryRepository: CategoryRepository
) : ViewModel() {
    private val _uiState = MutableStateFlow(StatisticsUiState())
    val uiState: StateFlow<StatisticsUiState> = _uiState.asStateFlow()

    init {
        // All'inizializzazione, avvia il caricamento asincrono dei dati
        viewModelScope.launch {
            quizRepository.getAllCompletedAttempts()
                .collect { attempts ->
                    // ... logica per calcolare le statistiche ...
                    recalculateGeneralStats()
                }
        }
        // ... altri lanci di coroutine per caricare quiz e categorie ...
    }
    // ...
}
```

Listing 9: Lo `StatisticsViewModel` espone lo stato della UI tramite `StateFlow`.

Il Fragment corrispondente osserva questo `StateFlow` per aggiornare la UI. Per ottenere un'istanza del ViewModel con le sue dipendenze, utilizza la factory custom.

```kotlin
// Estratto da StatisticsOverviewFragment.kt
class StatisticsOverviewFragment : Fragment() {
    // Inizializzazione del ViewModel condiviso
    private val statisticsViewModel: StatisticsViewModel by
    ↪ activityViewModels {
        StatisticsViewModelFactory(/* ... repository ... */)
    }
    override fun onViewCreated(view: View, savedInstanceState:
    ↪ Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        observeUiState()
    }
    // Osserva lo StateFlow per aggiornare la UI in modo reattivo.
    private fun observeUiState() {
        viewLifecycleOwner.lifecycleScope.launch {
            // Esegue la raccolta solo quando il Fragment e' attivo

            ↪ viewLifecycleOwner.repeatOnLifecycle(Lifecycle.State.STARTED)
            ↪ {
                statisticsViewModel.uiState.collect { state ->
                    // La UI viene aggiornata qui in base allo
                    ↪ 'state' ricevuto.
                }
            }
        }
    }
}
```

Listing 10: Il `Fragment` osserva lo `StateFlow` del ViewModel.

4.2.2 Persistenza Dati Locale con Room

Per la persistenza dei dati locali è stata utilizzata la libreria **Room**. Ogni tabella del database è rappresentata da una classe `@Entity` e le operazioni di accesso sono definite in un'interfaccia `@Dao`. L'uso di `Flow` come tipo di ritorno per le query permette al Repository di ricevere aggiornamenti reattivi dal database. Nello specifico, il database locale è strutturato per gestire l'intero ciclo di vita dell'attività dello studente all'interno dell'app. Le entità principali memorizzate includono:

- **Categorie e Documenti:** Le informazioni relative alle materie create dall'utente e i metadati dei documenti associati a ciascuna categoria.

- **Quiz e Tentativi:** La struttura completa dei quiz generati (domande, opzioni, risposte corrette) e tutti i tentativi di svolgimento effettuati dall'utente, incluse le risposte fornite e i punteggi ottenuti.

- **Conversazioni della Chat:** L'intera cronologia dei messaggi scambiati tra l'utente e il chatbot per ogni specifica materia, garantendo la persistenza delle conversazioni.

- **Dati per le Statistiche:** Informazioni aggregate, come i punteggi e le date di completamento dei quiz, che vengono utilizzate per calcolare le performance dell'utente e la "streak" di studio.

Questa architettura di persistenza garantisce che i dati dell'utente siano sempre disponibili, anche offline, e permette all'applicazione di funzionare in modo reattivo e fluido, riducendo le chiamate di rete al minimo indispensabile.

```kotlin
// Estratto da QuizAttemptEntity.kt e QuizDao.kt
@Entity(tableName = "quiz_attempts")
data class QuizAttemptEntity(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    @ColumnInfo(name = "quiz_id") val quizId: Long,
    // ... altri campi omessi per brevità ...
)
@Dao
interface QuizDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertQuizAttempt(attempt: QuizAttemptEntity): Long
    // Query reattiva che emette la lista aggiornata ad ogni cambiamento
    @Query("SELECT * FROM quiz_attempts WHERE is_completed = 1 ORDER BY completion_timestamp DESC")
    fun getAllCompletedAttempts(): Flow<List<QuizAttemptEntity>>
}
```

Listing 11: Definizione dell'entità `QuizAttemptEntity` e del suo DAO.

4.2.3 Comunicazione di Rete con Retrofit

La comunicazione con il backend API REST è gestita tramite **Retrofit**. Le operazioni dell'API sono definite nell'interfaccia `ApiService`.

```kotlin
// Estratto da ApiService.kt
interface ApiService {
    @GET("categories")
    suspend fun getCategories(): Response<CategoriesResponseDto>
    @POST("quiz")
    suspend fun generateQuiz(@Body quizRequest: QuizRequestDto):
    ↪ Response<QuizResponseDto>
}
```

Listing 12: Definizione degli endpoint nell'interfaccia del servizio Retrofit.

Per centralizzare l'autenticazione, è stata usata una concisa Interceptor lambda all'interno di `RetrofitInstance`. Questa aggiunge a ogni chiamata l'header `X-API-Key`, la cui chiave proviene da `BuildConfig`.

```kotlin
// Implementazione da RetrofitInstance.kt
object RetrofitInstance {
    // Interceptor per aggiungere l'header X-API-Key.
    private val apiKeyInterceptor = Interceptor { chain ->
        val originalRequest = chain.request()
        val newRequest = originalRequest.newBuilder()
            .header("X-API-Key", BuildConfig.MY_APP_API_KEY)
            .build()
        chain.proceed(newRequest)
    }
    // Client OkHttp che utilizza l'interceptor definito sopra.
    private val okHttpClient = OkHttpClient.Builder()
        .addInterceptor(apiKeyInterceptor)
        .build()
}
```

Listing 13: Implementazione dell'interceptor in `RetrofitInstance.kt`.

