Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il management

LLM per estrazione e classificazione di esercizi scolastici in PDF: analisi sperimentale ed integrazione in un'applicazione web

Relatore:
Chiar.mo Prof.
ANGELO DI IORIO

Presentata da:
ALESSIO MANIERI

III Sessione
Anno Accademico 2024/2025

"Diventa più facile.
Ogni giorno diventa un po' più facile.
Ma devi farlo ogni giorno.
Questa è la parte difficile.
Ma diventa più facile."

— Jogging baboon, *BoJack Horseman*

Abstract

Il progetto nasce dall'esigenza di rendere accessibili materiali didattici a bambini della scuola primaria con difficoltà di apprendimento o disturbi specifici, con l'obiettivo di favorire una didattica più inclusiva e personalizzata. La tesi ha come obiettivo la creazione di un sistema in grado di convertire automaticamente file PDF nativi contenenti esercizi scolastici in pagine HTML coerenti e semanticamente corrette tramite LLM.

Il lavoro proposto si concentra sulla realizzazione di due *pipeline*. La prima sfrutta direttamente modelli linguistici multimodali, inviando il documento in forma visiva per ricostruirne struttura, gerarchia e layout, sfruttando librerie di Python per la sola estrazione delle immagini. La seconda integra una fase preliminare di analisi del PDF, strutturazione in JSON e invio al modello. Il test di entrambi flussi mira ad un sistema automatizzato in grado di ricostruire la logica del documento, convertirlo in linguaggio HTML mantenendo il più possibile la fedeltà visiva, la leggibilità del contenuto e riconoscendo e classificando le tipologie di esercizio.

L'approccio sperimentale è basato su tecniche di *prompt engineering* e rappresenta un'alternativa moderna ai metodi maggiormente utilizzati ad oggi di conversione, tipicamente fondati su coordinate e posizionamenti assoluti. Il progetto, sebbene sia sviluppato in un contesto specifico, è pensato in modo flessibile e indirizzato verso l'estensione futura ad altri formati e finalità nel campo della trasformazione semantica di documenti.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Contesto e panoramica del progetto

Il presente lavoro nasce dall'esigenza di favorire l'accessibilità dei materiali didattici destinati ai bambini della scuola primaria, in particolare a coloro che presentano difficoltà di apprendimento o disturbi specifici. In questo contesto, la possibilità di rielaborare i contenuti dei libri di testo in forma più chiara, leggibile e personalizzabile rappresenta un passo importante verso una didattica realmente inclusiva.

L'ispirazione proviene da studi recenti nel campo dell'educazione accessibile e dell'analisi automatizzata dei contenuti didattici, tra cui il lavoro di Lasheb et al. [8], presentato alla conferenza *IEEE ICALT* nel 2025. Tale studio propone un approccio basato su tecniche di computer vision e deep learning, in particolare modelli *YOLOv10x*, per estrarre automaticamente la struttura degli esercizi scolastici. Il sistema raggiunge prestazioni significative e costituisce uno dei primi tentativi concreti di rendere i materiali educativi più accessibili tramite lintelligenza artificiale.

Il progetto complessivo che ha guidato questa tesi è articolato in due macrofasi:

- **Estrazione**, ovvero l'analisi e la comprensione del contenuto originale del PDF.
- **Adattamento**, ovvero la trasformazione del materiale in una forma più fruibile dai bambini interessati. Ad esempio la scomposizione di uno stesso esercizi su più pagine vista la difficoltà di elaborazione di informazioni molto dense.

Il presente elaborato si concentra sulla prima fase, esplorando metodi innovativi per l'estrazione strutturata di contenuti da PDF nativi con esercizi scolastici. In tale prospettiva, la ricerca si inserisce nel più ampio filone che studia limpiego dei modelli linguistici di grandi dimensioni (LLM) come strumenti di analisi documentale generativa.

Il formato scelto per l'output è l'**HTML**, per la sua natura marcata, semantica e manipolabile. Tra i vantaggi di questo formato:

- rappresentazione in modo chiaro di gerarchie, sezioni ed elementi grafici.
- possibilità di mantenere una buona fedeltà visiva rispetto al documento originale.
- facilitazione di successivi adattamenti: modifiche di layout, ridimensionamento di immagini, personalizzazioni del testo.
- integrazione del contenuto estratto in applicazioni web o sistemi educativi.

Questa scelta rende il sistema non solo adatto al contesto della scuola primaria, ma facilmente estendibile ad altri domini per quanto riguarda la conversione di documenti PDF in modo strutturato.

Per valutare le capacità degli LLM nella ricostruzione del layout dei PDF, sono state progettate due pipeline distinte di interazione con il modello:

- **Pipeline Inline**, che invia direttamente il PDF in forma visiva al modello LLM insieme ai metadati delle immagini estratte tramite PyMuPDF [13];
- **Pipeline JSON**, che introduce una fase preliminare di analisi strutturale tramite PDFPlumber [15], producendo un file JSON dettagliato contenente testo, immagini e coordinate.

Il confronto tra i due approcci consente di analizzare la capacità degli LLM di interpretare documenti complessi, ricostruirne la struttura logica e generare un output HTML coerente, accessibile e il più possibile fedele al layout originale. L'intero studio ha un carattere sperimentale: pur nascendo in un contesto specifico, il sistema è stato progettato con un'attenzione particolare alla flessibilità, affinché possa fungere da base per nuovi scenari applicativi e ricerche future nel campo dell'analisi documentale tramite i modelli di intelligenza artificiale.

1.2 Il formato PDF e le sfide legate alla sua interpretazione

Con l'obiettivo di condividere documenti digitali nasce negli anni '90, il più grande progetto di conversione da carta a digitale "The Camelot Project" da parte di uno dei co-fondatori di Adobe [1]. Il formato PDF (Portable Document Format) vero e proprio nacque nel 1993 e, da quel momento, sarebbe cresciuto sempre più fino a diventare il formato più utilizzato per la diffusione, condivisione e scrittura di documenti digitali. La volontà di essere estremamente versatile nella visualizzazione perfetta e identica su ogni tipologia di dispositivo, ha portato il PDF ad essere una tipologia di documento estremamente rigido e non semantico. Ogni testo, immagine e grafiche non è riconosciuto come tale, bensì come oggetto generico all'interno del documento posizionato mediante delle

coordinate assolute. Per questo motivo la ricerca nell'estrazione di contenuto da questo formato digitale ha costituito un elemento di discussione sempre dibattuto e in evoluzione in relazione alle tecnologie sviluppatesi in parallelo. Si fa una breve panoramica sulla struttura interna del PDF per comprenderne la complessità.

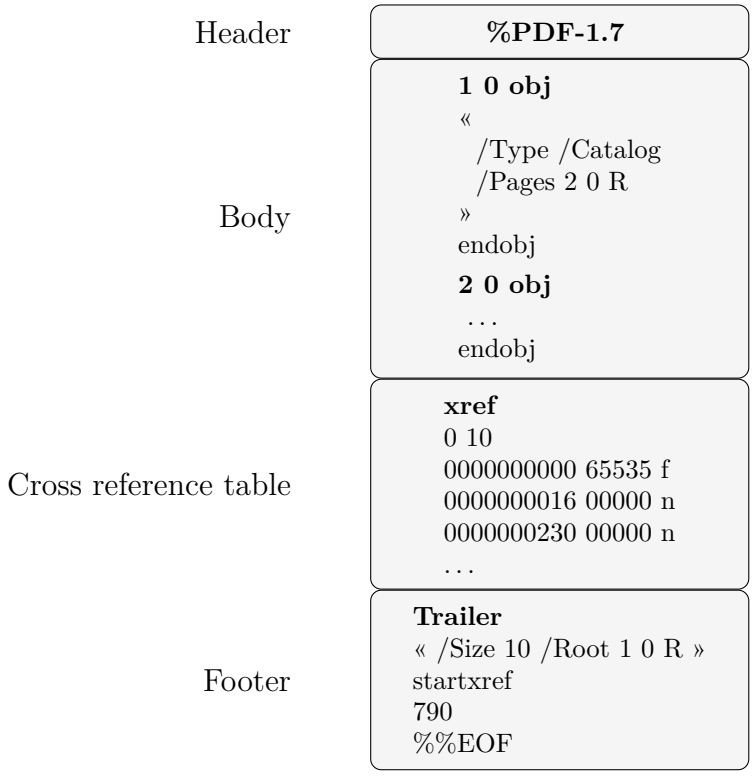

Figura 1.1: Esempio di struttura interna di un file PDF.

- **Header**, esplicita la versione del formato PDF, per permettere al *reader* di sapere quali sintassi e funzionalità aspettarsi.
- **Body**, contiene tutti gli *indirect objects* che descrivono struttura, risorse e contenuto del PDF.
  - `Catalog`, è l'oggetto radice dell'intero documento, non ha contenuti veri e propri e punta a elementi quali: l'albero delle pagine /Pages e il dizionario di font e annotazioni /Names.
  - `Pages`, struttura ad albero che memorizza la lista delle pagine in modo più efficiente

- `Page`, che costituisce ogni pagina del PDF e ne descrive contenuto e risorse. Include la definizione di /Contents, /Resources, /MediaBox che definiscono rispettivamente disegno dei contenuti, risorse (font, colori, immagini) e dimensioni della pagina.
- `Font`, `XObject Image` e `Streams` che definiscono rispettivamente stile dei caratteri, immagini incapsulate e dati binari ad essi associati.
- `Content streams`, che contengono *drawing commands* per il codice grafico utilizzato per il disegno degli elementi del PDF.

La **cross-reference table (xref)** elenca la posizione fisica di ogni oggetto nel PDF tramite gli *offset byte*. Ogni riga indica:

- l'offset dell'oggetto rispetto all'inizio del file;
- il numero di generazione dell'oggetto;
- lo stato dell'oggetto (n attivo, f libero).

Questa struttura permette un accesso diretto agli oggetti senza scansionare l'intero file.

- **Footer**, raggruppa le informazioni minime per permettere al reader di ricostruire correttamente il PDF.
  - `/Size`, il numero totale di entry nella xref;
  - `/Root`, il riferimento al Catalog;
  - `startxref`, la posizione (offset) a cui inizia la tabella xref;
  - `EOF`, il marcatore di fine file.

In sintesi, la struttura interna del PDF, seppur estremamente efficiente per il rendering fedele dell'impaginato su ogni tipologia di dispositivo , introduce un'elevata complessità nella fase di estrazione, ma soprattutto di interpretazione del contenuto. Questa caratteristica intrinseca fa sì che l'interpretazione affidabile di testo, immagini e layout rappresenti ancora oggi una sfida tecnica significativa, al quale si cerca di fornire nuovi strumenti mediante l'intelligenza artificiale generativa.

1.3 Struttura della tesi

La tesi racconta il lavoro svolto analizzandone i suoi aspetti in modo progressivo. Il **capitolo 2** delinea una rassegna dello stato dell'arte per quanto riguarda l'estrazione e la conversione da PDF, citando progetti e strumenti esistenti di diversa tipologia e complessità. Nel **capitolo 3** vengono presentati gli obiettivi che si intendono raggiungere e i casi d'uso che hanno guidato la realizzazione del progetto. Nel **capitolo 4** l'attenzione viene posta sulla descrizione dell'architettura proposta, la struttura del progetto, le tecnologie utilizzate e la costruzione del prompt. Il **capitolo 5** offre i dettagli implementativi del sistema, analizzando il flusso di lavoro. L'analisi dei risultati e la definizione dei limiti risiedono nel **capitolo 6**. Infine il **capitolo 7** che contiene un resoconto del lavoro svolto, in relazione ai risultati ottenuti e delle indicazioni per quanto riguarda gli sviluppi futuri in tal ambito.

Capitolo 2

Stato dell'arte

Vista la complessa struttura interna del formato PDF, i primi strumenti di estrazione di contenuto da questo formato restituivano il testo in *plain text*, non estraendo alcun tipo di informazione semantica e di layout. Nel 2004, con l'avvento di librerie come PDFMiner [14] si sono poste le basi per un'estrazione che introducesse logiche più avanzate di riconoscimento di colonne, blocchi di testo o tabelle. Successivamente si svilupparono strumenti sempre più efficienti quali pyMuPDF [13] e PDFPlumber [15]. Questi strumenti nell'operazione di estrazione di blocchi di testo o immagini, prelevano anche metadati associati, aprendo le porte a nuove possibilità nel riconoscimento del layout e nella comprensione dei componenti. Una strada parallela sviluppatasi è l'utilizzo di *OCR (Optical Character Recognition)* tramite strumenti come Tesseract. Lo strumento in questione analizza pixel per pixel e, tramite delle tecniche di *pattern matching*, restituisce in output un testo modificabile. Tuttavia si tratta di una tecnologia che, nel campo dei PDF, ha senso utilizzare solo nel caso di PDF scannerizzati e non nativi, per cui non è nell'interesse dello studio proposto. I PDF nativi non hanno bisogno di OCR poiché il testo risulta già presente digitalmente nel file, si tratta solo di estrarli in maniera strutturata tramite altre librerie sopra citate. Negli ultimi anni, il focus si è spostato dalla mera estrazione di informazioni dai documenti, ad un problema di comprensione della struttura semantica e logica di essi. Recentemente sono stati pubblicati diversi progetti per l'estrazione di contenuto da PDF che hanno contribuito alla ricerca nel settore.

