



UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Progetto di Text Mining

Descrizione Architetture, Modelli Utilizzati e Performance

Anno Accademico 2025/2026

Professori

Prof. Amato

Prof. Moccardi

Ing. Ghosh

Gruppo

Fabio Accurso M63001723

Vittoria Alberto M63001750

Benito Cervone M63001747

Descrizione Architetture

Il sistema sviluppato è una piattaforma di Retrieval-Augmented Generation (RAG), progettata per analizzare e interrogare corpus legali relativi a Divorzio e Successioni per tre giurisdizioni distinte: Italia, Estonia e Slovenia. L'architettura è modulare e segue una pipeline rigorosa che va dalla configurazione dell'ambiente alla generazione della risposta finale.

Gestione della Configurazione e Ingestion dei Dati

Il cuore della configurazione risiede nel modulo `backend/config.py`, dove la classe `RAGConfig` centralizza parametri critici come i percorsi dei dati, le chiavi API (gestite tramite file `.env`), i parametri di generazione e la gestione dei parametri operativi dei modelli, quali la temperatura di generazione e il valore `top_k` per il retrieval. Per garantire la portabilità del progetto su diversi sistemi operativi e facilitare il deployment in ambienti containerizzati come **Docker**, la gestione dei percorsi è implementata in modo dinamico e *OS-agnostic* utilizzando la libreria `pathlib` e definendo una `PROJECT_ROOT` relativa. Il processo di caricamento dei documenti è gestito dal modulo `document_loader.py`, il quale affronta la sfida dell'eterogeneità delle fonti date. I documenti originali sono forniti in formato JSON. Il modulo implementa una logica di normalizzazione che mappa attributi variabili (come “type” o “state”) in una tassonomia unificata

composta dai campi standard `country`, `law` e `doc_type`. Questa standardizzazione è preliminare e necessaria per garantire un retrieval efficace e filtrabile, permettendo di distinguere nettamente tra normative ("Codes") e giurisprudenza ("Cases").

Embeddings e Vector Database Granulari

A differenza di approcci tradizionali che tendono a creare indici monolitici o partizionati per nazione, la strategia di indicizzazione implementata nello script `build_vector_stores.py` adotta una segregazione basata sulla tipologia di contenuto giuridico. Vengono costruiti quattro *Vector Store* distinti e granulari:

`divorce_codes`, `divorce_cases`, `inheritance_codes` e `inheritance_cases`.

Questa separazione permette agli agenti di effettuare ricerche mirate, riducendo il rumore semantico. Permette quindi di isolare semanticamente i codici normativi (codici) dalla giurisprudenza (casi passati) e i domini del diritto (divorzio vs successioni), migliorando la precisione semantica durante la fase di interrogazione. La tecnologia sottostante utilizzata per l'indicizzazione e la ricerca di similarità è FAISS (Facebook AI Similarity Search), scelta per la sua efficienza nella gestione di vettori densi.

Orchestrazione e Utility di Routing

Il funzionamento operativo del sistema è coordinato da due moduli trasversali che gestiscono il flusso delle richieste e forniscono funzionalità condivise di analisi semantica.

Entry Point e Dispatching (Pipeline) Il modulo `rag_pipeline.py` funge da unico punto di ingresso (*entry point*) per l'interfaccia utente. La funzione principale `answer_question` astrae la complessità sottostante, verificando la configurazione attiva (tramite il flag `use_multiagent`) e indirizzando dinamicamente la chiamata verso il motore logico appropriato: l'architettura *Single-Agent* o il *Supervisore Multi-Agent*. Questo design pattern permette di commutare istantaneamente tra le due strategie senza modificare il codice del frontend.

Funzioni di Supporto al Retrieval (Utils) Il modulo `rag_utils.py` centralizza la logica di pre-elaborazione e routing intelligente, fornendo strumenti essenziali per entrambe le architetture:

- **Estrazione Filtri e Metadati:** La funzione `_extract_metadata_filters` analizza la query in linguaggio naturale per generare pre-filtri sui metadati, riducendo lo spazio di ricerca prima ancora di interrogare i vector store.
- **Descrizione Dinamica dei DB:** La funzione

`_describe_databases` genera descrizioni testuali dei contenuti presenti nei database vettoriali (campionando casualmente i documenti per evitare bias). Queste descrizioni sono fondamentali per permettere all'LLM di comprendere quale database sia più pertinente.