4.2.4 Task in Background con WorkManager

Per le attività in background, come i promemoria per la "streak" di studio, si usa `WorkManager`. Lo `StreakReminderWorker` calcola periodicamente la streak dell'utente e invia una notifica se questa è attiva ma non ci sono stati quiz in giornata.

```
// Estratto da StreakReminderWorker.kt
class StreakReminderWorker(
    appContext: Context,
    workerParams: WorkerParameters,
) : CoroutineWorker(appContext, workerParams) {
    override suspend fun doWork(): Result {
    return withContext(Dispatchers.IO) {
        try {
            // Logica per calcolare la streak di studio
            val currentStreak = calculateStudyStreak(...)
            val quizDoneToday = wasQuizDoneToday(...)
            // Invia notifica solo se la streak è attiva e non si è
            ↪ studiato oggi
            if (currentStreak > 0 && !quizDoneToday) {
              sendStreakReminderNotification(applicationContext,
              ↪ currentStreak)
            }
            Result.success()
        } catch (e: Exception) {
            Result.failure()
        }
    }
}
```

Listing 14: Logica del worker per l'invio delle notifiche di promemoria.

Questo worker viene schedulato tramite una richiesta `PeriodicWorkRequest` dall'applicazione, garantendo che il controllo venga eseguito periodicamente.

4.3 Conclusione del Capitolo

In questo capitolo, l'analisi si è spostata dall'architettura concettuale al codice sorgente, esaminando in dettaglio le scelte implementative che costituiscono il cuore tecnico di "MyStudyApp". Sono stati sezionati i due pilastri del sistema: il backend in Python e il frontend nativo per Android.

Per il backend, si è mostrato come il framework FastAPI sia stato utilizzato per orchestrare una robusta pipeline di Retrieval-Augmented Generation. Attraverso l'analisi di funzioni specifiche, è stato illustrato il processo di indicizzazione – dalla suddivisione custom dei documenti in chunk tramite `tiktoken` fino alla loro vettorizzazione e memorizzazione su Azure AI Search. Successivamente, si è visto come la pipeline di interrogazione sfrutti questi indici per recuperare contesto pertinente e generare, tramite prompt ingegnerizzati ad hoc, risposte coerenti per la chat, quiz strutturati in formato JSON e valutazioni qualitative per le risposte aperte.

Per il frontend, è stato dimostrato come l'architettura MVVM, supportata dai componenti Android Jetpack, abbia permesso di costruire un'interfaccia utente reattiva e disaccoppiata. Sono state analizzate le implementazioni di `StateFlow` per la gestione dello stato, di Room per una persistenza dati locale efficiente e reattiva, di Retrofit con un `Interceptor` per una comunicazione di rete sicura e centralizzata, e di WorkManager per la gestione di task in background affidabili.

In sintesi, questo capitolo ha evidenziato come le decisioni tecniche di basso livello abbiano concretamente dato vita all'architettura delineata, realizzando un sistema coeso in cui un backend potente e un frontend moderno collaborano per offrire un'esperienza utente fluida e intelligente. Con le fondamenta implementative stabilite, l'analisi può ora procedere alla valutazione del sistema finale.

Capitolo 5

Valutazione Sperimentale e Analisi Comparativa

Dopo aver descritto l'architettura e l'implementazione di "MyStudyApp" nei capitoli precedenti, questa sezione si concentra sulla validazione sperimentale del sistema. Per valutare in modo oggettivo l'efficacia delle scelte progettuali e tecnologiche adottate, è stata condotta un'analisi comparativa approfondita tra l'architettura cloud-based dell'applicazione e un'implementazione alternativa, basata interamente su modelli eseguiti in locale.

L'obiettivo di questo capitolo non è semplicemente decretare un "vincitore", ma piuttosto analizzare scientificamente i punti di forza e di debolezza di entrambi gli approcci in un contesto d'uso reale. Questa analisi è fondamentale per dimostrare come l'architettura scelta per "MyStudyApp" non solo funzioni, ma rappresenti anche la soluzione più adeguata per offrire un'esperienza di studio di alta qualità, affidabile ed efficace, colmando così il gap funzionale che ha motivato questo lavoro di tesi.

La valutazione è stata articolata su due piani distinti e complementari:

1. Una valutazione **quantitativa**, basata su metriche oggettive (*Similarity*, *Coherence*, *Fluency* e *Relevance*), per misurare la qualità e l'accuratezza delle risposte fornite dalla funzionalità di Q&A di entrambi i sistemi.

2. Una valutazione **qualitativa**, condotta tramite un sondaggio somministrato a un gruppo di utenti finali (studenti universitari), per misurare la qualità e l'utilità percepite dei quiz generati e per raccogliere feedback diretti sull'esperienza d'uso.

Nelle sezioni seguenti, verranno illustrati in dettaglio il disegno sperimentale, le metodologie di valutazione adottate, i risultati numerici e qualitativi ottenuti e, infine, un'analisi critica che discuterà le implicazioni di tali risultati e validerà le scelte architetturali alla base di "MyStudyApp".

5.1 Disegno Sperimentale

In questa sezione introduttiva, si delinea l'impostazione del confronto, definendo i sistemi messi alla prova, gli obiettivi della valutazione e l'ambiente di test, al fine di garantire la trasparenza e la riproducibilità dell'analisi.

5.1.1 Definizione dei Sistemi a Confronto

La valutazione si basa sul confronto diretto tra due implementazioni con architetture fondamentalmente diverse: una basata su servizi cloud e una eseguita interamente in locale all'interno dell'applicazione stessa.