2.1 Approccio classico all'estrazione da PDF

I primi tentativi di creazione di strumenti completi ed utilizzabili per la ricostruzione di un PDF in HTML miravano al rifacimento in maniera sostanzialmente identica del documento. Nell'ambito degli strumenti tradizionali di estrazione e conversione classica da PDF a HTML è rilevante considerare pdf2html, tutt'ora utilizzato da numerosi *tools* disponibili in rete. pdf2html si tratta di uno strumento *open-source* scritto in C++ e

basato sulle librerie *Poppler* per l'estrazione e *Cairo* per la riproduzione visiva. Il progetto iniziale appartiene a Yao Wei Tjong [12], ed è stato concepito per la conversione in HTML di PDF mantenendo il layout visivo originale in sostanzialmente perfetta. Quello che si ottiene in output è però un HTML in cui ogni elemento ha posizione assoluta, non vi è alcun tipo di gerarchia nè di comprensione semantica del layout.

2.2 Dall'estrazione al riconoscimento del layout

Parallelamente ai progetti di estrazione e conversione in modo *pixel-perfect*, si sviluppano strumenti che ponevano già l'occhio ad un output più semantico. Librerie come PyMuPDF e PDFPlumber, basate sulla già nota PDFMiner, sono utilizzate tutt'ora per ottenere un'estrazione strutturata dei blocchi del file e delle immagini al suo interno. La strada tracciata per lo sviluppo di questo ambito di *document analysis* è quella di ricavare una struttura piuttosto che una riproduzione. Sulla base di ciò cito due progetti completi particolarmente rilevanti per questo scopo, che utilizzano una combinazione di strumenti tradizionali e *machine learning*.

2.2.1 Docling

Docling [5] è un progetto open-source sviluppato dal gruppo AI4K di IBM Research nel 2024, pensato come una pipeline completa per la comprensione strutturale dei documenti digitali. Si basa su una combinazione di modelli multimodali di visione e linguaggio per analizzare e ricostruire il contenuto di file PDF, tra cui il modello di analisi del layout addestrato *DocLayNet* e il riconoscitore di strutture tabellari *TableFormer*. A differenza degli strumenti tradizionali di conversione, Docling mira non solo a preservare la fedeltà visiva ma anche a restituire una rappresentazione semantica coerente del documento, in formato *JSON* o *Markdown*. La pipeline si articola in varie fasi quali: la segmentazione del layout, il riconoscimento delle tabelle e la classificazione dei blocchi testuali.

2.2.2 LayoutLMv3

LayoutLMv3 [7] costituisce la terza generazione della famiglia di modelli *LayoutLM*, sviluppata da Microsoft Research per il campo del *Document AI*. Il modello introduce un'architettura multimodale unificata in grado di elaborare testo, immagini e informazioni geometriche in modo simultaneo. Il progetto prevede una fase di pre-training che utilizza obiettivi combinati di *masked text modeling* e *masked image modeling*, apprendendo così relazioni bidirezionali tra il contenuto testuale e la sua rappresentazione visiva. Utilizza un meccanismo di allineamento tra parole e patch visive (*word-patch alignment*) che migliora la comprensione spaziale del documento.

2.3 L'utilizzo degli LLM nell'analisi di PDF

Il tema dell'intelligenza artificiale è, senza dubbio, divenuto l'argomento principale quando si parla di nuove tecnologie. Lo Stanford AI index 2025 [11] riporta che sulla sfida *SWE-Bench*, che testa la capacità di un modello di risolvere problemi reali su Github, lo stesso modello di AI ha migliorato del 67% le proprie performance in appena 2 anni. Per quanto riguarda il benchmark GPQA (domande di livello post-laurea non risolvibili mediante singola ricerca web) le prestazioni sono passate dal 38.8% all'87.7% tra il 2023 e il 2025, e sono in continua crescita. Alla luce di questi risultati, è naturale ritenere che l'adozione di questa tecnologia sia destinata ad estendersi ad un numero sempre crescente di ambiti. Nel settore dell'analisi documentale l'intelligenza artificiale sta assumendo e assumerà sicuramente sempre più rilevanza. Prendiamo in considerazione qualche progetto che ha fatto uso dell'intelligenza artificiale generativa all'interno dei propri *workflow*.

2.3.1 LayoutLLM

LayoutLLM [9] è un progetto basato sul paradigma multimodale introdotto da LayoutLM, sviluppato da Microsoft Research Asia nel 2024. Il modello combina le capacità di ragionamento linguistico degli LLM con una rappresentazione visiva del layout del documento. LayoutLLM viene addestrato a comprendere istruzioni testuali che descrivono la disposizione e le relazioni spaziali degli elementi nel documento, consentendo una più accurata comprensione semantica e strutturazione visiva. Rispetto ai modelli precedenti, LayoutLLM ha come obiettivo quello di superare la distinzione tra testo e layout, fornendo una rappresentazione più coerente e orientata al ragionamento.

2.3.2 BLOCKIE

BLOCKIE [3] è un approccio innovativo proposto da Amazon nel 2025 per l'estrazione di informazioni da documenti complessi(*Visually Rich Documents*). Si distingue per lintroduzione del concetto di *semantic blocks*, ossia gruppi di testo auto-consistenti che vengono identificati e analizzati in modo indipendente tramite LLM. Ogni blocco viene considerato come un'unità semantica indipendente e viene elaborata autonomamente. Successivamente i blocchi vengono ricombinati in una struttura coerente del documento. Questo metodo consente una comprensione più approfondita di blocchi presi singolarmente, consentendogli di ottenere risultati migliori rispetto ad esempio a LayoutLLM per benchmark standard(CORD, FUNSD, SROIE).

Capitolo 3

Obiettivo e casi d'uso

Alla luce del contesto delineato all'interno del capitolo introduttivo, si definiscono ora gli obiettivi specifici e i casi d'uso in particolare.

3.1 Scopo del progetto

Le finalità riguardano la prototipazione e la valutazione di un sistema in grado di convertire PDF nativi in HTML strutturato e semantico, sfruttando i Large Language Models. Ciò che si cerca di ottenere in output è un file HTML che riproponga una ricostruzione coerente del layout, che non debba essere necessariamente allineata al PDF originale. La gerarchia e la struttura semantica di HTML sono più importanti della riproduzione posizionale perfettamente identica. Il sistema consente l'estrazione e l'immissione delle immagini con formato, risoluzione e *bounding box*, ovvero il rettangolo che racchiude completamente l'immagine, preservati. Si precisa che la presente trattazione si concentra esclusivamente sugli aspetti tecnici di estrazione e conversione. Questioni legate al copyright o alle implicazioni normativa relative al riuso e alla manipolazione dei PDF non vengono qui analizzate

L'applicazione web sviluppata si chiama PDeFy, nome che gioca sulla sigla PDF unita al verbo *defy* che significa "sfidare": il progetto mira a superare i limiti e la rigidità del formato del PDF, volendolo convertire in qualcosa di strutturato. La volontà di conversione del PDF nasce dal fatto che un file HTML, se ben strutturato, permette una maggiore accessibilità, possibilità di modifica e adattabilità a contesti diversi. Il lavoro proposto si classifica come un prototipo sperimentale per la valutazione di una piattaforma che combina strumenti tradizionali ai più moderni LLM accessibili a chiunque per la conversione semantica di documenti scolastici.

3.2 Casi d'uso e tipologie di esercizi

Il sistema è specializzato e testato per documenti contenenti esercizi per la scuola primaria, pensato in primo luogo per insegnanti che hanno bisogno di una ricostruzione di proprie schede, parti di libro di testo o test di verifica. Oltre ai docenti, il progetto può essere di interesse per educatori, case editoriali di libri scolastici, ricercatori nel campo della didattica digitale e sviluppatori nel campo dell'analisi di documenti. L'obiettivo è che la piattaforma possa essere utilizzata in piattaforme di e-learning oppure a monte della produzione di eserciziari o simili.

I test sono avvenuti con le tipologie di layout e di quesiti più disparate, cercando di coprire quanto possibile tutto il raggio possibile di tipologie di materiale per la scuola elementare. Sebbene il sistema sia incentrato su questi tipi di PDF, ciò non toglie che possa essere testato e utilizzato anche con altre tipologie di documenti. Il progetto non prende in considerazione i PDF scannerizzati come immagini, ma solo i PDF avente una struttura interna nativa classica.

Gli esercizi sono classificati attraverso specifici nomi di classi definite dal prompt e assegnate dal modello. L'LLM considera il contesto, la consegna e il contenuto degli esercizi e etichetta gli esercizi tra le seguenti categorie:

- **completamento**: esercizi all'interno del quale vi è un lavoro di riempimento di spazi bianchi all'interno di frasi o di parole.
- **scelta-multipla**: esercizi che prevedono la scelta corretta tra più risposte proposte.
- **collegamento**: individuabile quando vi è da associare un elemento ad un altro elemento.
- **vero-falso**: quando si chiede una risposta secca tra vero o falso.
- **ordinamento**: ovvero una tipologia di esercizio che prevedere il mettere nel giusto ordine una lista di parole o elementi.
- **domanda-aperta**: trattasi di una domanda che non prevede proposte di risposte, bensì frutto di un lavoro di scrittura.
- **individuazione**: esercizi in cui occorre sottolineare, evidenziare, cerchiare qualcosa.
- **scrittura**: simile alla domanda aperta, ma che non preveda necessariamente una domanda, ma anche una consegna con più libertà di scrittura.
- **calcolo**: esercizi che hanno a che fare con la matematica, ove occorre fare dei calcoli di qualsiasi natura.

- **disegno**: esercizi in cui occorre utilizzare la comunicazione artistica, tramite disegno o colori.

Talvolta potrebbero essere associate più classi a determinati esercizi, ciò accade quando si stanno analizzando esercizi che possono effettivamente rientrare in più categorie o risultano in bilico tra alcune di natura simile. Se nessuna di queste categorie è individuata, viene assegnata la classe predefinita `class="exercise exercise-generico"`.

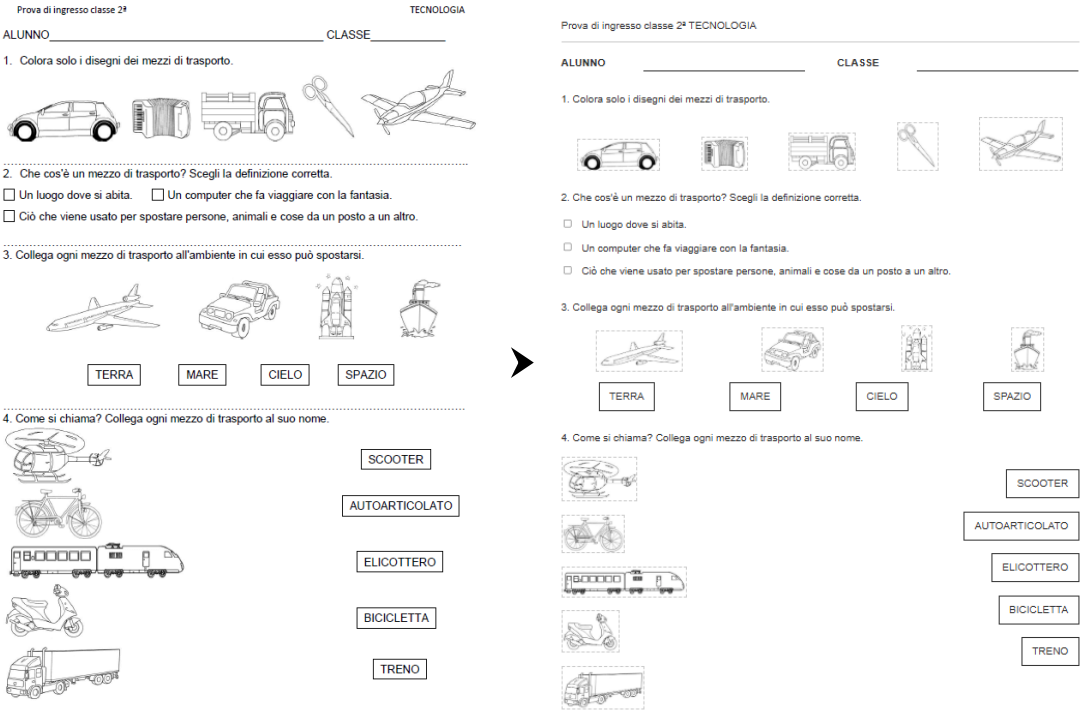

Figura 3.1: Esempio di conversione mediante PDeFy

Nello snippet sottostante viene riportato un esempio di un'identificazione di un esercizio di collegamento. Non sono riportate tutte le opzioni di collegamento ma vi è una chiara distinzione tra i `connection-item`, ovvero gli oggetti da collegare alle loro definizioni, le `connection-description`. Tramite l'LLM è stata riconosciuta sia la tipologia di esercizio che gli elementi che lo compongono e il loro ruolo. Nella figura sottostante riportiamo l'esercizio intero convertito.

```
<section id="exercise-1" class="exercise exercise-collegamento">
<div class="connection-container">
<div class="left-column">
<div class="connection-item">
<div class="image-placeholder" style="width: 54.33px; height: 139.77px;"><img
    style="width: 54.335999599999994px; height: 139.775999px;" src="blob:http
    ://localhost:5173/da32ea96-6317-4c79-8f9d-5a673eda2514"></div>
<p>RE</p>
...

<div class="right-column">
<div class="connection-description">Coltiva la terra.</div>
...
```

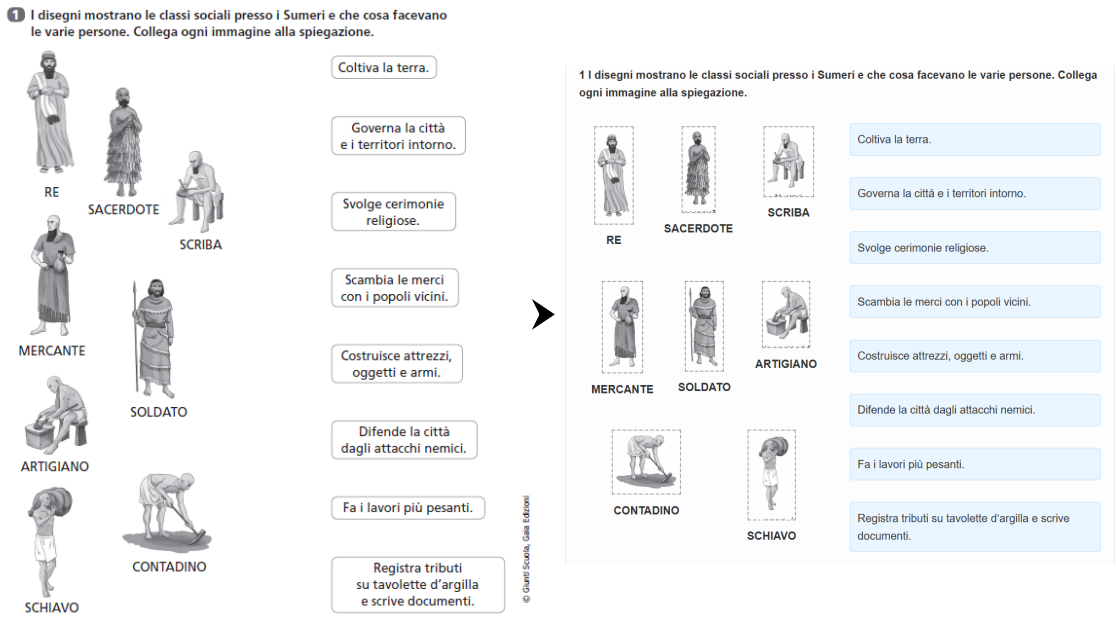

Figura 3.2: Esempio di conversione mediante PDeFy

Capitolo 4

Architettura del progetto

Il progetto prevede due modalità di utilizzo: una mediante *CLI (Command Line Interface)*, e una seconda mediante un'applicazione web appositamente progettata e implementata. Nel presente capitolo verranno dapprima discusse le due pipeline identificate da due endpoint separati, poi esplicitate le scelte tecnologiche, la struttura del progetto, le interfacce di utilizzo. Saranno infine menzionati gli LLM scelti e descritto il processo di costruzione del prompt.