- **Decision Making (Routing):** La funzione

`_decide_which_dbs` implementa la logica di routing vera e propria. Costruisce un prompt che presenta al modello le descrizioni dei database disponibili e la domanda dell'utente, delegando all'LLM la scelta di quali *Vector Store* consultare (mappando, ad esempio, una domanda sul "divorzio in Italia" ai soli DB pertinenti ed escludendo quelli su successioni o altri paesi).

Pipeline di Generazione: Single-Agent vs Multi-Agent

Il sistema supporta due paradigmi di generazione distinti, permettendo una valutazione comparativa delle strategie di reasoning.

Single-Agent (ReAct) L'architettura a singolo agente implementa una logica di tipo ReAct (*Reasoning and Acting*). Il flusso di esecuzione è governato da un *Router* logico che analizza l'intent dell'utente per discriminare tra query che necessitano di accesso alla knowledge base giuridica e query di natura generale (*general knowledge*). Nel

caso sia necessario il retrieval, l'agente esegue una ricerca globale sui vector store, seguita da una fase opzionale di *re-ranking* per raffinare il contesto prima della generazione della risposta.

Multi-Agent (Supervisor) L'architettura Multi-Agent introduce un livello di astrazione superiore mediante un agente Supervisore. Questo componente orchestra l'accesso ai dati agendo come un dispatcher intelligente: analizza la query e attiva esclusivamente i *tools* (ovvero i vector store specializzati) pertinenti alla richiesta. Il Supervisore ha la responsabilità di aggregare le informazioni provenienti dai diversi sotto-agenti specializzati e di sintetizzare una risposta coerente, gestendo efficacemente scenari complessi che richiedono il confronto tra normative di diverse giurisdizioni.

Logging e Valutazione Automatica

Per garantire la trasparenza e la valutabilità del sistema, è stato implementato nel modulo `Chatbot.py` un meccanismo di logging che archivia ogni interazione in formato JSON, salvando la query dell'utente, la risposta generata e, crucialmente, i riferimenti ai documenti sorgente grezzi utilizzati come contesto.

La valutazione qualitativa del sistema è automatizzata tramite il modulo `Evaluation.py`, il quale integra il framework RAGAS. Utilizzando un LLM giudice, il modulo calcola metriche oggettive quali la

Faithfulness (fedeltà della risposta al contesto recuperato) e l'*Answer Relevancy* (pertinenza della risposta rispetto alla domanda), fornendo una stima quantitativa delle performance delle due architetture. Per ciascuna query per la quale viene fornita la Ground Truth il sistema calcola anche le metriche di Context Precision, Context Recall e Answer Correctness confrontando le risposte generate con la risposta ideale fornita.

Modelli Utilizzati

La selezione dei modelli di intelligenza artificiale è stata guidata da un'analisi dei compromessi tra prestazioni, costi computazionali e capacità di ragionamento. Di seguito si riportano le specifiche dei modelli adottati, come definiti nel file di configurazione del sistema.

Embedding Model

Modello: `sentence-transformers/all-MiniLM-L6-v2`

Provider: HuggingFace

Per la fase di vettorizzazione dei documenti e delle query, è stato selezionato il modello `all-MiniLM-L6-v2`. La scelta di questo modello open-source è motivata dalla sua estrema leggerezza ed efficienza, che ne permette l'esecuzione rapida anche su CPU senza richiedere acceleratori hardware dedicati. Nonostante le dimensioni ridotte, il mod-

ello offre prestazioni eccellenti nel task di *semantic search* e garantisce un supporto multilingua di base adeguato al dominio trattato, rappresentando un ottimo punto di equilibrio tra accuratezza del retrieval e onere computazionale.