Sistema A: "MyStudyApp" (Cloud-based RAG) È l'architettura principale della tesi, così come descritta nel Capitolo 3. Questo sistema si basa su una pipeline distribuita su servizi Microsoft Azure, che include Azure AI Search per l'indicizzazione e il recupero delle informazioni e Azure OpenAI Service per l'accesso a modelli linguistici di grandi dimensioni (`gpt-4o` per la generazione e `text-embedding-ada-002` per la vettorizzazione).

Sistema B: "Offline RAG" (Local-based) È un'implementazione alternativa, integrata direttamente in una versione parallela dell'applicazione Android e scritta in Kotlin. Questa architettura, progettata per

operare interamente sul dispositivo, si basa sulla libreria **MediaPipe** di Google per orchestrare una pipeline RAG locale. Utilizza il modello LLM open-source `gemma3-1b-it-int4` per la generazione delle risposte e il modello `Gecko_1024_quant`, anch'esso eseguito localmente, per la creazione degli embedding e la ricerca semantica.

5.1.2 Obiettivi della Valutazione

L'esperimento si pone l'obiettivo di rispondere in modo sistematico alle seguenti domande di ricerca:

1. **Qualità del Sistema di Q&A**: Quale dei due sistemi fornisce risposte qualitativamente superiori in termini di accuratezza semantica (Similarity), coerenza logica, fluidità linguistica e pertinenza alla domanda?

2. **Qualità Percepita dei Quiz**: Quale dei due sistemi è percepito come più efficace dagli utenti finali (studenti) nella generazione di quiz utili e pertinenti per l'autovalutazione?

3. **Analisi dei Trade-off Architetturali**: Al di là della qualità, quali sono i vantaggi e gli svantaggi pratici di un'architettura cloud rispetto a una locale in questo specifico caso d'uso (es. latenza, costi operativi, privacy, scalabilità e flessibilità)?

5.1.3 Setup Sperimentale

- **Dataset di Valutazione**: Per garantire l'oggettività del test, la valutazione quantitativa del sistema di Q&A è stata condotta utilizzando un corpus di domande e risposte estratto dal dataset standardizzato **SQuAD (Stanford Question Answering Dataset)**. Per creare un ambiente di test controllato, la valutazione si è basata su un singolo documento di contesto, ovvero un articolo di Wikipedia (il database

SQuAD si basa su un insieme di articoli tratti da lì) e sono state selezionate unicamente le coppie di domanda e risposta pertinenti a tale documento. Per neutralizzare le differenze nelle capacità multilingua dei modelli e assicurare un confronto equo, entrambi i sistemi sono stati istruiti a generare le risposte in lingua inglese. Per la valutazione qualitativa, invece, i documenti di riferimento erano materiali di studio reali, relativi a un esame universitario, forniti a un gruppo omogeneo di utenti.

- **Pipeline di Valutazione Quantitativa**: Le metriche sono state calcolate tramite una pipeline di valutazione appositamente sviluppata in Python. Uno script orchestratore legge da un corpus di input le domande, le risposte di riferimento e le risposte pre-generate da ciascun modello. Per ogni set di dati, invoca le funzioni di un modulo di valutazione dedicato, il quale a sua volta interroga dei modelli LLM esterni usati come giudici per ottenere i punteggi delle metriche. Infine, lo script orchestratore aggrega i risultati e li formatta in un file di output strutturato. È importante sottolineare che, sebbene un test su larga scala sarebbe scientificamente desiderabile, la valutazione è stata condotta su un sottoinsieme rappresentativo del corpus a causa dei costi economici e computazionali associati all'interrogazione di modelli LLM di grandi dimensioni tramite API.

- **Strumenti di Valutazione Qualitativa**: La valutazione qualitativa, invece, è stata condotta tramite l'interazione diretta con l'applicazione "MyStudyApp" e la somministrazione di un sondaggio online a studenti universitari tramite Google Forms.

5.2 Metodologie e Metriche di Valutazione

Questa sezione descrive nel dettaglio *come* sono state misurate le performance dei due sistemi. Vengono illustrate sia le metriche quantitative

automatizzate, calcolate tramite un modello LLM usato come giudice, sia la struttura del sondaggio qualitativo somministrato agli utenti finali per valutare l'efficacia percepita dei quiz.

5.2.1 Valutazione Quantitativa del Sistema di Q&A

Per valutare oggettivamente la qualità delle risposte generate, è stata utilizzata una pipeline di valutazione automatizzata che confronta le risposte fornite dai due sistemi (Cloud e Offline) con delle risposte di riferimento ("ground truth"). Un modello LLM (`gpt-4o`) è stato impiegato come giudice imparziale, istruito a fornire un punteggio intero da 1 a 5 per ciascuna delle seguenti metriche.

Similarità (Similarity) È la metrica fondamentale di questa valutazione, poiché misura il grado di equivalenza semantica tra la risposta generata dal sistema e la risposta corretta di riferimento. Al modello-giudice viene chiesto di valutare se le informazioni e il contenuto nella risposta predetta sono simili o equivalenti alla risposta corretta. La scala di valutazione va da 1 ("per niente simile") a 5 ("completamente simile"), premiando le risposte che veicolano lo stesso significato della fonte di verità, anche se con una formulazione diversa.

Coerenza (Coherence) Questa metrica valuta la presentazione logica e ordinata delle idee all'interno della risposta, permettendo al lettore di seguire facilmente il filo del discorso. Il giudizio si basa sulla presenza di connessioni chiare tra le frasi e su una sequenza logica che indirizzi direttamente la domanda. Una risposta ottiene un punteggio massimo (5) se è eccezionalmente coerente, con una struttura e un flusso impeccabili, mentre ottiene un punteggio minimo (1) se consiste in parole o frasi sconnesse e incomprensibili.