4.1 UI e funzionalità dell'applicazione web

L'applicativo web è pensato e progettato per essere *user-friendly*, mirando alla garanzia di facilità di utilizzo anche per utenti non esperti. L'interfaccia privilegia la chiarezza puntando a rendere immediatamente comprensibili le funzionalità. Gli obiettivi della piattaforma emergono anche dagli elementi che compongono il logo visibile in figura: L'evidenziazione della sigla PDF mediante un colore diverso, la presenza della piega in cima alla P, simbolo del formato PDF, le parentesi di tag tipici di HTML e la volontà di porre l'attenzione sulla parola "Defy"

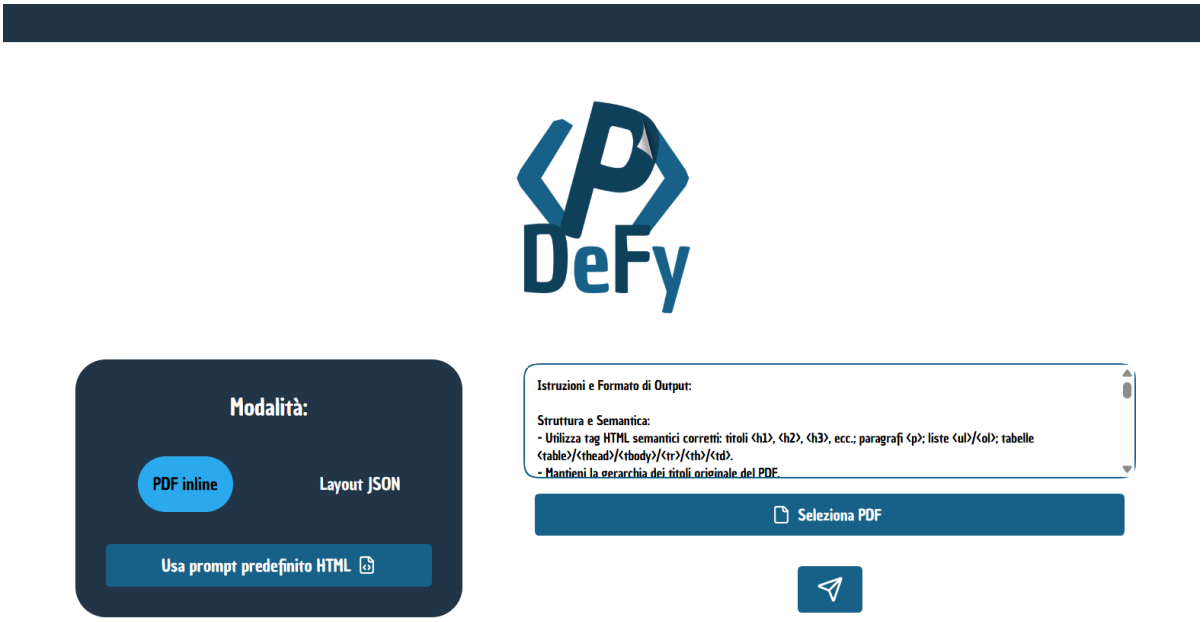

Figura 4.1: Screenshot dell'applicazione web

All'interno della sezione a sinistra della schermata sono presenti due *radio button* che permettono la selezione della modalità di invio dei dati all'LLM. Dopo aver selezionato quella che si intende utilizzare, è possibile utilizzare un bottone che inserisce all'interno della *textarea* il prompt ottimizzato relativo alla modalità scelta.

Il prompt è quindi modificabile dall'utente a proprio piacimento, tuttavia, ai fini dell'ottenimento di risultati soddisfacenti, è consigliabile utilizzare quello predefinito. L'interfaccia permette poi la selezione di uno o più PDF che verranno processati a partire dal click sul comando di invio sottostante. Quando un risultato è pronto, viene automaticamente effettuato un download del file `.html` e viene mostrato un *alert* qualora il file in questione fosse l'ultimo dell'eventuale coda. Nonostante il file venga scaricato direttamente, all'interno della pagina comparirà l'editor WYSIWYG tramite il quale effettuare le eventuali e opportune modifiche, procedendo successivamente ad un nuovo download.

Tramite il bottone azzurro che raffigura i due tag <>, è possibile aprire una schermata che mostra il codice HTML modificabile.

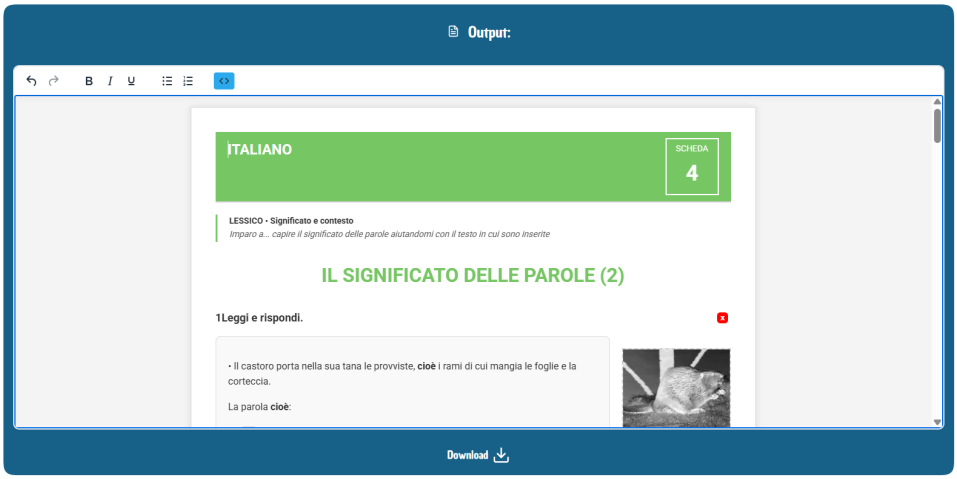

Figura 4.2: Screenshot dell'editor TinyMCE

Figura 4.3: Screenshot della schermata di codice sorgente

4.2 Utilizzo mediante CLI

Oltre all'esecuzione tramite interfaccia web, il sistema può essere utilizzato anche da da riga di comando (`cli.js`). Questo tipo di esecuzione utilizza uno dei due end-point già esplicitati che ricordiamo effettuare le seguenti operazioni:

- ricezione di uno o più PDF passati come argomento;
- invocazione delle API Gemini con il prompt di generazione HTML;
- estrazione delle immagini tramite lo script Python `extract_images.py`;

- sostituzione dei placeholder `[IMAGE_X]` con le immagini codificate in Base64;
- esportazione del risultato in un file `HTML` autonomo.

È inoltre supportata l'elaborazione *batch*, con un limite configurabile di processi concorrenti, rendendo lo strumento adatto a scenari di automazione o integrazione in pipeline di produzione.

Per utilizzarlo occorre posizionarsi nella cartella *cli_app*, dove al cui interno è presente lo script *cli.js* e utilizzare il comando pdefy avente la seguente struttura:

```
pdefy [--mode <modalità>] [--out <cartella>] <pdf1>
```

Il comando *pdefy* costituisce l'interfaccia a riga di comando dell'applicazione che esegue sostanzialmente tramite node `cli.js`. Dopo il nome del comando si immettono uno o più file che si intende analizzare. Il comando completo prevede la presenza di due configurazioni che si possono opzionalmente specificare:

- –**mode**, che specifica la modalità di invio dei dati al modello (inline o json). Se non si specifica di default verrà utilizzata la modalità inline.
- –**out**, a seguito del quale si definisce la directory in cui si vuole salvare l'output. Se mancante verrà salvato all'interno della cartella stessa `cli_app`.

È prevista la possibilità di esecuzione di più documenti direttamente mediante lo stesso comando

```
pdefy [--mode <modalità>] [--out <cartella>] <pdf1> <pdf2> ...
```

4.3 Le due pipeline: inline e JSON

Già menzionate precendentemente, questa suddivisione logica costituirà un punto fondamentale per il lavoro svolto. Nei successivi capitoli sarà presente questa suddivisione del flusso di lavoro che prevede l'invio del PDF al modello di Gemini in maniere differenti e utilizzando strumenti differenti. Le due pipeline costituiscono due endpoint separati risiedenti entrambi nel file `server.js` nel progetto: `/api/generate` per il flusso *inline* e `/api/generate_JSON` per il flusso *json*.

Nel flusso *inline* il PDF viene passato direttamente all'LLM, assieme ai metadati delle immagini estratte da uno script Python (`extract_images.py`) e al prompt. Il layout viene ricostruito per integrazione tra il backend e la risposta fornita dal modello.

Pipeline INLINE

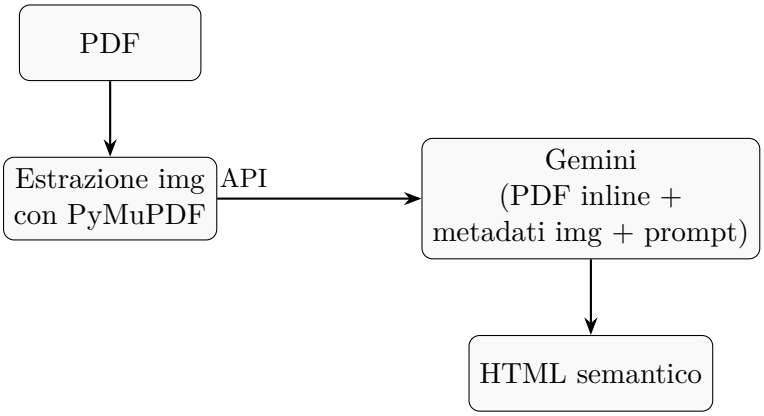

Figura 4.4: Diagramma riassuntivo della pipeline INLINE

Nel flusso *json* il file viene dapprima estratto in un formato json tramite uno script python(`extract_layout_JSON`), che estrae testo, elementi e immagini. Il JSON completo verrà inviato all'LLM assieme al prompt e le immagini verranno immesse in modo deterministico come avviene nell'altra pipeline.

Questa suddivisione è stata introdotta per verificare quale modalità di invio producesse risultati migliori. Lo scopo è comprendere in che modo il modello generativo ricostruisce e interpreta meglio il layout originale. Il dettaglio dei due *workflow* sarà più completo all'interno del capitolo 5.

Pipeline JSON

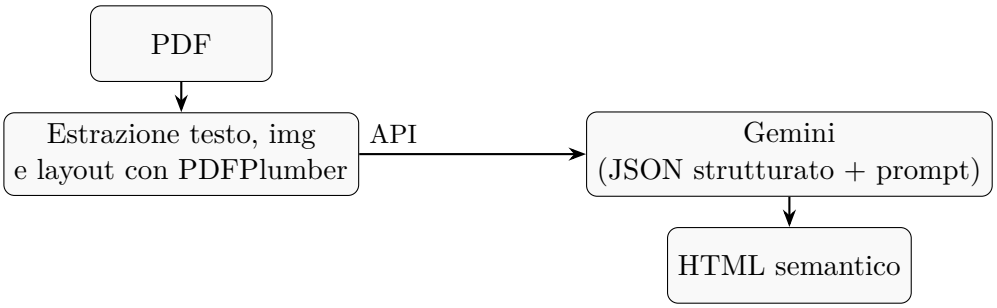

Figura 4.5: Diagramma riassuntivo della pipeline JSON

4.4 Scelte tecnologiche e struttura del progetto

L'applicazione web è stata progettata secondo un'architettura di tipo *client-server*, suddivisa in due macro-componenti principali: il **frontend** e il **backend**. Il *frontend* è responsabile dell'interfaccia utente, della visualizzazione dei risultati e della manipolazione di questi ultimi, il *backend* gestisce le pipeline di estrazione, conversione e generazione dell'output.

