UNIVERSITÀ DEGLI STUDI DI PALERMO
DIPARTIMENTO DI INGEGNERIA
CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

Paper Review

Documentazione LLM

Studenti
Leonardo Giovanni Caiezza
Diego Corona
Luca Gaetani
Daniele Orazio Susino

Docente
Prof.ssa Valeria Seidita

Anno accademico
2024 - 2025

Indice

1 Introduzione

1.1 Scopo del progetto

Questo progetto nasce per fornire un servizio API in grado di automatizzare l'estrazione di parole chiave da documenti PDF. L'obiettivo è semplificare il processo di analisi testuale, rendendolo accessibile tramite una semplice chiamata HTTP. Attraverso l'uso di un modello LLM (LLaMA 3) eseguito localmente, l'API analizza il contenuto del documento e restituisce, se possibile, una selezione ragionata di parole chiave scelte da una lista predefinita.

1.2 Funzionalità principali

Il sistema consente di caricare un file PDF, specificare una lista di parole chiave e ricevere in risposta un array JSON con i tag più coerenti rispetto al contenuto del testo. Il processo è completamente automatizzato: il file viene letto, il testo estratto e inviato a un modello linguistico per l'analisi semantica. L'output viene poi filtrato, validato e restituito all'utente in un formato strutturato.

1.3 Tecnologie usate

Il progetto è costruito su FastAPI, un framework asincrono per API web che offre prestazioni elevate e una gestione chiara degli endpoint. Per l'elaborazione linguistica viene utilizzato Ollama, che permette l'esecuzione locale di modelli LLM, in questo caso LLaMA 3. L'estrazione del testo dai PDF è affidata a PyMuPDF (fitz), una libreria potente e affidabile per questo tipo di operazioni. L'intera infrastruttura è basata su Python, sfruttando librerie standard per la gestione dei file e la manipolazione dei dati.

1.4 Requisiti hardware

Per eseguire localmente i modelli LLM tramite `Ollama` (come `LLaMA 3`), è necessario disporre di una scheda video **NVIDIA compatibile con CUDA**. Questo perché:

- I modelli di grandi dimensioni richiedono elevate capacità computazionali, in particolare durante la fase di inferenza.
- `CUDA` (Compute Unified Device Architecture) è una piattaforma di calcolo parallelo sviluppata da `NVIDIA`, che permette di sfruttare le GPU per accelerare le operazioni matematiche, migliorando drasticamente i tempi di risposta rispetto all'uso della sola CPU.
- `LLaMA 3` e modelli simili sono ottimizzati per funzionare in ambienti *GPU-accelerated*, dove possono sfruttare il parallelismo massiccio offerto dalle GPU compatibili con `CUDA`.

Senza una scheda `NVIDIA` compatibile, l'esecuzione dei modelli LLM può essere drasticamente più lenta o addirittura impossibile, a seconda delle risorse disponibili. Per questo motivo, si consiglia una **GPU con almeno 8 GB di VRAM** e **supporto CUDA 11 o superiore**, per garantire prestazioni fluide e reattive nell'uso dell'API.

2 Panoramica delle scelte progettuali

2.1 Linguaggio di programmazione: Python

La scelta di utilizzare Python è stata guidata dalla sua flessibilità, semplicità e dall'ampia disponibilità di librerie mature per ogni parte del flusso: dalla gestione dei file (tempfile, os), alla lettura dei PDF (PyMuPDF), fino alla creazione di API (FastAPI) e all'interfaccia con modelli linguistici (ollama).

Python è inoltre il linguaggio di riferimento per la maggior parte dei progetti di intelligenza artificiale, data science e automazione, e si integra facilmente con ambienti in cui è richiesto un rapido ciclo di sviluppo, test e iterazione. La sua sintassi chiara e leggibile ha permesso di mantenere il progetto semplice da comprendere e facilmente estendibile, anche da parte di team eterogenei o futuri collaboratori.

2.2 Libreria API: FastAPI

FastAPI è stato scelto per la sua architettura moderna, il supporto completo all'asincronia e la facilità con cui consente di definire endpoint robusti e validati.

2.3 Libreria LLM: Ollama

La scelta di Ollama è stata guidata dal desiderio di eseguire il modello LLM in locale, evitando l'utilizzo di API esterne e garantendo maggiore controllo sui dati. Ollama fornisce un'interfaccia semplice per l'esecuzione di modelli come LLaMA 3, permettendo analisi testuali avanzate senza dipendere da connessioni cloud o credenziali esterne. LLaMA 3 è stato scelto per la sua capacità di comprendere il contesto e generare output coerenti anche in scenari multilingua.