Fluidità (Fluency) La fluidità si riferisce alla qualità della comunicazione scritta, focalizzandosi sulla correttezza grammaticale, l'ampiezza del

vocabolario, la complessità delle frasi e la leggibilità generale. Una risposta con "Fluidità Eccezionale" (5) dimostra una padronanza del linguaggio con vocabolario sofisticato e strutture sintattiche varie, risultando scorrevole e stilisticamente curata. Al contrario, una risposta con "Fluidità Emergente" (1) è in gran parte incomprensibile a causa di errori grammaticali pervasivi e un vocabolario estremamente limitato.

Pertinenza (Relevance) Questa metrica valuta l'efficacia con cui una risposta indirizza la domanda posta. Il giudizio si basa sull'accuratezza, la completezza e la pertinenza diretta delle informazioni fornite. La scala di valutazione distingue tra una risposta "Irrilevante" (1), una "Incompleta" (3), una "Completa" (4) che include tutti i dettagli essenziali, e una "Esaustiva con Approfondimenti" (5) che, oltre a rispondere pienamente, include spunti o elaborazioni aggiuntive che ne migliorano la comprensione.

5.2.2 Valutazione Qualitativa della Generazione di Quiz

Per misurare l'efficacia percepita e l'esperienza utente, è stato condotto un sondaggio anonimo tra un gruppo di studenti universitari con familiarità con la materia oggetto dei documenti ("Laboratorio di Applicazioni Mobili"). Il sondaggio, realizzato con Google Forms, chiedeva ai partecipanti di analizzare due diversi quiz generati automaticamente (Quiz A e Quiz B) e di valutarli separatamente.

Per ciascuno dei due quiz, agli utenti è stato chiesto di esprimere il loro grado di accordo su una scala Likert da 1 ("per niente d'accordo") a 5 ("completamente d'accordo") riguardo le seguenti sei affermazioni:

1. Le domande del quiz sono pertinenti e focalizzate sui concetti chiave presenti nel materiale di studio.

2. Le domande, la risposta corretta indicata e le opzioni alternative (distrattori) sono coerenti e corrette secondo quanto riportato nel testo.

3. Il testo delle domande e delle opzioni è chiaro, ben formulato e privo di ambiguità.

4. Il quiz nel suo complesso copre in modo equilibrato i principali argomenti del materiale, senza tralasciare concetti importanti o concentrarsi solo su dettagli marginali.

5. Le opzioni sbagliate sono plausibili e stimolanti, ma chiaramente errate sulla base del testo.

6. Quanto ritieni utile questo quiz per ripassare e autovalutare la tua preparazione sull'argomento?

Limitazioni del Campione Qualitativo È importante sottolineare che, a differenza della valutazione quantitativa scalabile, la raccolta di dati qualitativi tramite sondaggio è intrinsecamente più complessa e limitata dalle risorse e dalla disponibilità dei partecipanti. Il campione di utenti per questa valutazione qualitativa, sebbene omogeneo e qualificato, è da considerarsi di dimensioni contenute. Pertanto, i risultati di questa sezione devono essere interpretati come un'indicazione qualitativa forte delle tendenze di usabilità e della percezione utente, piuttosto che come una conclusione statisticamente generalizzabile su larga scala.

5.3 Risultati Sperimentali

In questa sezione vengono presentati in modo chiaro e sintetico i dati raccolti durante la valutazione. Per fornire un'analisi completa, i risultati sono riportati sia in formato tabellare, per una lettura precisa dei valori, sia in formato grafico, per una visualizzazione immediata del confronto e della dispersione dei dati.

5.3.1 Risultati della Valutazione Quantitativa (Q&A)

La valutazione quantitativa, condotta su un campione esteso di domande relative a un singolo articolo tratto da Wikipedia, ha prodotto risultati che evidenziano una notevole differenza prestazionale tra i due sistemi. La Tabella 5.1 riporta i punteggi medi esatti ottenuti dal Sistema A (Cloud-based) e dal Sistema B (Local-based) per ciascuna delle quattro metriche di qualità.

Tabella 5.1: Confronto dei punteggi medi delle metriche di qualità per i sistemi di Q&A.

Metrica	Sistema A (Cloud RAG)	Sistema B (Offline RAG)
Similarità	4.27	2.63
Coerenza	4.53	3.53
Fluidità	3.97	2.97
Pertinenza	4.00	2.80

Per una visualizzazione più efficace, gli stessi dati, comprensivi di deviazione standard, sono presentati nel grafico a barre della Figura 5.1.

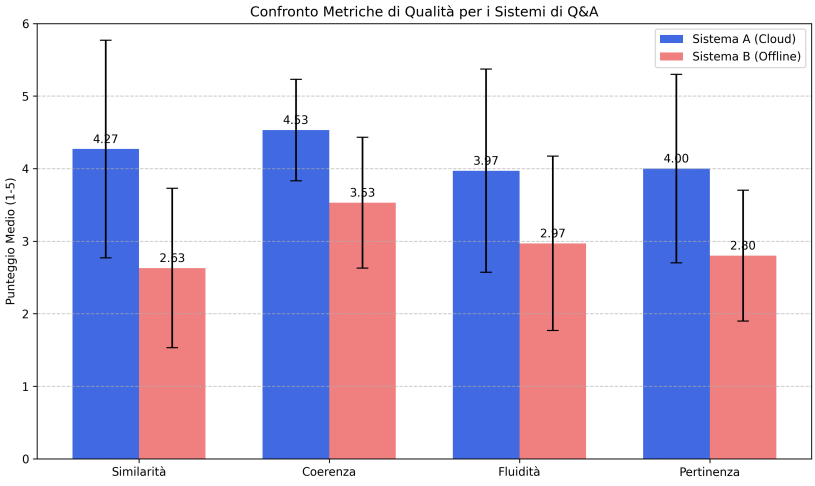


Figura 5.1: Confronto dei punteggi medi e della deviazione standard per i sistemi di Q&A.