4.4.1 Lato server: Node.js, Express e Python

L'ambiente di esecuzione del *backend* è **Node.js**, che permette di utilizzare JavaScript lato server per gestire la logica applicativa, l'invio e la ricezione di richieste HTTP, e l'interazione con gli script Python dedicati all'estrazione dei contenuti PDF. Il *backend* dell'applicazione è strutturato in maniera modulare ed è strutturato nel seguente modo:

```
backend/
 cli_app/
    cli.js
 node_modules/
 uploads/
    tmp_layout_inline/
       images/
       pdf/
    tmp_layout_JSON/
        images/
        layouts/
 .env
 extract_images.py
 extract_layout_JSON.py
 html_prompt.js
 html_prompt_2.js
 package.json
 package-lock.json
 server.js
```

- `server.js`, rappresenta il punto di accesso e lo snodo principale del *backend*. Al suo interno è definito il server Express gli endpoint REST `/api/generate` e `/api/generate_JSON`. La separazione dei due endpoint definisce le due pipeline in maniera netta: nel primo caso all'LLM viene passato in input il PDF inline, i metadati delle immagini ricavati da *PyMuPDF* e il prompt; nel secondo caso l'LLM

riceverà in input un JSON strutturato creato mediante la libreria *pdfPlumber*, e il prompt.

- `extract_images.py`, script Python utilizzato dall'endpoint `/api/generate`.
- `extract_layout_JSON.py`, script Python utilizzato dall'endpoint `/api/generate_JSON`.
- `/uploads`, cartella al cui interno vengono salvati temporaneamente le immagini e i PDF estratti.
- `html_prompt` e `html_prompt_2`, contenente i *prompt* rispettivamente per l'endpoint `/api/generate` e `/api/generate_JSON`.
- `/cli_app`, contenente il file `cli.js`, all'interno del quale è gestito l'utilizzo mediante *Command Line interface*.

4.4.2 Lato client: React e TinyMCE

Il lato client dell'applicazione è sviluppato in **React**, framework JavaScript che consente di creare interfacce dinamiche e modulari, basate su componenti riutilizzabili.

```
frontend/
 node_modules/
 public/
 src/
    assets/
       App.css
    App.jsx
    index.css
    main.jsx
    WysiwygEditor.css
    WysiwygEditor.jsx
 .env
 .gitignore
 eslint.config.js
 index.html
 package.json
 package-lock.json
```

- `App.jsx`, rappresenta l'interfaccia principale che permette il caricamento di uno o più file PDF, la selezione della pipeline desiderata (PDF inline o JSON) e la

visualizzazione dell'output generato dal modello linguistico. Il componente gestisce la comunicazione con il *backend* attraverso chiamate `fetch`

- `WysiwygEditor.jsx`, è il componente React che si occupa della gestione dell'editor all'interno del quale è possibile manipolare l'output. La scelta dell'editor è ricaduta su **TinyMCE** [16] tramite *API*, scelta motivata dall'elevata personalizzazione offerta. Si tratta di un editor *WYSIWYG* (*What You See Is What You Get*), ovvero un ambiente che permette di visualizzare e modificare un documento in una forma il più possibile simile al risultato finale. In altre parole, ogni intervento dell'utente sull'interfaccia produce un effetto visivo immediatamente verificabile dal contenuto visualizzato. L'editor consentirà la modifica totale dell'output, la rimozione di esercizi singoli e il *drag & drop* per il riordinamento delle immagini, ove necessario. Inoltre l'editor offre una visualizzazione alternativa contenente direttamente il codice HTML, le cui modifiche si rifletteranno immediatamente sul documento. Questa duplice modalità di visualizzazione combina semplicità a controllo tecnico, migliorando l'accessibilità sia per utenti non esperti che per sviluppatori.

4.5 Implementazione e scelta Gemini tramite API

Per la realizzazione del progetto, la scelta è ricaduta sul chatbot di intelligenza artificiale generativa di Google, **Google Gemini**. Perchè, a differenza dei propri competitor (ChatGPT, Claude ecc.), Gemini offre un *free tier* consistente e con limiti di token e di chiamate alto e sufficiente per il test di prototipi. Nel mio caso specifico ho usufruito di limiti più alti e di un accesso leggermente più privilegiato rispetto al piano gratuito poiché ho inserito la fatturazione. Nonostante ciò tutto l'iter di valutazione e test della piattaforma è avvenuto in modo gratuito mediante l'utilizzo del credito gratuito messo a disposizione da Google. Per continuare ad utilizzare in maniera illimitata l'applicazione web, ci sarebbero ipoteticamente dei costi aggiuntivi valutati in base all'utilizzo della stessa. Inoltre ho avuto la possibilità di testare l'applicazione anche utilizzando il modello in pre-rilascio del nuovo modello di Gemini, uscito a Novembre del 2025. In particolare le sperimentazioni sono state fatte mediante l'utilizzo di tre modelli [6]:

- **Gemini 2.5 Pro**
- **Gemini 2.5 Flash**
- **Gemini 3 Pro Preview**

All'interno dell'architettura dell'applicazione, il modello di intelligenza artificiale Gemini viene integrato tramite chiamate API REST fornite da Google. L'interazione con Gemini avviene attraverso l'endpoint ufficiale `generateContent`, il quale permette l'invio combinato di testo, metadati e contenuti binari.

4.6 Costruzione del prompt

La costruzione del prompt da inviare a Gemini costituisce un passaggio cruciale: esso stabilisce le regole e i vincoli che guidano il comportamento del modello riflettendosi sulla qualità del risultato. Prima del prompt principale, invio all'LLM un *system prompt*, che definisce il contesto all'interno del quale deve aspettarsi di lavorare il modello. Il "prompt preliminare" che ho utilizzato è visibile nel corpo della richiesta al Capitolo 5 e fornisce i primi strumenti per far capire all'LLM il ruolo in cui deve immedesimarsi. Utilizzando altri LLM la tipologia di prompt in questione è caratterizzato da `role:system`, ciò non avviene in Gemini. In Gemini un prompt di questo tipo viene semplicemente definito come parte dei `contents` da inviare e all'interno della chiamata alla API: il dettaglio importante è che si invii come primo contenuto. In questo modo, pur mantenendo un `role:user`, il modello comprenderà che quello corrisponde al *system prompt*, proprio dall'ordine gerarchico utilizzato.

Prompt preliminare

```
Restituisci esclusivamente HTML puro.
Non formattare l'output come blocco di codice.
Non inserire i delimitatori \\\html o \\\.
Inizia direttamente dal primo tag HTML.
e termina con l'ultimo.
Se stai per inserire un blocco \\\html,
rimuovilo e restituisci solo il contenuto.
Non includere spiegazioni, testo extra
o introduzioni/conclusioni.
```

In questo prompt si assegna al modello un ruolo specifico, definendo il perimetro del suo compito. È risultato importante evitare che il modello aggiunga elementi indesiderati, come testi ausiliari, o delimitatori di codice che compromettono la pulizia del risultato finale.

Il prompt completo costituisce il vero punto di contatto tra la fase di estrazione e il modello di Gemini, determinando in modo diretto la qualità e la stabilità dell'output generato. La progettazione delle istruzioni in input ha costituito delle vere e proprie fasi di *prompt engineering* ossia di studio delle tecniche che permettono la comunicazione efficace con un LLM. Seguendo quanto emerge dalla recente letteratura sul *prompt engineering*, in particolare nella review di Chen et al. [4], la creazione di un prompt ben strutturato è un processo sistematico e richiede uno studio delle tecniche attraverso

il quale si riducono le *hallucinations* e si ottengono risultati più vicini a quelli sperati. Dopo varie fasi di sviluppo iterativo intervallati da test ripetuti, sperimentazione e raffinamento si è giunti alla creazione di un prompt coerente, chiaro e privo di ridondanze. I prompt relativi alle due pipeline differiscono per qualche dettaglio operativo, ma condividono una struttura comune che sarà analizzata nelle prossime sezioni. Gli *snippet* di prompt che verranno mostrati corrispondono a porzioni salienti a cui si vuole dar rilievo.

- **Regole di layout**
- **Posizionamento placeholders immagini**
- **Riconoscimento e classificazione degli esercizi**

Attraverso l'applicazione web, il prompt è modificabile nel caso si voglia fornire istruzioni differenti o aggiungerne di nuove. In alcune porzioni il prompt è molto sensibile e una modifica ad esse potrebbe portare a malfunzionamenti. Ne è un esempio la sezione che definisce la modalità di immissione dei placeholder, cambiarla potrebbe portare il *backend* a non essere più compatibile con ciò che riceve dal modello in fase di rimpiazzo con le immagini fisiche. Il prompt finale completo, utilizzato per tutte le sperimentazioni, è riportato in formato integrale in **Appendice A** (pag. 48). Il prompt per la pipeline JSON è simile, ma presenta una differenza importante di definizione dell'input, presente in **Appendice B** (pag. 51).

4.6.1 Regole di layout

Composta da tre sottosezioni: struttura e semantica, definizione dello stile e individuazione esercizi. La prima si tratta di una sezione introduttiva in cui viene specificato l'utilizzo dei tag corretti HTML per evitare ambiguità, mantenere la corretta gerarchia di testo del PDF e l'ordine generale del contenuto.

Snippet sezione di struttura e semantica

```
- Utilizza tag HTML semantici corretti: titoli <h1>, <h2>, <
   h3>, ecc.; paragrafi <p>; liste <ul>/<ol>; tabelle <table
   >/<thead>/<tbody>/<tr>/<th>/<td>.
```

Nella seconda viene esplicitato il fatto di creare un foglio di stile integrato in `<head>` in modo che all'interno del file possa esserci tutto il contenuto necessario.

Snippet di definizione dello stile

```
- Includi una sezione <style></style> all'interno di <head>
   per definire uno stile gradevole, pulito e coerente.
```

Nella terza sezione si mira a realizzare una separazione strutturale degli esercizi mediante delle section. Viene inoltre specificato di non risolvere gli esercizi in modo tassativo poiché è un qualcosa che l'LLM è portato a fare. Senza istruzioni specifiche l'invio di un esercizio che riesce a risolvere lo porta a "pensare" che debba farlo.

Snippet sugli esercizi

```
- NON risolvere NESSUN esercizio. Riportali fedelmente.
- Ogni esercizio deve essere contenuto in una <section> con
   id esatto: id="exercise-n"
```

4.6.2 Posizionamento placeholders delle immagini

Per gestire le immagini è stato implementato un meccanismo a due fasi. Una prima fase di immissione di *placeholder* da parte di Gemini, che serve per marcare il corretto posizionamento dell'immagine all'interno del layout. si utilizza il modello per inserire dei `<div>` contenenti i suddetti placeholder delle dimensioni corrette. La seconda fase è l'immissione delle immagini estratte dagli script python da parte del backend. Quest'ultima fase verrà esplicata in modo dettagliato nella sezione 5.5.

Snippet di definizione dei placeholder

```
- NON includere immagini reali: sostituisci ogni immagine
   con un placeholder centrato del formato esatto: [IMAGE_X
   ], dove X è l'indice progressivo dell'immagine nel PDF (
   la prima immagine è [IMAGE_1], ecc.).
- Inserisci il placeholder [IMAGE_X] dentro un contenitore <
   div class="image-placeholder"> centrato.
```

Tra i metadati relativi alle immagini sono presenti: pagina, coordinate x/y e larghezza/altezza. Il modello userà quei dati per identificare la posizione e l'area occupata, interpretando il layout nel modo più coerente possibile.

Snippet per utilizzo metadati delle immagini

```
- Usa le coordinate per posizionarle nel punto
   corrispondente al layout del pdf.
- Usa width e height per considerarle nel layout di quella
   dimensione
```

4.6.3 Riconoscimento e classificazione degli esercizi

Affido all'LLM i compito di classificazione della tipologia di esercizio mediante la visione del contesto e soprattutto le parole chiave all'interno della consegna dell'esercizio.

Snippet per classificazione esercizi

```
- Identifica automaticamente la tipologia dell'esercizio
   basandoti su parole chiave presenti nel titolo o nell'
   enunciato.
- Aggiungi alla sezione dell'esercizio un attributo class
   che includa la categoria riconosciuta.
  Esempio: <section id="exercise-1" class="exercise exercise
   -completamento">.
```

La scelta del modello è stata accuratamente guidata attraverso una serie di categorie disponibili, per cui ho indicato delle parole chiave specifiche. Ne riporto un esempio nello snippet sottostante (L'elenco completo delle categorie si trova alla sezione 3.3 "Casi d'uso e tipologie di esercizi")

Snippet esempio classificazione

```
- "scelta-multipla" - se compaiono termini come "scegli", "
   seleziona", "indica la risposta corretta", "cerchia".
```

L'LLM può associare più classi ad alcuni esercizi, ciò è voluto dato che esistono degli esercizi che possono appartenere a categorie intermedie, quindi a più tipologie.

Un esempio di conversione di un esercizio del file "Aggettivi" dal dataset di analisi utilizzato nel capitolo 6.