Generative Model

Modello: `gpt-4o-mini`

Provider: OpenAI

Parametri: Temperature impostata a 0.2

Per il componente generativo e di reasoning (sia per il Single-Agent che per il Multi-Agent), il sistema si avvale del modello `gpt-4o-mini`. Questa scelta strategica deriva dal rapporto costo/efficienza estremamente vantaggioso offerto dal modello. `gpt-4o-mini` garantisce una latenza ridotta, essenziale per un'applicazione interattiva, e possiede capacità di ragionamento logico sufficientemente avanzate per gestire compiti complessi come il routing delle query e la sintesi legale, mantenendo i costi di utilizzo delle API significativamente inferiori rispetto ai modelli *flagship* della serie GPT-4.

Evaluation Layer

Per la fase di valutazione automatica delle performance del sistema RAG, gestita tramite il framework *Ragas*, sono stati adottati modelli specifici forniti da OpenAI, al fine di garantire un "Giudice" esterno

di elevata capacità logica e affidabilità standardizzata.

- **Evaluation LLM (LLM-as-a-Judge)**

Modello: `gpt-4o-mini`

Provider: OpenAI

Parametri: Temperature impostata a 0.

Motivazione: L'utilizzo del pattern LLM-as-a-Judge richiede un modello con forti capacità di ragionamento per valutare metriche complesse come Faithfulness e Context Precision. `gpt-4o-mini` è stato scelto come compromesso ideale tra costi contenuti, velocità di esecuzione e accuratezza analitica, spesso comparabile a modelli maggiori in task di valutazione strutturata. La temperatura a zero garantisce il determinismo e la riproducibilità dei punteggi di valutazione