5.3.2 Analisi delle Performance: Latenza di Risposta

Oltre alla qualità semantica delle risposte, è stata misurata la latenza di ciascun sistema, ovvero il tempo necessario per generare una risposta a partire da una domanda dell'utente. Questa metrica è fondamentale per valutare la praticità e l'esperienza utente (UX) di un sistema interattivo.

La Tabella 5.2 riassume le statistiche principali dei tempi di risposta registrati durante i test, misurati direttamente dall'applicazione Android per garantire un confronto equo.

Tabella 5.2: Confronto dei tempi di risposta (latenza) dei due sistemi.

Metrica Latenza	**Sistema A (Cloud RAG)**	**Sistema B (Offline RAG)**
Tempo Medio	1.39 s	29.96 s
Tempo Minimo	0.98 s	23.78 s
Tempo Massimo	3.42 s	43.16 s

Per un'analisi più approfondita della consistenza delle performance, la Figura 5.2 visualizza la latenza di risposta per ogni singola domanda del test.

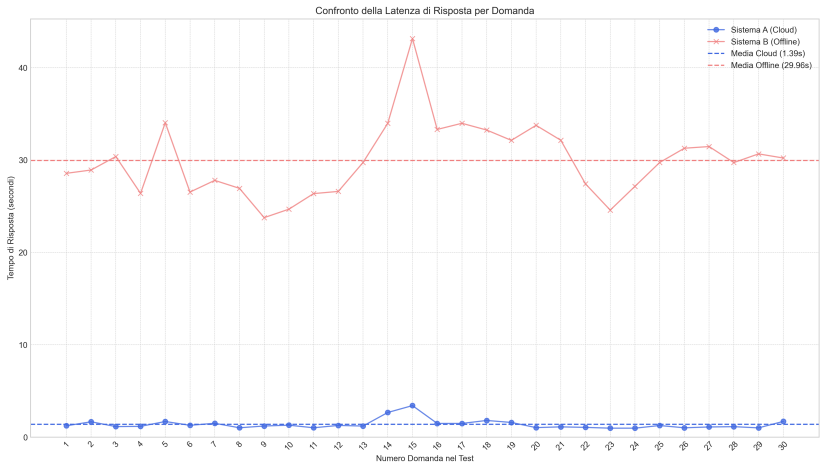


Figura 5.2: Confronto della latenza di risposta per ogni domanda del test. Le linee tratteggiate indicano il tempo di risposta medio per ciascun sistema.

L'analisi congiunta della tabella e del grafico evidenzia in modo netto la differenza prestazionale tra i due approcci. Il **Sistema A (Cloud-based)** mostra una latenza non solo nettamente inferiore, ma anche più consistente e prevedibile. Al contrario, il **Sistema B (Local-based)** presenta una latenza di un ordine di grandezza superiore, rendendo l'interazione con l'applicazione lenta e poco fluida e confermando i vincoli prestazionali come una delle sue limitazioni più significative.

5.3.3 Risultati della Valutazione Qualitativa (Quiz)

I risultati del sondaggio somministrato al gruppo di studenti hanno fornito un riscontro qualitativo sull'efficacia percepita dei quiz generati dai due sistemi. L'analisi aggregata delle risposte, riassunta nella Tabella 5.3, ha mostrato una chiara preferenza per il Sistema A (Cloud), ritenuto significativamente migliore su tutti i criteri di valutazione.

Tabella 5.3: Confronto dei punteggi medi dei criteri di valutazione qualitativa dei quiz.

Criterio di Valutazione	**Quiz A (Cloud)**	**Quiz B (Offline)**
Pertinenza e Focus	4.78	3.33
Coerenza e Correttezza	4.89	3.44
Chiarezza e Ambiguità	4.67	3.33
Copertura degli Argomenti	4.67	3.33
Qualità dei Distrattori	4.44	2.89
Utilità Generale del Quiz	4.78	3.22
Punteggio Medio Totale	**4.70**	**3.26**

Per una visualizzazione più immediata del divario tra i due sistemi e della consistenza delle valutazioni, i dati sono rappresentati anche nel grafico a barre della Figura 5.3.

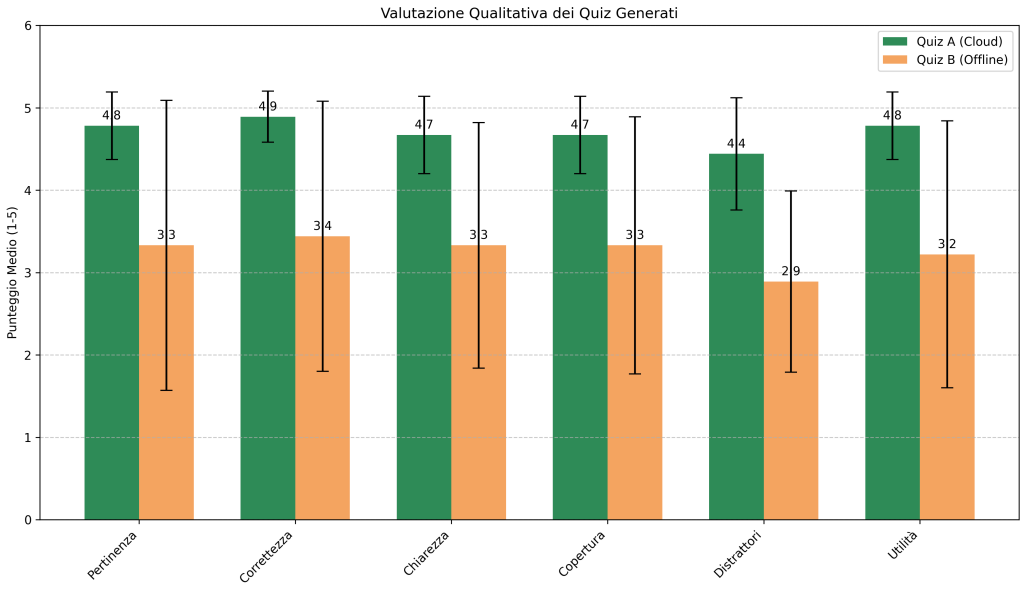


Figura 5.3: Confronto dei punteggi medi e della deviazione standard per i criteri di valutazione qualitativa dei quiz.