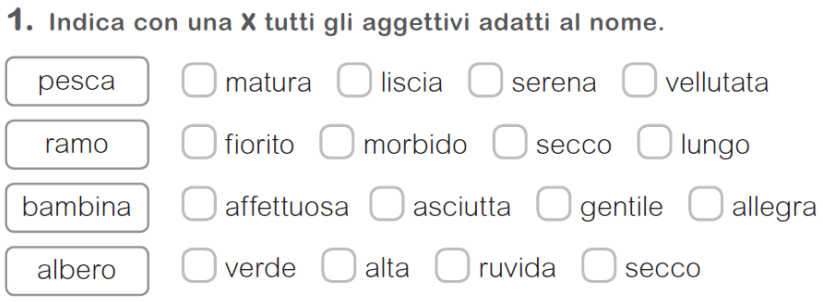

Figura 4.6: Screenshot di un esercizio

```
<section id="exercise-1" class="exercise exercise-scelta-multipla" style="
    position: relative;">
<div class="exercise-title"><span class="exercise-number">1.</span> Indica con
     una X tutti gli aggettivi adatti al nome.</div>
<div class="ex1-row">
<div class="noun-box">pesca</div>
<div class="adj-options"><label class="adj-option"> matura</label> <label
    class="adj-option"> liscia</label> <label class="adj-option"> serena</
    label> <label class="adj-option"> vellutata</label></div>
</div>
<div class="ex1-row">
<div class="noun-box">ramo</div>
<div class="adj-options"><label class="adj-option"> fiorito</label> <label
    class="adj-option"> morbido</label> <label class="adj-option"> secco</
    label> <label class="adj-option"> lungo</label></div>
</div>
<div class="ex1-row">
<div class="noun-box">bambina</div>
<div class="adj-options"><label class="adj-option"> affettuosa</label> <label
    class="adj-option"> asciutta</label> <label class="adj-option"> gentile</
    label> <label class="adj-option"> allegra</label></div>
</div>
<div class="ex1-row">
<div class="noun-box">albero</div>
<div class="adj-options"><label class="adj-option"> verde</label> <label class
    ="adj-option"> alta</label> <label class="adj-option"> ruvida</label> <
    label class="adj-option"> secco</label></div>
</div>
```

Capitolo 5

Implementazione del sistema

In questo capitolo viene descritta nel dettaglio l'implementazione di PDeFy, illustrando le componenti operative che entrano in gioco e le modalità attraverso le quali comunicano tra loro. Dopo una fase preliminare di *pre-processing*, viene presentato il *workflow* generale della conversione e, a seguire, l'implementazione delle diverse fasi che compongono le due pipeline (inline e JSON). Si analizzeranno i meccanismi di gestione dei file, l'estrazione dei contenuti, la costruzione delle richieste verso il modello Gemini e la sostituzione programmatica delle immagini. Il capitolo si conclude con la descrizione dell'editor integrato, che permette all'utente la revisione manuale dell'output generato.

5.1 Pre-processing del PDF

Un'operazione importante da svolgere per riuscire ad usufruire di PDeFy è la **Rimozione di immagini di sfondo** ove presenti. Il flusso di estrazione non prevede l'estrazione di immagini al di sotto di testo: mantenere le immagini solo se non prevedono testo al di sopra di esse. Ulteriori accorgimenti per il miglioramento dei risultati:

- **Rimozione di immagini non utili agli esercizi**, ovvero di immagini di contorno, che non apportano informazioni utili e possono influenzare la percezione del layout da parte del modello.
- **Snellimento di elementi grafici**, piccole icone, ornamenti o figure astratte possono essere estratte come immagini vere e proprie creando rumore all'interno del risultato finale, soprattutto se interferiscono con il contenuto.
- **Riduzione di timbri o immagini sovrapposte**, in quanto il processo di conversione presuppone che ogni immagine sia collocata all'interno di div strutturati e non interferisca con il testo. Elementi sovrapposti possono essere estratti in modo ambiguo e, successivamente, reinseriti in posizioni errate all'interno dell'HTML generato dal modello.

Determinate operazioni non sono necessarie e costituiscono solo un mezzo attraverso il quale ottenere risultati migliori: un layout più pulito in input garantirà risultati più soddisfacenti in output. La diminuzione del rumore in ingresso ove possibile fa si che la comprensione del layout da parte del modello sia più corretta e lineare.

Il processo di estrazione e conversione da PDF ad HTML è reso possibile da una concatenazione di diverse operazioni che trasformano il documento in un output semantico e leggibile. Il *workflow* è stato progettato per essere adattabile, modulare e scalabile. Testare il sistema con altri modelli di intelligenza artificiale risulta semplice poiché è sufficiente modificare le costanti dichiarate nel server ed andare a modificare, eventualmente, la chiamata API nella configurazione prevista dal modello scelto.

```
1 const LLM_API_BASE = "https://generativelanguage.googleapis.com/v1beta/models"
    ;
2
3 const LLM_MODEL_PRO = "gemini-2.5-pro:generateContent";
4 const LLM_MODEL_FLASH = "gemini-2.5-flash:generateContent";
5 const LLM_MODEL_3PRO = "gemini-3-pro-preview:generateContent";
6
7 const LLM_SELECTED_MODEL = LLM_MODEL_PRO;
8 const LLM_MODEL_URL = `${LLM_API_BASE}/${LLM_SELECTED_MODEL}`;
```

Nell'esposizione delle varie fasi del lavoro, il dettaglio verrà suddiviso ove necessario tra le due pipeline, andando successivamente a confluire nella gestione dell'output.

5.2 Upload e gestione dei file PDF

La gestione del caricamento dei file PDF avviene all'interno di `App.jsx`. I file PDF caricati verranno salvati in primis in uno `State Hook (files, setFiles)`, che converte la lista di file ottenuta dallevento `onChange` in un array JavaScript standard. In questo modo il sistema è in grado di mantenere e visualizzare in modo dinamico i file selezionati prima dell'elaborazione. I file selezionati non verranno processati in parallelo, bensì in modo sequenziale. il metodo `processQueue` implementa una *queue frontend*, iterando su tutti i file. Ad ogni iterazione il file corrente viene caricato in un `FormData` e inviato al server tramite una chiamata fetch. La chiamata, in particolare, riguarderà uno dei due endpoint lato server, a seconda della scelta dell'utente per la modalità di conversione.

La scelta di gestione della coda lato *frontend* è stata effettuata per 2 motivazioni principali:

- **riduzione concorrenza** lato server, in modo che riceva direttamente una coda ordinata.
- **gestione errori** più accurata per singolo file, con gestione degli errori più chiara.

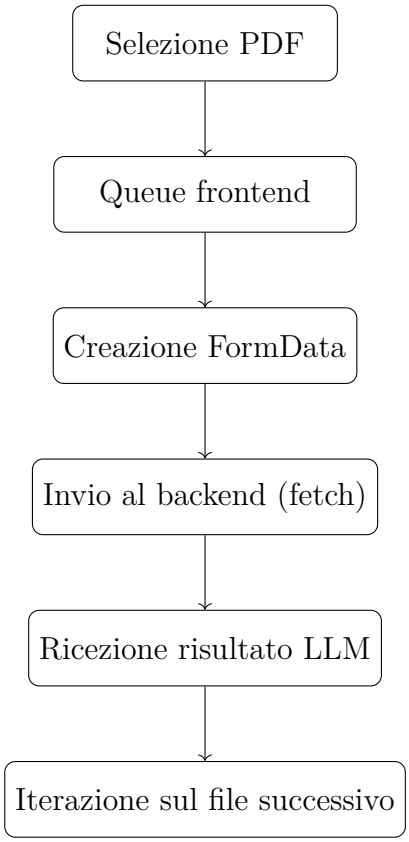

Figura 5.1: Workflow di elaborazione sequenziale dei PDF lato frontend.

5.3 Estrazione di testo e immagini

5.3.1 Pipeline `/api/generate`

Il PDF viene inviato *inline* al modello di intelligenza artificiale, per cui non vi è vera e propria estrazione del testo o dei contenuti a partire dal PDF. L'elemento su cui si concentra l'estrazione manuale da PDF sono le immagini. Vengono estratte mediante lo script `extract_images.py` che utilizza la libreria PyMuPDF. Si riporta uno snippet di `extract_images.py`

```
for page_index, page in enumerate(pdf, start=1):
image_list = page.get_images(full=True)
    for img_index, img in enumerate(image_list, start=1):
        xref = img[0]
        base_image = pdf.extract_image(xref)
        image_bytes = base_image["image"]
        ext = base_image["ext"].lower()

        img = Image.open(io.BytesIO(image_bytes))
            path = f"{output_dir}/page{page_index}_img{img_index}.png"
        img.save(path, "PNG")

        rects = []
        for inst in page.get_image_info(xrefs=True):
            if inst["xref"] == xref:
                rects.append(inst["bbox"])
```

Lo snippet mostra il funzionamento centrale dello script di estrazione delle immagini. Per ogni pagina del PDF, la funzione `get_images(full=True)` individua tutte le immagini incorporate nel documento, restituendo una lista di riferimenti interni (*xref*). Ogni *xref* viene utilizzato per estrarre il contenuto binario dell'immagine tramite `extract_image`, successivamente decodificato in memoria e salvato in formato PNG nella directory temporanea appositamente creata. Il metodo `get_image_info(xrefs=True)` consente di recuperare tutte le istanze grafiche delle immagini presenti nella pagina, incluse le relative *bounding box*. Poiché un'immagine può comparire più volte nella stessa pagina, il codice confronta l'*xref* corrente con le istanze rilevate, associando ad ogni occorrenza la sua posizione geometrica.

Per ogni `rect` vengono memorizzate le informazioni finali, che verranno consolidate in una struttura JSON e integrate nella richiesta da inviare al modello. Se ne riporta un esempio:

```
{
  "page": 1,
  "path": "uploads/tmp_layout_inline/images/page1_img1.png",
  "ext": "png",
  "bbox": [72.0, 155.2, 352.0, 295.2],
  "x": 72.0,
  "y": 155.2,
  "width": 280.0,
  "height": 140.0
}
```

- **page**: il numero di pagina da cui l'immagine è stata estratta.
- **path**: il percorso del file immagine generato durante l'elaborazione. Tale file rappresenta la versione rasterizzata dell'immagine così come appare nel PDF, utile per l'inserimento successivo all'interno dell'HTML.
- **ext**: l'estensione del formato immagine utilizzato in output. Questo campo specifica il tipo di codifica adottato per la rappresentazione raster.
- **bbox**: il riquadro di delimitazione (*bounding box*) dell'immagine all'interno della pagina PDF, espresso nel formato `[x0, y0, x1, y1]`. Il valore `x0` rappresenta la coordinata orizzontale del margine sinistro, `y0` quella del margine superiore, `x1` e `y1` indicano rispettivamente i margini destro e inferiore. Questo riquadro consente di determinare la posizione esatta e lo spazio occupato nel layout.
- **x** e **y**: le coordinate dell'angolo superiore sinistro della *bounding box*. Sono equivalenti a `x0` e `y0`, ma vengono fornite separatamente per semplificare l'accesso diretto alle coordinate principali.
- **width** e **height**: la larghezza e l'altezza dell'immagine, calcolate rispettivamente come `x1 - x0` e `y1 - y0`. Questi valori descrivono l'estensione delle immagini, valori poi utilizzati nella ricostruzione dell'output.

È opportuno osservare che la libreria PyMuPDF estrae le immagini nella loro forma interna così come sono codificate nel PDF, ciò porta a vedere l'immagine estratta talvolta

con colori alterati o invertiti. Questo avviene poiché alcuni documenti PDF potrebbero utilizzare in alcune immagini al suo interno spazi di colore non standard (ad esempio CMYK) , maschere di trasparenza o filtri di compressione complessi. Il comportamento in questione non dipende quindi da PDeFy bensì accade poiché si riflette fedelmente la rappresentazione strutturale dell'immagine all'interno del file PDF.

5.3.2 Pipeline `/api/generate_JSON`

Il PDF viene inviato sotto forma di JSON strutturato all'LLM, per cui viene effettuata un'estrazione completa del testo, delle tabelle e delle immagini contenute nel file. Si utilizza la libreria *PDFPlumber* all'interno dello script `extract_layout_JSON` di cui se ne riporta uno *snippet*:

```
with pdfplumber.open(pdf_path) as pdf:
    for page_index, page in enumerate(pdf.pages, start=1):
        full_text = page.extract_text() or ""
        words = page.extract_words() or []
        tables = page.extract_tables() or []
        images_info = []
        for img_index, img in enumerate(page.images, start=1):
            # Bounding box
            x0, y0, x1, y1 = img["x0"], img["top"], img["x1"], img["bottom"]

            cropped = page.within_bbox((x0, y0, x1, y1)).to_image(resolution
                =300)
            img_path = f"{output_dir}/page{page_index}_img{img_index}.png"
            cropped.save(img_path, format="PNG")
```

Lo *snippet* mostra la logica centrale dello script di estrazione che prepara il layout del PDF in formato JSON. Per ogni pagina del documento, tramite la libreria *PDFPlumber* vengono ottenuti tre insiemi di informazioni: il testo completo (`extract_text`), per una comprensione semantica globale del contenuto, la lista delle singole parole con le relative coordinate (`extract_words`) e le eventuali tabelle (`extract_tables`).

La sezione successiva del codice gestisce l'estrazione delle immagini. L'oggetto `page.images` fornisce infatti una lista di immagini individuate nella pagina, ciascuna accompagnata dalle proprie coordinate geometriche consentendone la memorizzazione della sua *bounding box*. Ciò definisce l'area da ritagliare e verrà convertita in immagine raster tramite il metodo `within_bbox`. L'immagine estratta viene infine salvata in formato PNG e il percorso risultante viene associato alla struttura JSON finale, insieme ai metadati necessari per la ricostruzione del layout nel passaggio all'LLM.