2.4 Libreria PDF: Fitz

La libreria fitz, parte del pacchetto PyMuPDF, è stata selezionata per la sua affidabilità e precisione nell'estrazione di testo da file PDF. A differenza di soluzioni più semplici, PyMuPDF è in grado di gestire documenti multi-pagina, layout complessi e testi disposti in colonne o blocchi, offrendo un'estrazione coerente e strutturata. Questo la rende particolarmente adatta a essere integrata in pipeline di elaborazione semantica successive. Il testo estratto viene infine convertito in formato JSON, così da garantire una rappresentazione chiara, organizzata e facilmente manipolabile dal punto di vista programmatico.

2.5 Formato risposte: JSON

Il formato JSON è stato adottato per la sua diffusione e per la semplicità con cui può essere interpretato da client e sistemi terzi. L'output dell'API è sempre un array JSON di stringhe, che rappresentano le parole chiave individuate nel testo. Questo formato garantisce compatibilità con quasi tutti i linguaggi di programmazione e framework di integrazione.

2.6 Dipendenze varie

Le dipendenze necessarie al funzionamento del progetto sono riportate nel file requirements.txt. Tra le librerie fondamentali si trovano fastapi, uvicorn, ollama e PyMuPDF, oltre a strumenti di supporto come python-multipart per la gestione degli upload e pydantic per la validazione dei dati. Si consiglia l'uso di un ambiente virtuale per installare queste dipendenze in modo isolato.

3 Descrizione delle Rotte

Di seguito si riporta l'unica rotta dell'applicazione, con i relativi parametri ed un esempio di risposta.

3.1 `/tag-pdf`

Metodo HTTP	`POST`
Rotta	`/tag-pdf`
Parametri	`file` (Upload PDF), `parole_chiave` (JSON stringhificato con lista di parole)
Descrizione	Analizza il contenuto testuale di un file PDF e restituisce un array JSON contenente al massimo due parole chiave selezionate tra quelle fornite.
JSON di risposta	`{ "tags": ["scienza", "tecnologia"] }`

3.1.1 Parametri accettati

La richiesta accetta due parametri tramite `multipart/form-data` (è un tipo di codifica del corpo della richiesta HTTP, usata principalmente quando si devono inviare file insieme ad altri dati. Diversamente da application/x-www-form-urlencoded (dove i dati vengono codificati come stringhe), multipart/form-data divide i contenuti in più "parti", ognuna con i propri header e corpo. Questo è necessario, ad esempio, per caricare file in un form web o via curl).
Il primo parametro è `file`, che rappresenta il PDF da analizzare.
Il secondo parametro è `parole_chiave`, un JSON "stringhificato" che contiene la lista di parole chiave. Quest'ultima viene convertita internamente in una lista vera e propria, da cui il modello potrà scegliere. Esempio di lista accettata: ["economia", "tecnologia", "diritto"].

3.1.2 Formato della richiesta e della risposta

La richiesta deve essere costruita come `multipart/form-data` e può essere testata facilmente anche tramite `curl` (è un comando da terminale molto potente usato per fare richieste HTTP, il nome deriva da "Client URL", permette di comunicare con server web direttamente da riga di comando, senza bisogno di un browser). In risposta, il server restituirà un oggetto JSON che include l'array dei tag trovati, oppure un errore se il modello non ha prodotto un output valido.

3.1.2.1 Esempio di risposta positiva

```
{
  "tags": ["tecnologia", "economia"]
}
```

3.1.2.2 Esempio di risposta in caso di errore

Ad esempio, se si verifica un problema nel parsing dell'output generato dal modello, il server restituirà un errore HTTP 500 con un messaggio utile per la diagnostica:

```
{
  "error": "Errore di parsing JSON",
  "risposta_modello": "Testo non riconosciuto come array"
}
```

3.1.2.3 Codici di stato e messaggi di errore

Il sistema può rispondere con diversi codici HTTP a seconda dell'esito dell'operazione. Il codice `200 OK` indica che la richiesta è stata elaborata con successo e che i tag sono stati restituiti correttamente. Un errore `500 Internal Server Error` segnala un problema interno, come ad esempio una mancata interpretazione del PDF o un output errato da parte del modello linguistico.

4 Copyright e diritto d'autore

La presente **documentazione** è protetta dalle **leggi sul diritto d'autore**. Nessuna parte di questo documento può essere riprodotta, distribuita o trasmessa in alcuna forma o con alcun mezzo, elettronico o meccanico, inclusa la fotocopia, la registrazione o altri sistemi di memorizzazione o recupero di informazioni, senza il **previo consenso scritto degli autori**.