L'analisi congiunta dei dati mostra come il **Sistema A (Cloud)** abbia ricevuto valutazioni eccellenti e consistenti, con un punteggio medio totale di **4.70 su 5**. Il **Sistema B (Offline)**, al contrario, si ferma a una media di **3.26**, mostrando performance notevolmente inferiori su tutti i criteri legati alla qualità e all'affidabilità. Il divario più marcato si osserva sulla "Qualità dei Distrattori" (4.44 vs 2.89), indicando una maggiore capacità del sistema cloud di creare opzioni di risposta complesse e pertinenti, e confermando un'esperienza utente percepita come significativamente superiore.

5.4 Analisi e Discussione dei Risultati

Questa sezione conclusiva del capitolo interpreta i dati presentati nelle sezioni precedenti, discute i trade-off emersi dal confronto tra le due architetture e, infine, valida le scelte progettuali alla base di "MyStudyApp" in relazione agli obiettivi iniziali della tesi.

5.4.1 Interpretazione dei Risultati

L'analisi dei dati sperimentali rivela un divario di performance netto e consistente tra il Sistema A (Cloud-based) e il Sistema B (Local-based).

Nella valutazione quantitativa del Q&A (Tabella 5.1), il Sistema A ha ottenuto punteggi medi significativamente superiori in tutte e quattro le metriche. La differenza più marcata si riscontra nella metrica di **Similarità**, con un punteggio di 4.27 per il sistema cloud contro il 2.63 del sistema locale. Questo dato è di fondamentale importanza, poiché indica che le risposte del Sistema A non sono solo qualitativamente migliori, ma anche semanticamente molto più accurate e fedeli alla fonte di verità ("ground truth"). Tale superiorità suggerisce che l'impiego di modelli più avanzati e potenti come `gpt-4o` è cruciale per garantire l'affidabilità fattuale delle risposte, mitigando il rischio di generare informazioni errate.

Questa conclusione è fortemente corroborata dai risultati del sondaggio qualitativo (Tabella 5.3). Gli utenti hanno valutato i quiz generati dal Sistema A con un punteggio medio di 4.89 su 5 per il criterio "Coerenza e Correttezza", un riscontro quasi perfetto che si allinea con l'alta performance registrata nelle metriche quantitative. Al contrario, il Sistema B ha ottenuto solo 2.89 su 5 sullo stesso criterio, confermando una percepita inaffidabilità.

5.4.2 Discussione sui Trade-off Architetturali

Il confronto ha messo in luce i classici trade-off tra un'architettura cloud e una locale, ma con implicazioni pratiche molto nette per questo specifico caso d'uso.

- **Qualità vs. Costo e Privacy**: Il Sistema A (Cloud) offre una qualità dell'output ineguagliabile, ma introduce costi operativi legati all'utilizzo delle API e richiede una connessione a internet, con implicazioni per la privacy dei dati. Il Sistema B (Offline), d'altra parte, è completamente gratuito dopo l'implementazione iniziale, funziona offline e garantisce che i dati dell'utente non lascino mai il dispositivo. Tutta-

via, come dimostrato, questo avviene a un costo altissimo in termini di qualità e affidabilità.

- **Performance e User Experience**: La scoperta più critica riguarda la latenza. Come mostrato nella Tabella 5.2, il Sistema A risponde in media in circa **1.39 secondi**, un tempo assolutamente accettabile per un'interfaccia conversazionale. Il Sistema B, invece, richiede in media **29.96 secondi** per generare una singola risposta. Una latenza di questo tipo rende l'esperienza utente frustrante e quasi impraticabile, vanificando i vantaggi teorici dell'approccio offline.

- **Scalabilità e Manutenzione**: Come emerso durante i test, il sistema locale presenta significative difficoltà di gestione: l'indicizzazione di nuovi documenti è un processo lento e ad alto consumo di risorse, che può fallire con file di dimensioni realistiche. Il sistema cloud, invece, è intrinsecamente più scalabile e la gestione della base di conoscenza è demandata a servizi ottimizzati e robusti.

5.4.3 Validazione delle Scelte Progettuali di "MyStudyApp"

In conclusione, i risultati sperimentali validano in modo conclusivo le scelte architetturali alla base di "MyStudyApp". L'obiettivo primario di questa tesi era progettare e sviluppare un'applicazione che non fosse solo una novità funzionale, ma che si proponesse come uno strumento di studio *efficace, affidabile e piacevole da usare.*

I dati dimostrano in modo inequivocabile che, nonostante i vantaggi teorici dell'approccio locale in termini di costi e privacy, i suoi limiti pratici in termini di qualità delle risposte e, soprattutto, di performance (latenza), lo rendono inadeguato per raggiungere tale obiettivo. Un'esperienza utente di alta qualità, caratterizzata da risposte accurate e reattive, è un requisito fondamentale per un'applicazione di questo tipo.

Pertanto, i risultati confermano che la scelta di un'architettura cloud-based, sebbene più complessa da implementare e soggetta a costi operativi, è stata quella strategica giusta per costruire un prodotto in grado di colmare con successo il gap funzionale identificato, offrendo agli utenti uno strumento di studio realmente utile e performante.