Si riporta un esempio di memorizzazione di metadati relativi ad una parola:

```
{
  "text": "colore",
  "x0": 175.14380000000003,
  "x1": 208.85150000000004,
  "top": 255.69700000000012,
  "doctop": 2805.3190000000004,
  "bottom": 268.6970000000001,
  "upright": true,
  "height": 13,
  "width": 33.70770000000002,
  "direction": "ltr"
}
```

- **text**: il contenuto testuale della parola così come riconosciuto dal motore di estrazione.
- **x0** e **x1**: le coordinate orizzontali del riquadro che racchiude la parola. Il valore `x0` rappresenta il margine sinistro della *bounding box*, mentre `x1` ne indica il margine destro. Questi parametri consentono di disporre il termine lungo l'asse orizzontale.
- **top** e **bottom**: le coordinate verticali superiori e inferiori della *bounding box* della parola. Il valore `top` identifica la distanza dal margine superiore, mentre `bottom` la distanza tra il margine superiore e il margine inferiore del riquadro.
- **doctop**: la coordinata verticale della parola rispetto all'intero documento, non soltanto alla singola pagina. il valore risulta utile per garantire un ordinamento globale degli elementi in PDF composti da più pagine.
- **upright**: un valore booleano che indica se la parola è orientata correttamente (`true`) oppure se è ruotata o disposta secondo altre direzioni.
- **height** e **width**: altezza e larghezza, espresse in punti PDF, della *bounding box* che contiene la parola. Tali misure permettono di inferire informazioni indirette come la dimensione del font o l'allineamento tipografico.
- **direction**: la direzione del testo, generalmente `ltr` (left-to-right). In presenza di scritture verticali o orientamenti differenti, il campo può assumere valori differenti quali `ttb` (top-to-bottom).

5.4 Struttura della richiesta

5.4.1 Pipeline `/api/generate`

La *requestBody* in input prevede l'articolazione in 4 componenti:

- **Blocco *system-like***, ovvero un blocco di istruzioni regolative iniziali che pongono le basi per la gestione del prompt completo vero e proprio.
- **Metadati relativi alle immagini**
- **File PDF inline**
- **Prompt con le istruzioni specifiche**

```
const requestBody = {
    contents: [
        {
            role: "user",
            parts: [{
                text: "Restituisci esclusivamente HTML puro.
                    Non formattare l'output come blocco di codice.
                    Non inserire i delimitatori \`\`\`html o \`\`\`.
                    Inizia direttamente dal primo tag HTML
                    e termina con l'ultimo.
                    Se stai per inserire un blocco \`\`\`html,
                    rimuovilo e restituisci solo il contenuto.
                    Non includere spiegazioni, testo extra o
                    introduzioni/conclusioni."
            }]
        },
        {
            role: "user",
            parts: [
                { text: "Metadata immagini estratte:\n"
                + JSON.stringify(images) },

                { inlineData: { data: base64File,
                mimeType: "application/pdf" } },

                { text: prompt }
            ],
        },
    ],
};
```

5.4.2 Pipeline `/api/generate_JSON`

La *requestBody* in input prevede l'articolazione in 3 componenti:

- **Blocco *system-like***, ovvero un blocco di istruzioni regolative iniziali che pongono le basi per la gestione del prompt completo vero e proprio.
- **Struttura in JSON del PDF**
- **Prompt con le istruzioni specifiche**

```
const requestBody = {
    contents: [
        {
            role: "user",
            parts: [{
                text: "Restituisci esclusivamente HTML puro.
                    Non formattare l'output come blocco di codice.
                    Non inserire i delimitatori \`\`\`html o \`\`\`.
                    Inizia direttamente dal primo tag HTML
                    e termina con l'ultimo.
                    Se stai per inserire un blocco \`\`\`html,
                    rimuovilo e restituisci solo il contenuto.
                    Non includere spiegazioni, testo extra o
                    introduzioni/conclusioni."
            }]
        },
        {
            role: "user",
            parts: [
                {text: "Struttura estratta del PDF:\n" +
                        JSON.stringify(structuredJson)},

                {text:prompt}
            ]
        }
    ]
};
```

5.5 La gestione delle immagini

Avendo adesso le immagini estratte, e la risposta di Gemini, la fase successiva è la sostituzione programmatica dei placeholder immessi dall'LLM. Entrambe le pipeline di PDeFy: la pipeline **inline** e la pipeline **JSON**, convergono in questa stessa fase finale di ricostruzione del documento. Nel caso della pipeline *inline*, il modello dove visualizza delle immagini, posiziona un *placeholder* come `[IMAGE_1]`, `[IMAGE_2]` e così via. L'LLM li posiziona e lascia il compito al *backend* di rimpiazzo con le immagini estratte tramite lo script Python. Ogni immagine estratta è già accompagnata dai dati geometrici (larghezza e altezza) calcolati dallo script Python, che vengono incorporati nellattributo `style` dell'HTML finale per una resa il più fedele possibile al layout originale. Nella pipeline denominata *JSON*, le immagini non sono concretamente visibili dal modello, ma sono contenute all'interno del file estratto contenente l'intera struttura della pagina. Il *backend* ricostruisce la mappatura tra il JSON e le immagini fisiche e procede mediante lo stesso algoritmo di sostituzione.

Il metodo che permette ciò è `replacePlaceholders` di cui si riporta uno snippet. La *function* in questione prende come parametro l'html e le immagini e per ogni immagine effettua un'operazione di sostituzione *hard-coded* costruendo dinamicamente dei tag `<img>` contenente:

- **L'immagine fisica** incorporata in **base 64** direttamente nell'attributo `src` del tag creato
- **Il MIME type** corretto a seconda dell'estensione dell'immagine in questione
- **Le dimensioni** espresse in pixel calcolate durante l'estrazione dal PDF

Viene ritornato il risultato con i placeholder sostituiti.

```
function replacePlaceholders(html, images) {
    images.forEach((img, i) => {
        const placeholder = `[IMAGE_${i + 1}]`;
        const base64 = fs.readFileSync(img.path).toString("base64");
        const imgTag =
            `<img src="${base64}" style="width:${
               img.width}px;height:${img.height}px;" />`;
        html = html.replaceAll(placeholder, imgTag);
    });
    return html;
}
```

L'inclusione delle immagini in formato Base64 direttamente nell'HTML produce un file completamente autosufficiente. Questo approccio:

- elimina il rischio di immagini mancanti o percorsi non risolti;

- garantisce la portabilità del documento e la visualizzazione offline
- consente di utilizzare il file HTML come asset unico, semplificando l'integrazione in applicazioni didattiche o ambienti web.

5.6 Editor di testo e revisione manuale lato utente

L'applicativo prevede una fase di *post-processing* eventuale da parte dell'utente per effettuare eventuali correzioni o modifiche prima dell'esportazione finale. Tale fase avviene tramite un editor progettato per coprire due esigenze distinte: un'interfaccia semplice e accessibile per tutti gli utenti e un'altra che mostri il codice per consentire agli utenti più esperti di avere un maggiore controllo.

5.6.1 Le due visuali dell'editor

La visuale WYSIWYG (*What You See Is What You Get*) permette di modificare il documento HTML come se fosse un normale editor di testo, senza necessità di conoscere la struttura interna del markup.

L'editor utilizzato è `TinyMCE` che consente una sincronizzazione immediata e parallela tra le due tipologie di visualizzazioni. Sono presenti strumenti di formattazione (grassetto, corsivo, liste, tabelle) e fornisce un'anteprima molto simile a quello che sarà il risultato finale. Sono inoltre state aggiunte funzionalità personalizzate, assenti nel componente standard:

- **pulsante di eliminazione della sezione** per ogni esercizio, generato automaticamente nel `DOM` dell'editor.
- **gestione del drag & drop** delle immagini, che consente il riordino fluido degli elementi mantenendone la struttura originale.

Per gli utenti più avanzati è possibile utilizzare direttamente la modalità di editing in linguaggio HTML. Ciò consente il controllo e la modifica totale dei contenuti che si rifletteranno sull'editor e quindi sul risultato finale.

Al termine della revisione, l'utente può esportare il documento come file HTML. Prima del download viene applicato un filtro utile per:

- rimozione elementi ausiliari dell'editor (ad esempio i pulsanti di eliminazione).
- unione struttura HTML e foglio di stile estratto.

Capitolo 6

Analisi dei risultati e valutazione

6.1 Costruzione del dataset

Per effettuare un'analisi dei risultati sono stati presi in esame 10 PDF aventi caratteristiche eterogenee per test ripetuti e valutazione delle performance. Il dataset in esame comprende un bacino di file che possano rappresentare in maniera sufficientemente ampia la variabilità dei materiali didattici della scuola primaria. I PDF in questione sono nativi e rispettano le richieste di *pre-processing* esplicitate nel capitolo 5.

File analizzati	**N. pagine**	**N. esercizi**	**Tipo layout**	**Img**
Calcolo	3	3	Complesso	Si
Aggettivi	1	4	Semplice	No
Grammatica	5	13	Intermedio	Sì
Doppie	2	8	Semplice	Sì
Numeri relativi	4	11	Intermedio	Si
Significato parole	4	4	Semplice	Sì
Sillabe	7	7	Complesso	Si
Storia	3	5	Intermedio	Si
Tecnologia	2	8	Intermedio	Si
Vero/Falso	10	40	Intermedio	Si

Tabella 6.1: Caratteristiche descrittive dei PDF analizzati: numero di pagine, numero di esercizi, complessità del layout e presenza di immagini.

6.2 Metodologia di test e valutazione

Sulla base del dataset di riferimento, si analizzano adesso i risultati ottenuti facendo una prima distinzione tra la modalità *inline* e la modalità *layout JSON*, e poi una seconda distinzione tra i tre modelli di Gemini utilizzati per le sperimentazioni: 2.5 flash, 2.5 pro e 3 pro. La fase di collaudo è stata svolta testando il sistema manualmente e valutando quindi il risultato per ognuno dei 10 file, nelle 2 modalità di invio e per ognuno dei 3 modelli di Gemini .Per ogni casistica assegno un punteggio dato dalla media della valutazione di 3 macro-categorie scelte:

- **Fedeltà visiva**
- **Immagini**
- **Classificazione esercizi**

6.2.1 Fedeltà visiva

Si intende valutare la similarità di layout e la correttezza strutturale dell'output ottenuto. Per effettuare la valutazione ho considerato 5 caratteristiche a cui assegnare {0,1} in modo binario.

- **Ordine dei blocchi**: Si vuole valutare l'ordine coerente di blocchi di contenuto ben separati.
- **Separazione degli esercizi**: Si valuta se la separazione tra i vari esercizi all'interno dei file PDF è chiara e corretta.
- **Gerarchie testuali**: Si valuta se è stata correttamente assegnata la rilevanza del testo all'interno del file. Ad esempio la corretta individuazione di titolo e sottotitoli, e la relativa corretta assegnazione gerarchica espressa mediante stile e dimensione del font.
- **Correttezza liste e tabelle**: si intende osservare se le liste e le tabelle presenti nel file in input siano correttamente riportate nel risultato e se ciò accade nel formato corretto.
- **Allineamento grafico**, si fa una valutazione dell'interpretazione grafica svolta. Si osserva se è stata fatta un'interpretazione complessivamente adeguata del contenuto.

Lo *score* complessivo è definito come:

$$\text{ScoreFedeltà} = \frac{N_{\text{criteri soddisfatti}}}{N_{\text{criteri totali}}}$$

6.2.2 Immagini

La corretta collocazione delle immagini all'interno del risultato costituisce un punto molto critico a cui attribuisco un punteggio:

$$\text{ScoreImmagini} = \frac{N_{\text{img corrette}}}{N_{\text{img totali}}}$$

Con immagini corrette non prendo in considerazione solo il fatto che ci siano nel risultato finale, ma che siano correttamente inserite e contestualizzate in modo coerente.

6.2.3 Classificazione esercizi

Ad ogni esercizio vengono associate una o più classi HTML che identificano la sua tipologia. La misura per il calcolo dell'accuratezza è misurata come:

$$\text{ScoreClassificazione} = \frac{N_{\text{classificati correttamente}}}{N_{\text{totale esercizi}}}$$

6.3 Risultati

I PDF sono stati testati prendendo in considerazione ogni combinazione possibile tra le modalità di invio e i modelli di Gemini. Ciò è stato fatto poiché si ritiene importante non solo la combinazione che crei i migliori risultati, ma anche in che modo la grandezza e la qualità del modello impatti sulla realtà che si sta analizzando.

Il valore ottenuto su ogni cella della tabella è ottenuto calcolando la media dei punteggi, tra 0 e 1, attribuiti per ogni *score* esplicitato precendentemente.

File analizzati	INLINE			JSON		
	2.5 Flash	**2.5 Pro**	**3 Pro**	**2.5 Flash**	**2.5 Pro**	**3 Pro**
Calcolo	0.83	0.75	0.89	0.33	0.75	0.77
Aggettivi	0.90	1.00	1.00	0.90	1.00	1.00
Grammatica	0.55	0.97	1.00	0.85	0.81	0.93
Doppie	0.83	0.94	1.00	0.84	1.00	1.00
Numeri relativi	0.87	1.00	1.00	0.87	1.00	1.00
Significato parole	0.93	1.00	1.00	0.82	0.83	1.00
Sillabe	0.60	0.91	1.00	0.30	0.50	0.70
Storia	0.69	0.91	0.91	0.89	0.91	0.91
Tecnologia	0.86	0.92	1.00	0.81	0.95	0.97
Vero/Falso	1.00	1.00	1.00	0.96	1.00	1.00

Tabella 6.2: Confronto degli score complessivi per file, divisi tra pipeline e modello. *Valori tra 0 e 1: ottenuti come media tra scoreFedeltà, scoreImmagini e scoreClassificazione*

6.3.1 Confronto tra PDF inline e JSON

Si vuole comprendere se sia possibile stabilire, in linea generale, una modalità di invio di dati inerenti al PDF migliore tra le due proposte. Sulla base dei dati raccolti, si calcola una media generale trascendendo il modello utilizzato, e si confrontano i dati ottenuti.

$$\text{Score}_{\text{PDF\_inline}} = 0.9087$$
$$\text{Score}_{\text{PDF\_json}} = 0.8533$$

Dai risultati ottenuti si può constatare come l'invio del PDF *inline* all'LLM consenta ad esso di comprendere in maniera più affidabile il layout, specialmente in presenza di disposizioni molto complesse contenenti molte immagini. La relazione spaziale tra testo e immagini è maggiormente rispettata diminuendo la perdita di informazione strutturale. Con l'invio in *JSON*, vi è una fase di scomposizione in elementi atomici che, seppur con metadati allegati, pone un livello di astrazione intermedio che diminuisce notevolmente la comprensione della struttura in determinati casi. È giusto mettere in luce come in certi casi, i modelli si comportino particolarmente bene a prescindere dalla modalità di invio, facendo così della complessità del documento una variabile molto impattante.