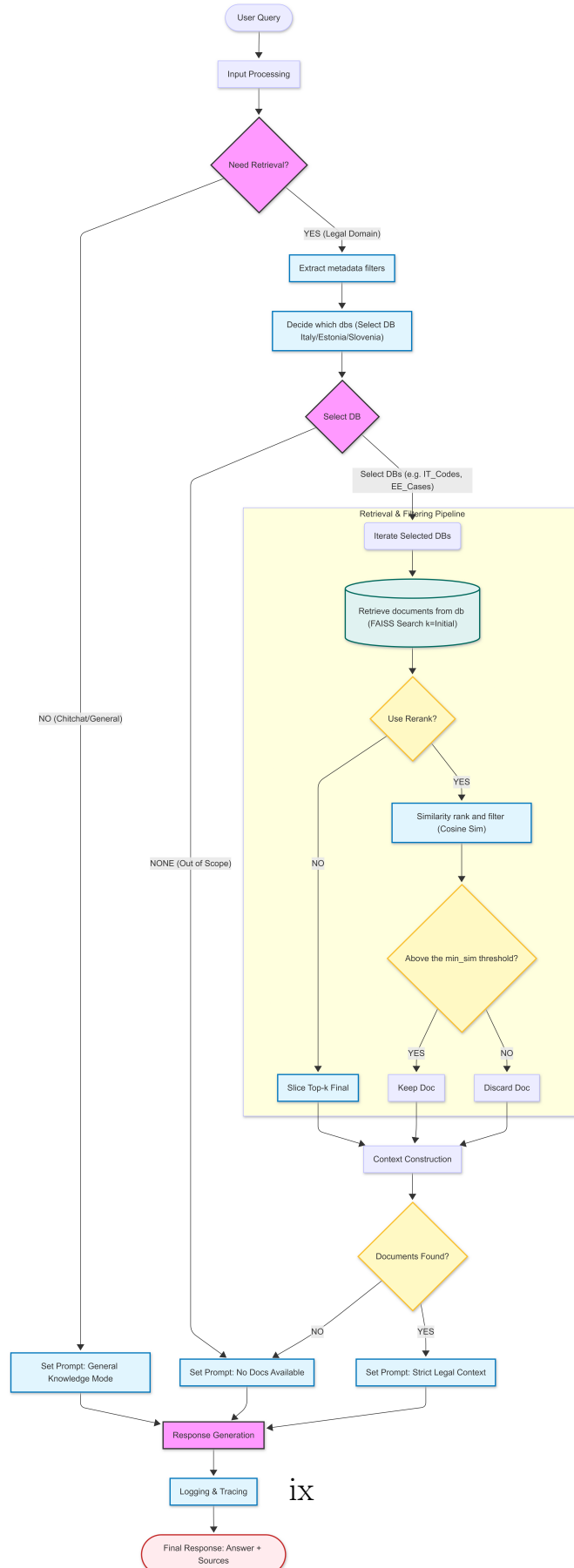
- **Evaluation Embeddings**

Modello: `text-embedding-3-small`

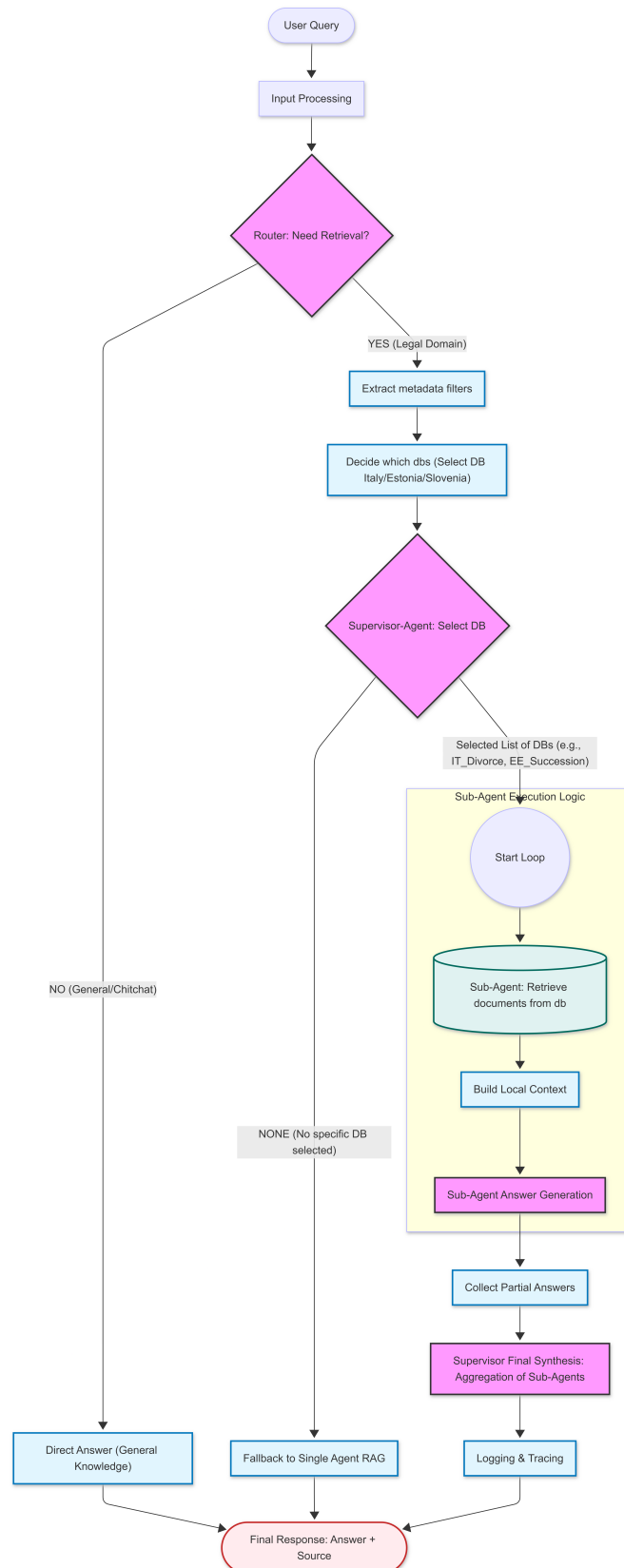
Provider: OpenAI

Motivazione: Questo modello è impiegato specificamente per il calcolo delle metriche vettoriali all'interno di Ragas (come Answer Relevancy), fornendo rappresentazioni semantiche robuste e allineate con lo standard industriale per la valutazione della similarità testuale.

Flowchart Single-Agent



Flowchart Multi-Agent



Analisi Comparativa delle Strategie Architet- turali

Questa sezione presenta un confronto critico tra le due architetture implementate: il sistema a Agente Singolo con logica ReAct e il sistema Multi-Agente orchestrato da un Supervisore. La valutazione si basa sulle metriche del framework RAGAS ottenute da un set di 5 query controllate, che spaziano da domande fuori dominio (es. ricetta della carbonara) a complessi confronti inter-giurisdizionali (es. differenze di divorzio tra Italia e Slovenia).