Conclusioni

Il presente lavoro di tesi si è posto l'obiettivo di esplorare l'applicazione delle architetture di Retrieval-Augmented Generation (RAG) per creare uno strumento di studio innovativo, in grado di superare i limiti di affidabilità e specificità dei tradizionali Large Language Models. In questo capitolo conclusivo, si intende ripercorrere sinteticamente il percorso intrapreso, consolidare i risultati ottenuti e delineare le prospettive per futuri sviluppi.

Sintesi del Percorso di Tesi

Il percorso progettuale ha avuto inizio con un'approfondita analisi dello stato dell'arte, che ha esaminato il panorama della Document AI e l'evoluzione dei modelli linguistici, identificando nei sistemi RAG la soluzione più promettente per ancorare la generazione di testo a una base di conoscenza esterna e verificabile. Questa indagine preliminare ha permesso di individuare una significativa lacuna nel mercato delle applicazioni per studenti: la mancanza di uno strumento che integrasse in un unico ecosistema un ciclo di apprendimento completo, dalla gestione dei materiali alla chat contestuale e all'autovalutazione attiva.

Per rispondere a tale esigenza, è stata progettata e sviluppata l'applicazione mobile nativa "MyStudyApp". L'architettura del sistema, basata su un robusto modello client-server, è stata implementata sfruttando i servizi cloud di Microsoft Azure per orchestrare una pipeline RAG performante e scalabile. Il lavoro ha coperto l'intero ciclo di sviluppo, dalla progettazione

dell'infrastruttura backend alla realizzazione dell'interfaccia utente Android. Infine, per validare l'efficacia dell'approccio scelto, è stata condotta una rigorosa valutazione sperimentale, confrontando le performance del sistema cloud con quelle di un'implementazione alternativa completamente offline.

Risultati Chiave e Raggiungimento degli Obiettivi

L'analisi sperimentale, descritta nel Capitolo 5, ha fornito risultati inequivocabili che validano con forza le scelte architetturali alla base di "MyStudyApp". Il sistema cloud (Sistema A) ha dimostrato una superiorità netta rispetto all'alternativa offline (Sistema B) in tutte le metriche di valutazione.

I dati quantitativi hanno evidenziato come le risposte generate dal sistema cloud non solo fossero più coerenti e fluide, ma, dato di fondamentale importanza, mostrassero un grado di similarità semantica con le fonti di verità notevolmente più elevato, mitigando drasticamente il rischio di generare informazioni imprecise o errate. Tale superiorità è stata confermata dalla valutazione qualitativa, dove gli utenti hanno promosso i quiz generati dal sistema cloud con punteggi eccellenti, giudicandoli significativamente più pertinenti, corretti e utili per l'apprendimento.

Il divario più critico, tuttavia, è emerso nell'analisi delle performance. Con un tempo di risposta medio di un ordine di grandezza inferiore, l'architettura cloud ha garantito un'esperienza utente reattiva e fluida, requisito imprescindibile per un'applicazione interattiva, a fronte di una latenza dell'alternativa locale che ne ha compromesso la praticabilità d'uso.

I risultati ottenuti permettono quindi di affermare che l'obiettivo primario della tesi è stato pienamente raggiunto. È stato realizzato non solo un prototipo funzionante, ma uno strumento di studio che, grazie alle scelte tecnologiche adottate, si è dimostrato efficace, affidabile e performante, colmando con successo la lacuna funzionale identificata nello stato dell'arte.

Limitazioni del Lavoro

Per completezza scientifica, è importante riconoscere i limiti intrinseci del presente studio, i quali possono offrire a loro volta spunti per future indagini.

In primo luogo, la valutazione qualitativa dell'esperienza utente, sebbene abbia fornito indicazioni chiare, è stata condotta su un campione di partecipanti numericamente contenuto e omogeneo per percorso di studi. Sebbene i risultati emersi siano netti, una validazione su una base di utenti più ampia e diversificata potrebbe ulteriormente rafforzare le conclusioni sull'usabilità percepita dell'applicazione.

In secondo luogo, la valutazione quantitativa delle performance del sistema di Q&A è stata eseguita su un singolo contesto documentale standardizzato per garantire la controllabilità dell'esperimento. Le performance potrebbero teoricamente variare in presenza di documenti con formattazioni estremamente complesse, layout non convenzionali o un linguaggio altamente specialistico non ben rappresentato nei dati di addestramento dei modelli di embedding.

Infine, la scelta di un'architettura cloud, pur essendosi rivelata vincente, introduce una dipendenza intrinseca da servizi di terze parti. Questo implica non solo costi operativi legati all'utilizzo delle API, ma anche una dipendenza dalla disponibilità e dalle policy dei provider, aspetti che un'architettura completamente offline eviterebbe, seppur a scapito delle prestazioni, come ampiamente dimostrato.

Sviluppi Futuri

Il lavoro svolto pone solide fondamenta per numerose estensioni e miglioramenti futuri, volti ad arricchire ulteriormente l'esperienza di apprendimento offerta da "MyStudyApp". Tra le direzioni più promettenti si individuano:

- **Evoluzione verso un'architettura "Bring Your Own..." (BYO) per Sicurezza e Sostenibilità:** Per aumentare il controllo da parte

dell'utente e garantire la viabilità del progetto, un'evoluzione strategica consisterebbe nell'adottare un paradigma "Bring Your Own".

  - Dal punto di vista della **sicurezza**, si potrebbe implementare un modello BYOK (Bring Your Own Key), che permetterebbe agli utenti di gestire le proprie chiavi di crittografia per i dati archiviati, massimizzando la privacy.