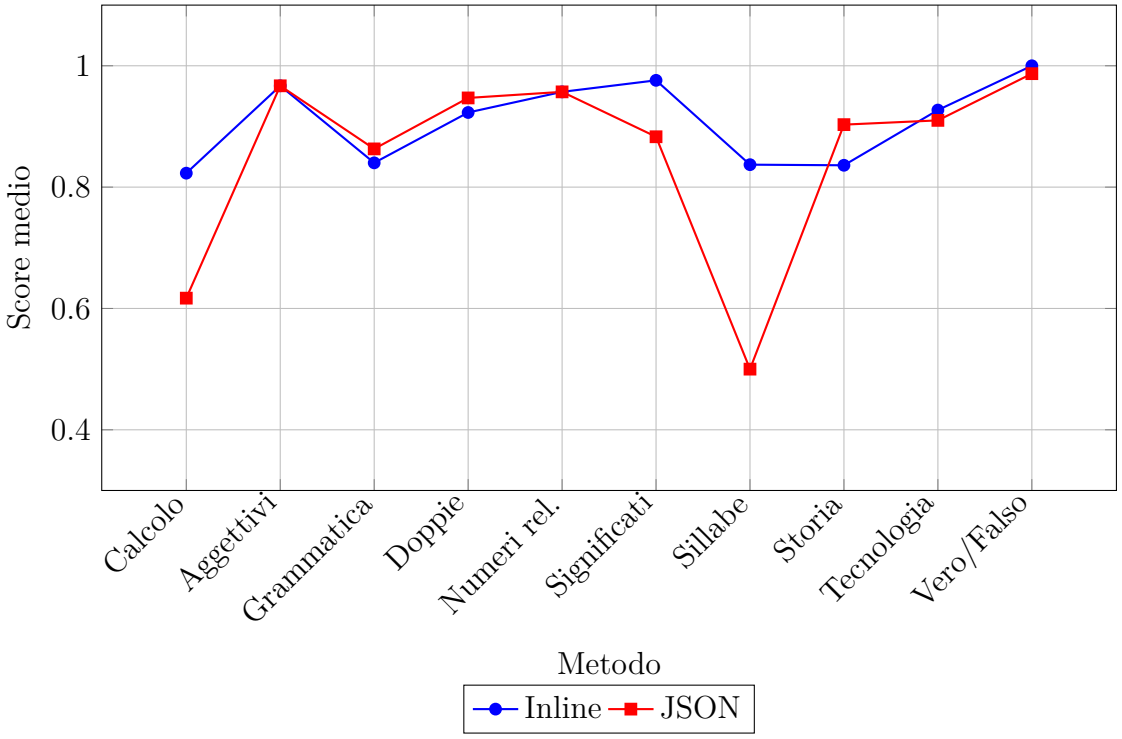

Figura 6.1: Confronto tra pipeline INLINE e JSON tramite score medi per file.

6.3.2 Confronto tra modelli di Gemini

Un'analisi della valutazione delle performance in relazione al modello utilizzato risulta essere utile per comprendere lo stato e l'evoluzione dei modelli che abbiamo a nostra disposizione. Comprendere quanto la grandezza e la potenza del modello impatti su una specifica realtà di riferimento permette di capirne le probabili evoluzioni future e le differenze tra l'utilizzo di modelli aventi costi differenti.

In modo sorprendente l'andamento dei punteggi medi dei metodi in relazione ai modelli di Gemini risulta sostanzialmente uguale, con la linea del metodo *inline* traslato del 5/6 % verso l'alto. L'analisi in questione mostra come la differenza tra i 3 modelli sia particolarmente rilevante ai fini dei risultati, ma anche di come i modelli inferiori testati offrano comunque discreti risultati seppur non ottimali in ottica di automazione. Possiamo considerare il modello 3 pro-preview molto vicino a poter essere utilizzato in una pipeline automatica, che può spesso fare a meno di interventi successivi. Non è, a mio parere, possibile dire lo stesso per gli altri due modelli che non offrono questo tipo di garanzie.

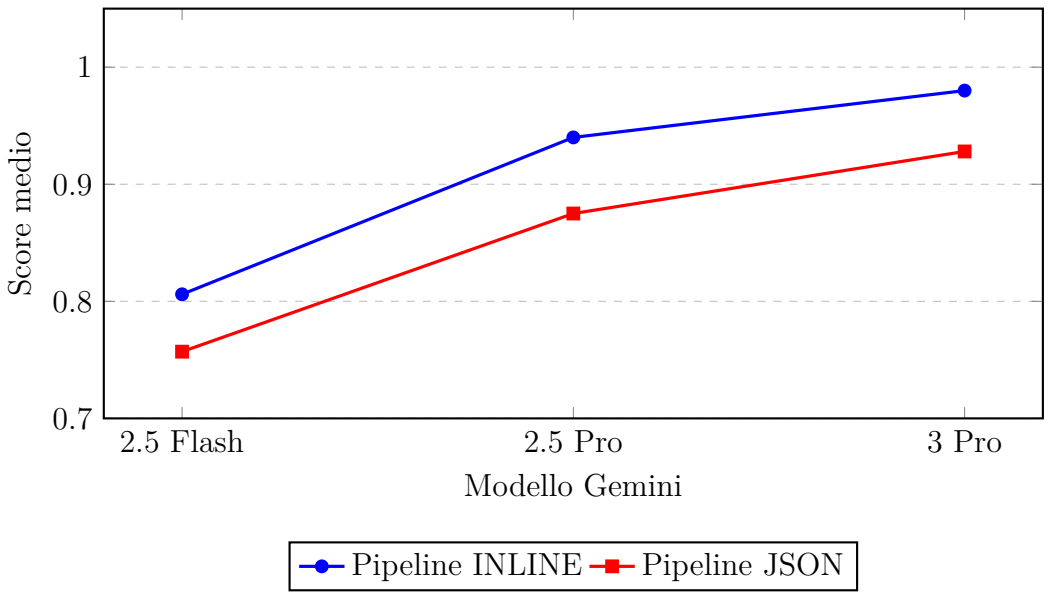

Figura 6.2: Andamento degli score medi per modello Gemini nelle due pipeline (Inline e JSON).

6.3.3 Peso relativo del modello e della pipeline

Un confronto complessivo tra le due variabili in gioco, modalità di invio e modello utilizzato, mostra come entrambi i fattori incidano sul risultato finale, ma con pesi diversi.

Il passaggio da *json* a *inline* comporta un miglioramento medio di circa il 6.5%, segno che preservare la struttura originaria del documento facilita in modo misurabile la ricostruzione del layout da parte del modello.

Il salto tra i tre modelli di Gemini, tuttavia, risulta ancor più determinante: tra *2.5 Flash* e *3 Pro* si osserva un incremento medio complessivo nell'ordine del 20−25%. Ciò indica che, nei casi più complessi, la capacità intrinseca del modello gioca un ruolo predominante rispetto alla modalità di rappresentazione del PDF.

In sintesi, il modello influenza maggiormente la qualità finale, ma la pipeline *Inline* garantisce comunque un miglioramento consistente e sistematico, risultando la scelta preferibile in tutti gli scenari analizzati. Dai risultati, è naturale dire che la combinazione tra invio del PDF inline e modello di *Gemini 3 pro* ha offerto i migliori risultati osservabili dal questo lavoro.

6.4 Limiti

Il progetto prevede delle limitazioni che dipendono sia dalla tipologia di input, sia dal non determinismo offerto dagli LLM e sia dalle risorse limitate

6.4.1 Dipendenza da Gemini

La connessione all'API di Gemini risulta fondamentale nel flusso di lavoro del documento implicandone la dipendenza assoluta. Il mancato funzionamento del servizio di fruizione dei modelli generativi messi a disposizione da Google compromette il funzionamento di PDeFy. Talvolta i modelli di Gemini (soprattutto il modello 2.5-pro e il modello 3-pro-preview) risultano in uno stato di sovraccarico (*overload*), ciò non consentirà l'invio di richieste. Nell'idea di una piattaforma condivisa e mantenuta attiva, i costi alle chiamate API sono considerevoli e variabili a seconda del modello che si intende utilizzare.

I modelli 2.5 flash e pro sono utilizzabili anche senza impostare la fatturazione, ma con minore priorità e con limiti nettamente più stringenti di richieste al minuto (RPM), token in input al minuto (TPM) e richieste al giorno (RPD). Il flusso di richieste di Gemini impatta notevolmente anche sui tempi di elaborazione della richiesta: le misurazioni effettuate sul tempo impiegato dai modelli sono indicativi e non deterministici.

Modello	**Livello gratuito**	**Livello a pagamento (per 1M token)**
Gemini 3 Pro Preview	Non disponibile	Input: \$2 (prompt $\leq$ 200K token) Input: \$4 (prompt $>$ 200K token) Output: \$12 (prompt $\leq$ 200K token) Output: \$18 (prompt $>$ 200K token)
Gemini 2.5 Pro	Senza costi	Input: \$1.25 (prompt $\leq$ 200K token) Input: \$2.50 (prompt $>$ 200K token) Output: \$10 (prompt $\leq$ 200K token) Output: \$15 (prompt $>$ 200K token)
Gemini 2.5 Flash	Senza costi	Input: \$0.30 (testo / immagine / video) Input: \$1.00 (audio) Output: \$2.50

Tabella 6.3: Confronto dei costi tra i modelli Gemini (livello gratuito e a pagamento).

6.4.2 Il non determinismo dei Large Language Model

Il comportamento degli LLM non è completamente deterministico, anche fornendo lo stesso input, i modelli possono restituire risposte sensibilmente differenti. Sull'argomento è importante considerare un parametro di configurazione dei modelli, la temperatura. La

temperatura controlla la creatività del modello, nello specifico di Gemini si assegna un valore tra 0 e 2.

- **0-0.5**, per maggiore determinismo
- **0.7-1**, per un equilibrio tra coerenza e creatività
- **1.1-2**, per risposte più originali e fantasiose

I test all'interno di questo lavoro sono svolti con una temperatura di 1, ovvero quella di default di Gemini tramite API. É rilevante notare come, solitamente, la temperatura per operazioni di questo tipo possa dover essere più vicina allo 0. Sulla base dei test effettuati è stato provato il fatto che utilizzando un prompt strutturalmente solido, una temperatura intermedia non incide sulla qualità dei risultati, e può, anzi, essere utile in certi casi di interpretazione del layout. Astekin et al. hanno condotto nel 2024 un'analisi sistematica di questo fenomeno in relazione al *log parsing* (trasformazione di dati grezzi in dati in una forma strutturata) utilizzando diversi LLM[2]. Alla luce di questo studio, anche nel caso di PDeFy i risultati non saranno perfettamente riproducibili e affidabili. Tramite istruzioni specifiche, *prompt engineering* e pipeline solida si cerca di limitare questa caratteristica, cercando di ottenere degli output sempre il più coerenti possibili. In definitiva, pur adottando strategie mirate alla stabilità del processo, risulta fisiologico che in determinati casi la risposta non sia ottimale e possano essere necessari degli interventi umani per correggere delle imperfezioni.

6.4.3 Lunghezza dei PDF e tempistiche

Il numero delle pagine che vengono caricate per una singola richiesta sono un fattore determinante per la qualità dell'output. È stato testato come un file con un minor numero di pagine fornisca risultati visibilmente più soddisfacenti rispetto ad un file più corposo. Sia per quanto riguarda l'invio di JSON che di file, gli LLM tendono a degradare la qualità della propria risposta in modo direttamente proporzionale alla quantità di materiale in ingresso. Nel caso dell'invio di JSON questo fenomeno è chiamato *attention decay*, che può essere tramutato nel caso di invio di PDF e file in generale in perdita di precisione visiva.

Dai test emerge che la lunghezza in termini di pagine dei PDF in input fa crescere proporzionalmente le tempistiche di elaborazione: Per file tra le 15 e le 25 pagine il tempo di risposta supera spesso i 3 minuti. Si considera quindi PDeFy ottimizzato per PDF in input che non superino le 10 pagine. Questo è vero specialmente nei casi di massiccia presenza di immagini: la qualità dell'output non cala in modo particolare se non vi sono immagini al suo interno.

Le tempistiche di esecuzione di conversioni mediante PDeFy dipendono da diversi fattori quali:

- **Il modello di intelligenza artificiale utilizzato**: modelli più grandi tendono a aumentare progressivamente il tempo necessario per l'analisi
- **Il file in input**, la sua dimensione in termini di pagine e la sua complessità può condizionare la durata
- **La coda interna a Gemini**, che varia a seconda del traffico di chiamate tramite API

Capitolo 7

Conclusioni e sviluppi futuri

Il lavoro presentato ha esplorato una nuova frontiera per quanto riguarda l'analisi e la conversione documentale, mostrando come gli strumenti di intelligenza artificiale se correttamente guidati e incastonati in una pipeline strutturata ed efficace possano, in linea con il progresso dei modelli, offrire buoni risultati. La qualità dell'output sarà sempre fortemente influenzata dalle istruzioni fornite all'LLM, dalla qualità dell'input e dalla coerenza dell'intero flusso di lavoro. La conversione tramite questi strumenti non può ancora definirsi deterministica: variabilità, errori di interpretazione o ricostruzioni errate sono ancora fenomeni possibili. Tuttavia i test effettuati con il modello più performante, ovvero Gemini 3 pro, sono molto promettenti e hanno evidenziato degli ottimi risultati sotto ogni punto di vista analizzato.