Analisi delle Metriche RAGAS





I risultati quantitativi, riassunti nella tabella delle prestazioni, rivelano una divergenza significativa nelle performance tra i due approcci.

- **Answer Relevancy e Correctness:** Il sistema Multi-Agente dimostra prestazioni nettamente superiori con una *Answer Relevancy* aggregata di **0.86** e una *Answer Correctness* di **0.70**, rispetto ai punteggi dell'Agente Singolo rispettivamente di **0.35** e **0.37**. È fondamentale notare come l'Agente Singolo abbia subito un crollo catastrofico nella rilevanza (punteggio 0) sulle query complesse e comparative (Query 3, 4 e 5). Ciò suggerisce che, sebbene l'agente singolo fosse in grado di recuperare informazioni, il ciclo ReAct ha faticato a sintetizzare una risposta coerente di fronte a scenari legali sfaccettati o confronti tra paesi.
- **Prestazioni di Retrieval (Context Recall & Precision):** Entrambi i sistemi hanno mantenuto standard di recupero comparabili. L'Agente Singolo ha ottenuto una *Context Recall* di

0.77 e una *Context Precision* di **0.64**, mentre il Multi-Agente ha registrato rispettivamente **0.80** e **0.63**. Questo indica che il problema principale dell'Agente Singolo non risiedeva nell'incapacità di trovare i documenti, ma nella minor capacità di utilizzarli efficacemente nella fase di generazione.

- **Faithfulness:** L'Agente Singolo ha ottenuto un punteggio di fedeltà leggermente superiore (**0.69**) rispetto al Multi-Agente (**0.56**). Questa discrepanza può essere attribuita al passaggio di aggregazione del Supervisore, che sintetizza le risposte parziali degli agenti specializzati. Tale processo potrebbe astrarre leggermente dal testo sorgente diretto rispetto a una generazione immediata, pur risultando in una risposta finale molto più corretta e pertinente.

Valutazione Critica e Scelta Strategica

I risultati sperimentali evidenziano i limiti di un approccio ReAct monolitico quando applicato a un dataset legale multinazionale eterogeneo. L'architettura a Agente Singolo, sebbene più semplice da implementare, ha fallito nel mantenere un ragionamento coerente attraverso query complesse che richiedevano la distinzione tra contesti specifici (Italia vs Slovenia vs Estonia). I bassi punteggi di rilevanza indicano che l'agente si è spesso "perso" nella traccia di ragionamento o non è riuscito a rispettare i vincoli del prompt.

Al contrario, l'**architettura Multi-Agente con Supervisore** si è dimostrata la strategia più robusta. Decomponendo lo spazio del problema—instradando domande legali specifiche ad agenti specializzati —il sistema ha garantito che il modello generativo operasse all'interno di un contesto più pulito e focalizzato. Questo approccio "divide et impera" ha prevenuto l'inquinamento del contesto e le allucinazioni, come evidenziato dall'alta correttezza delle risposte.

Conclusioni sulla Scalabilità ed Estensioni Future

Sulla base sia delle metriche quantitative che delle osservazioni qualitative, la **strategia Multi-Agente con Supervisore** è identificata come l'approccio più promettente per una futura scalabilità.

- **Scalabilità:** L'aggiunta di un nuovo paese (es. Belgio) o di un nuovo dominio legale (es. Diritto Commerciale) in un sistema multi-agente richiede solo l'aggiunta di un nuovo agente specializzato e l'aggiornamento della descrizione del Supervisore. In un sistema a agente singolo, ciò aumenterebbe esponenzialmente la complessità del prompt ReAct e dello spazio di ricerca vettoriale, degradando probabilmente le prestazioni.
- **Robustezza:** Il Supervisore agisce come un efficace router semantico, gestendo la complessità della query prima che avvenga il recupero. Questo assicura che una query sul "Divorzio Italiano" non venga mai confusa con la "Successione Estone", un

requisito cruciale per un assistente legale affidabile.

Pertanto, l'architettura Multi-Agente è la scelta selezionata per gestire il dataset di diritto civile multinazionale, offrendo il miglior bilanciamento tra precisione del recupero, qualità della risposta ed estensibilità architetturale.