  - Dal punto di vista della **sostenibilità economica**, in modo analogo, si potrebbe introdurre un'opzione "BYO-AI", consentendo agli utenti di inserire la propria chiave API personale per i servizi Azure OpenAI. Questo assocerebbe i costi computazionali direttamente al loro account, eliminando il principale ostacolo economico alla scalabilità dell'applicazione.

  Questa duplice strategia non solo rafforzerebbe la fiducia nell'applicazione, ma ne assicurerebbe la sostenibilità a lungo termine, permettendone una distribuzione su vasta scala.

- **Estensione delle Funzionalità di Apprendimento Attivo:** Oltre alla chat e ai quiz, l'applicazione potrebbe essere potenziata con la capacità di generare automaticamente riassunti concisi o flashcard digitali dai documenti, offrendo così ulteriori modalità di ripasso e memorizzazione.

- **Supporto a Nuovi Formati di File:** Per aumentare la versatilità dello strumento, si potrebbe estendere il supporto a formati di file comuni nel mondo accademico e professionale, come le presentazioni PowerPoint (`.pptx`) o le immagini contenenti testo, integrando un sistema di Optical Character Recognition (OCR).

- **Avanzamenti della Pipeline RAG:** La pipeline di recupero delle informazioni potrebbe essere affinata implementando tecniche più avanzate, come la *hybrid search* (che combina ricerca semantica e a

parole chiave) o l'aggiunta di un modello di *re-ranking*, per migliorare ulteriormente la pertinenza dei contesti forniti al LLM.

- **Potenziamento della Gamification:** Per incentivare un utilizzo costante e rendere lo studio più coinvolgente, gli elementi di gamification potrebbero essere ampliati introducendo obiettivi di apprendimento, badge per il raggiungimento di traguardi e statistiche di performance ancora più dettagliate.

- **Sviluppo Cross-Platform:** Infine, per raggiungere un'utenza più vasta, si potrebbe considerare lo sviluppo di una versione dell'applicazione per il sistema operativo iOS o la transizione verso un framework di sviluppo cross-platform.

Bibliografia

[1] Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. Dense x retrieval: What retrieval granularity should we use? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15159–15177, 2024.

[2] Lei Cui, Yiheng Xu, Tengchao Lv, and Furu Wei. Document ai: Benchmarks, models and applications. *arXiv preprint arXiv:2111.08609*, 2021.

[3] Graham A Cutting and Anne-Françoise Cutting-Decelle. Intelligent document processing–methods and tools in the real world. *arXiv preprint arXiv:2112.14070*, 2021.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[5] Stefan Feuerriegel, Jochen Hartmann, Christian Janiesch, and Patrick Zschech. Generative ai. *Business & Information Systems Engineering*, 66(1):111–126, 2024.

[6] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-

augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023.

[7] Yikun Han, Chunjiang Liu, and Pengfei Wang. A comprehensive survey on vector database: Storage and retrieval technique, challenge. *arXiv preprint arXiv:2310.11703*, 2023.

[8] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[9] Sotirios Kastanas, Shaomu Tan, and Yi He. Document ai: A comparative study of transformer-based, graph-based models, and convolutional neural networks for document layout analysis. *arXiv preprint arXiv:2308.15517*, 2023.

[10] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, 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.

[11] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[12] Ggaliwango Marvin, Nakayiza Hellen, Daudi Jjingo, and Joyce Nakatumba-Nabende. Prompt engineering in large language models. In *International conference on data intelligence and cognitive informatics*, pages 387–402. Springer, 2023.

[13] Koji Osawa. Integrating automated written corrective feedback into e-portfolios for second language writing: Notion and notion ai. *Relc Journal*, 55(3):881–887, 2024.

[14] Subhajit Panda. Enhancing pdf interaction for a more engaging user experience in library: Introducing chatpdf. *IP Indian Journal of Library Science and Information Technology*, 8(1):20–25, 2023.

[15] Vipula Rawte, Amit Sheth, and Amitava Das. A survey of hallucination in large foundation models. *arXiv preprint arXiv:2309.05922*, 2023.

[16] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[17] 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.

[18] Hayden Wolff. Rag 101: Demystifying retrieval-augmented generation pipelines, 12 2023.

Ringraziamenti

Al termine di questo importante percorso di studi, desidero rivolgere un pensiero di gratitudine a tutti coloro che hanno reso possibile il raggiungimento di questo traguardo. Per prima cosa vorrei ringraziare il mio relatore, il Professor Federico Montori, per la professionalità e la disponibilità dimostrate. I suoi consigli e la sua fiducia mi hanno permesso di affrontare questo lavoro con stimolo e autonomia.

Un ringraziamento speciale va ai miei genitori, che fin dal primo momento hanno approvato e supportato (anche economicamente) la mia scelta di intraprendere la vita da fuorisede. Il loro sostegno mi ha concesso di vivere con serenità la mia esperienza universitaria, esulandomi da tante preoccupazioni e consentendomi di concentrarmi al meglio su questo percorso. Senza di loro, nulla di tutto questo sarebbe stato possibile. Estendo la mia gratitudine a tutta la mia famiglia, per l'affetto e l'incoraggiamento che non mi hanno mai fatto mancare.

Ai miei amici di lunga data con cui il rapporto non è cambiato di una virgola pur non vedendoci o non sentendoci per mesi, agli amici e ai colleghi conosciuti durante questo nuovo percorso. Un grazie particolare a chi mi ha sopportato quotidianamente, anche via chat, ai miei consolidati "compagni di progetto" con cui ho condiviso infinite ore di debug e ai miei coinquilini con cui le giornate non sono mai noiose. Infine, un grazie a chiunque in un modo o in un altro ha fatto parte della mia vita rendendomi chi sono oggi.

A tutti voi, grazie.

-Ciccio