Questo porta ad una riflessione generale più ampia: Gli LLM sono uno strumento molto potente e prenderne atto cercando di sfruttarne le peculiarità e integrandoli in modo complementare a strumenti tradizionali può rappresentare un ottimo compromesso. Un approccio ibrido di questo tipo può concretamente portare a fare passi in avanti enormi, certamente nell'ambito della *Document analysis*, ma anche in ogni altro settore di ricerca. Il valore principale di questo lavoro risiede nell'aver implementato e testato una vera e propria un'unione tra questi due mondi in modo da contribuire alla creazione di nuovi spazi di innovazione. La progettazione e l'implementazione dello strumento e dell'applicazione web non è solo una soluzione operativa, ma è un esempio di ingegnerizzazione degli approcci basati su LLM. Il contributo è anche quello di cercare di comprendere a che punto ci si potesse spingere nella realizzazione di strumenti di questo tipo mediante modelli generativi a disposizione di tutti. Ho dimostrato come questo tipo di conversione strutturata sia ottenibile con buona qualità, specialmente con l'invio del PDF inline. Una parte dei limiti individuati nel capitolo precedente, come la variabilità dei modelli generativi, la difficoltà nella gestione di layout particolarmente complessi e i tempi di esecuzione su documenti lunghi, resta inevitabilmente legata allo stato attuale degli LLM. Per ulteriori approfondimenti tecnici e per l'accesso al codice sorgente, è disponibile in bibliografia il riferimento alla repository GitHub del progetto [10].

Alla luce dei risultati ottenuti, il progetto presenta dei margini di miglioramento e nuove possibili direzioni di ricerca.

- **Estensione a nuovi ambiti**, Verificare la robustezza del sistema provandolo per altre tipologie di documenti, estendendolo eventualmente affinché diventi versatile e adattabile.
- **Test con altri LLM**, Il sistema può essere testato con altri modelli (ChatGPT, Claude, Perplexity, Llama ecc.) e aggiornato in relazione ai nuovi LLM che usciranno in futuro, garantendo la longevità del sistema.
- **Ottimizzazione del pre-processing**, automazione delle fasi di *pre-processing* ove necessarie, riducendo il più possibile il lavoro manuale da parte dell'utente.
- **Gestione di file più grandi**, mediante l'impiego di modelli più grandi permettendo la conversione in tempi ragionevoli di PDF particolarmente lunghi, come libri di testo interi.
- **Aggiornamento applicazione web**, con l'integrazione di nuove funzionalità quali editing collaborativo e correzione assistita degli errori.
- **Test del sistema in altre lingue**, valutare il comportamento del sistema con PDF redatti in altre lingue, confrontando la capacità dei modelli di interpretare layout, classificare esercizi e generare HTML semantico in contesti linguistici diversi.
- **Supporto a PDF non nativi tramite OCR**, estendere la pipeline alla gestione di PDF acquisiti tramite scansione, integrando un modulo OCR che consenta l'estrazione e la conversione di PDF con immagini, senza che gli script Python estraggano le pagine intere come immagini.

Appendice A

Prompt completo utilizzato

Prompt utilizzato

```
Istruzioni e Formato di Output:

Struttura e Semantica:
- Utilizza tag HTML semantici corretti: titoli <h1>, <h2>, <
   h3>, ecc.; paragrafi <p>; liste <ul>/<ol>; tabelle <table
   >/<thead>/<tbody>/<tr>/<th>/<td>.
- Mantieni la gerarchia dei titoli originale del PDF.
- Mantieni l'ordine originale del contenuto.

Layout e Stile:
- Assegna classi CSS significative (es. class="nome-sezione
   ") a sezioni ed elementi per facilitare la stilizzazione.
- Includi una sezione <style></style> all'interno di <head>
   per definire uno stile gradevole, pulito e coerente.
- Il file HTML prodotto deve essere autosufficiente (CSS
   inline in <head>).

Esercizi:
- NON risolvere NESSUN esercizio. Riportali fedelmente.
- Ogni esercizio deve essere contenuto in una <section> con
   id esatto: id="exercise-n", dove n è il numero
   progressivo dell'esercizio nel documento (1,2,3,...).
- Mantieni i testi originali.
```

Prompt utilizzato

```
Immagini:
- NON includere immagini reali: sostituisci ogni immagine
   con un placeholder centrato del formato esatto: [IMAGE_X
   ], dove X è l'indice progressivo dell'immagine nel PDF (
   la prima immagine è [IMAGE_1], ecc.).
- Inserisci il placeholder [IMAGE_X] dentro un contenitore <
   div class="image-placeholder"> centrato.
- Fai il div della dimensione adeguata in modo da contenere
   tutta l'immagine e non coprire altre parti del layout.

Metadati delle immagini che ti fornirò:
- page: numero della pagina
- x, y: coordinate del punto superiore sinistro dell'
   immagine nel PDF
- width, height: dimensioni effettive dell'immagine

    Uso dei metadati:
- Le immagini devono essere inserite nellHTML come
   placeholder [IMAGE_X] nello stesso punto logico dedotto
   dal layout originale.
- Usa le coordinate per posizionarle nel punto
   corrispondente al layout del PDF.
-Usa width e height per considerarle nel layout di quella
   dimensione.

Riconoscimento della tipologia di esercizio:
- Identifica automaticamente la tipologia dell'esercizio
   basandoti su parole chiave presenti nel titolo o nell'
   enunciato.
- Aggiungi alla sezione dell'esercizio un attributo class
   che includa la categoria riconosciuta.
  Esempio: <section id="exercise-1" class="exercise exercise
   -completamento">.
- Le categorie principali da riconoscere sono:
  - "completamento" - se compaiono termini come "completa",
   "riempi", "inserisci", "scrivi", "determina","sostituisci
   ".
  - "scelta-multipla" - se compaiono termini come "scegli",
   "seleziona", "indica la risposta corretta", "cerchia".
  - "collegamento" - se compaiono termini come "abbina", "
   collega", "unisci", "metti in relazione".
```

Prompt utilizzato

```
  - "vero-falso" - se compaiono termini come "vero o falso",
    "V/F", "se è vero segna".
  - "ordinamento" - se compaiono termini come "metti in
   ordine", "ordina", "riordina".
  - "domanda-aperta" - se compaiono parole come "spiega", "
   descrivi", "rispondi", "argomenta".
  - "individuazione" - se compaiono parole come "individua",
    "sottolinea".
  - "scrittura" - se compaiono parole come "scrivi", "
   scrivere".
  - "calcolo" - se compaiono parole come "calcolo", "
   addizione", "sottrazione", "moltiplicazione", "divisione
   ".
  - "disegno" - se compaiono parole come "disegno", "
   rappresentazione", "costruisci", "modella", "crea".
    Se nessuna categoria è riconosciuta, assegna class="
   exercise exercise-generico".

    Marca le opzioni o sottosezioni di un esercizio facendo
   capire che fanno riferimento a quel numero di esercizio.

Nel caso di un esercizio identificato come di "completamento
   " o di "scrittura" o di "domanda-aperta"
  - Per ogni immagine ([IMAGE_X]) presente all'interno dell'
   esercizio di completamento, <input type="text" class="
   image-input"/> se c'è un campo di completamento in input.
  - Se c'è uno spazio per il completamento anche senza
   immagine inserisci sempre: <input type="text"/>.

Fine delle istruzioni.
Genera il file HTML seguendo esattamente queste regole.
```

Appendice B

Modifica al prompt principale per invio layout JSON

Blocco da aggiungere al prompt

```
    Formato dell'input (JSON di layout):
- Ti verrà fornito un JSON che rappresenta il contenuto di
   un PDF scolastico.
- Il JSON conterrà elementi testuali e immagini già in
   ordine logico di lettura.
- Per ogni immagine saranno forniti almeno i seguenti
   metadati:
  - page: numero di pagina
  - x, y: coordinate del punto superiore sinistro dell'
   immagine nel PDF
  - width, height: dimensioni effettive dell'immagine
- Utilizza questi dati solo per rispettare la posizione
   logica e la dimensione relativa delle immagini nel layout
    HTML.
```

Bibliografia

[1] Adobe Systems. «History of the PDF Timeline». In: (2023). Ultimo accesso: novembre 2025. URL: https://www.adobe.com/acrobat/resources/pdf-timeline.html.

[2] Astekin, Merve, Hort, Max e Moonen, Leon. «An Exploratory Study on How Non-Determinism in Large Language Models Affects Log Parsing». In: (2024). doi: 10.1145/3643661.3643952 (cs.SE). URL: https://doi.org/10.1145/3643661.3643952.

[3] Bhattacharyya, Aniket et al. «Information Extraction from Visually Rich Documents using LLM-based Organization of Documents into Independent Textual Segments». In: (2025). arXiv: 2505.13535 [cs.IR]. URL: https://arxiv.org/abs/2505.13535.

[4] Chen, Banghao et al. «Unleashing the potential of prompt engineering for large language models». In: *Patterns* 6.6 (2025). doi: 10.1016/j.patter.2025.101260 (cs.AI). URL: https://doi.org/10.1016/j.patter.2025.101260.

[5] Deep Search Team. «Docling Technical Report». Ver. 1.0.0. In: (ago. 2024). DOI: 10.48550/arXiv.2408.09869. eprint: 2408.09869. URL: https://arxiv.org/abs/2408.09869.

[6] Google DeepMind. «Gemini Model Overview». In: (2025). Ultimo accesso: novembre 2025. URL: https://ai.google.dev/models/gemini.

[7] Huang, Yupan et al. «LayoutLMv3: Pre-training for Document AI with Unified Text and Image Masking». In: (2022). arXiv: 2204.08387 [cs.CL]. URL: https://arxiv.org/abs/2204.08387.

[8] Lasheb, Mohamed Amine et al. «Extracting and Structuring Textbooks for Inclusive Education: A Computer Vision Approach». In: *Proceedings of the 25th IEEE International Conference on Advanced Learning Technologies (ICALT 2025)* (2025). URL: https://hal.science/hal-05206586.

[9] Luo, Chuwei et al. «LayoutLLM: Layout Instruction Tuning with Large Language Models for Document Understanding». In: (2024). arXiv: 2404.05225 [cs.CV]. URL: https://arxiv.org/abs/2404.05225.

[10] Manieri, Alessio. «PDeFy». In: (2025). URL: https://github.com/Manieri03/PDeFy_Tesi.

[11] Maslej, Nestor et al. «Artificial Intelligence Index Report 2025». In: *arXiv preprint arXiv:2504.07139* (2025). Ultimo accesso: novembre 2025. URL: https://arxiv.org/abs/2504.07139.

[12] pdf2htmlEX Community. «pdf2htmlEX — Convert PDF to HTML while retaining text, format and vector graphics». In: (2025). Ultimo accesso: novembre 2025. URL: https://github.com/pdf2htmlEX/pdf2htmlEX.

[13] PyMuPDF Community. «Extracting Images from PDFs — PyMuPDF Documentation». In: (2025). Ultimo accesso: novembre 2025. URL: https://pymupdf.readthedocs.io/en/latest/recipes-images.html.

[14] Shinyama, Yusuke. «PDFMiner — Python PDF Parser and Analyzer». In: (2025). Ultimo accesso: dicembre 2025. URL: https://pdfminer-docs.readthedocs.io/.

[15] Singer-Vine, Jeremy. «pdfplumber — Tools for Extraction and Parsing of PDF Documents». In: (2025). Ultimo accesso: novembre 2025. URL: https://www.pdfplumber.com/.

[16] Tiny Technologies Inc. «TinyMCE — Rich Text Editor». In: (2025). Ultimo accesso: novembre 2025. URL: https://www.tiny.cloud/.

Ringraziamenti

Desidero rigraziare il professor Angelo Di Iorio per la sua disponibilità e per il prezioso supporto ricevuto in questi mesi. La sua guida è stata di fondamentale importanza per la realizzazione di questo progetto.

Ringrazio mia madre e mio padre, Nadia e Fabrizio per il sostegno morale ed economico. Grazie per la fiducia che mi avete sempre dimostrato e per avermi dato la libertà di costruirmi il mio percorso senza pressioni, lo avete reso più sereno con la vostra presenza, discreta ma fondamentale.

Ringrazio mio fratello Riccardo, per avermi insegnato ad apprezzare fino in fondo i momenti da studente. Grazie per aver dato un valore enorme e significativo alle poche giornate che siamo riusciti a passare insieme in questi anni.

Ringrazio i miei nonni, Marisa, Renato e Gabriella, per l'affetto e i valori che hanno saputo trasmettermi e i miei zii, Silvia, Antonio insieme a mia cugina Adele, per i momenti sereni passati insieme.

Ringrazio i miei amici di vecchia data, perchè nonostante passino gli anni il nostro legame rimane saldo, e anche quando ci sentiamo meno, sappiamo che , se serve, ci saremo sempre l'uno per l'altro.

Ringrazio i miei amici di corso Alessandro e Daniele per tutto il tempo passato insieme tra lezioni, progetti, pause pranzo ed esami. Avete contribuito a rendere questo periodo universitario davvero speciale.

Ringrazio gli amici di "Rassmuss" perchè mi hanno accolto nel loro gruppo già consolidato sin dal primo giorno ponendo le basi per la nascita di amicizie destinate a durare nel tempo. Grazie per le risate e per la leggerezza che mi avete portato anche nei momenti più difficili.

Ed infine ringrazio Ilaria, che più di tutti si è sorbita i miei momenti no, che ha saputo ascoltarmi e darmi forza quando non ne avevo. Lei che, allo stesso tempo, ha saputo donarmi tutto il suo entusiasmo nei momenti felici, trasmettendomi tutto il suo appoggio e la sua stima nei miei confronti. Grazie per aver creduto in me più di quanto io stesso riuscissi a fare. Grazie di esserci stata in questo viaggio e di continuare ad esserci.

Grazie a tutti.