ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Alma Mater Studiorum - Università di Bologna

Corso di Laurea in Ingegneria e Scienze Informatiche

Analisi di documenti PDF strutturati ed estrazione dati

Nome Tirocinante: Jiekai Sun
Tutor Didattico: Chiara Ceccarini
Laboratorio: DISI - Dipartimento di Informatica - Scienza e Ingegneria
Periodo del Tirocinio: 26 Febbraio 2025 - 6 Aprile 2025

18 aprile 2025

1 Introduzione

Il presente tirocinio è stato svolto presso il DISI - Dipartimento di Informatica - Scienza e Ingegneria nell'ambito del progetto AlmAware. L'attività ha riguardato l'analisi e l'estrazione di dati dal bilancio di sostenibilità 2024 dell'Alma Mater Studiorum - Università di Bologna. I dati di interesse, prevalentemente rappresentati tramite grafici, tabelle e infografiche all'interno del documento PDF, non erano disponibili in formato sorgente, il che ha reso l'operazione di estrazione una sfida significativa.

L'obiettivo del tirocinio è stato quello di studiare, sviluppare e testare metodologie efficaci per la conversione di tali dati in un formato strutturato, come ad esempio JSON, includendo anche i metadati contestuali. L'intento finale è di automatizzare il più possibile questo processo, minimizzando la necessità di intervento manuale.

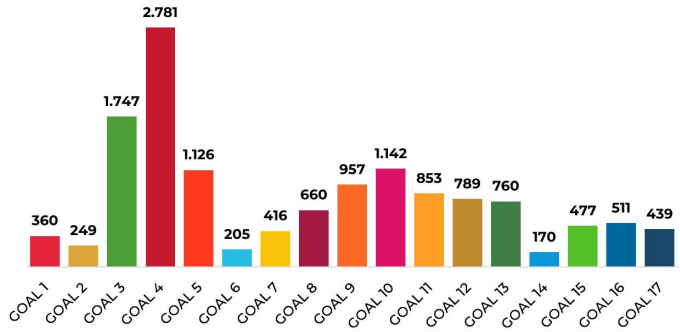
Di seguito sono riportate alcune immagini esemplificative delle modalità grafiche di rappresentazione dei dati presenti nel documento analizzato.

Figura 1: Pagina contenente infografica

4.5 IMPATTO SUGLI OBIETTIVI DI SOSTENIBILITÀ DELL'ONU SDGS

L'Università di Bologna contribuisce al perseguimento degli Obiettivi di sviluppo sostenibile (SDGs) attraverso i propri insegnamenti. Il numero complessivo degli insegnamenti associati ad almeno un SDGs è 5.545. Un singolo insegnamento è conteggiato più volte quando corrisponde a più obiettivi.

Figura 32 – Insegnamenti per SDGs



80

Figura 2: Pagina contenente grafico a barre

2 Tecnologie

Durante il tirocinio, l'attività si è concentrata principalmente sull'impiego del linguaggio di programmazione Python per lo sviluppo degli script necessari all'estrazione e all'elaborazione dei dati.

Inizialmente, è stata condotta una fase di valutazione di diverse tecnologie di Large Language Models (LLM). Questa valutazione ha incluso sia soluzioni commerciali (ChatGPT-4o, Google Gemini, Microsoft Copilot) sia modelli open-source eseguiti in locale attraverso piattaforme come Ollama e LM Studio.

L'esplorazione di LLM locali tramite Ollama e LM Studio è stata motivata dalla volontà di valutare un'alternativa all'utilizzo di API cloud, prestando particolare attenzione alle problematiche relative alla privacy dei dati.

Sono stati testati diversi modelli open-source con capacità di visione (tra cui `llava`, `llama3.2-vision`, `minicpm-v` e `granite3.2-vision`), ma le prime impressioni hanno evidenziato problemi di incompletezza o incoerenza nelle risposte.

Le fasi di test e sviluppo sono state eseguite su due configurazioni hardware basate su sistemi operativi Windows: una macchina dell'ateneo (CPU Intel Core i7-13700T, 32GB RAM, GPU integrata UHD 770) e il computer personale (CPU Intel Core i7-12700F, 16GB RAM, GPU dedicata AMD RX 6600 8GB VRAM).

3 Attività

3.1 Analisi e valutazione di strumenti OCR

Le analisi iniziali hanno rivelato che gli strumenti OCR tradizionali come `pyplumber`, `pypdf` e `Tesseract` non sono adeguati per interpretare accuratamente le informazioni grafiche presenti nel bilancio di sostenibilità. Tali strumenti restituiscono un output non strutturato difficile da interpretare, evidenziando una significativa limitazione nella comprensione dei dati grafici.

È stata anche valutata la possibilità di utilizzare piattaforme commerciali come Microsoft Azure e Google Cloud Platform, le quali offrono soluzioni OCR avanzate. Tuttavia, tali soluzioni si sono rivelate anch'esse limitate nella capacità di comprendere il contesto e il significato dei dati grafici, rendendole poco ideali per i requisiti specifici del problema.

3.2 Esplorazione di soluzioni LLM

Successivamente, l'attenzione si è spostata sull'utilizzo di Large Language Models (LLM). I primi test con GPT-4o di OpenAI hanno mostrato risultati promettenti in termini di qualità e velocità di elaborazione, sebbene sia emersa una difficoltà nell'etichettatura precisa dei dati estratti. Sono stati testati anche altri LLM (Gemini 2, Copilot, Mistral e Claude), ma si sono ripresentate problematiche analoghe relative all'accuratezza dell'etichettatura.

Durante la sperimentazione, si è notato che gli LLM hanno ancora difficoltà intrinseche nella lettura e interpretazione precisa di grafici complessi. Nonostante questa limitazione, si è comunque deciso di proseguire con tale approccio, prevedendo di intervenire manualmente sui risultati estratti per correggere eventuali imprecisioni o mancanze.

3.3 Strategia generale

La strategia adottata per l'estrazione di dato da un file PDF consiste nei seguenti passaggi:

1. **Conversione del PDF in immagini:** Trasformare ogni pagina del PDF in un file immagine (il formato jpeg è preferito in quanto più leggero).
2. **Creazione di un indice:** Definire una struttura ad albero (indice) che rifletta l'organizzazione del documento PDF, associando ad ogni sezione/capitolo i rispettivi numeri di pagina. Tale indice può essere creato manualmente o con l'ausilio di LLM.
3. **Elaborazione delle immagini con un LLM:** Inviare ogni immagine all'API di un LLM assieme ad un prompt.
4. **Ricezione dei dati estratti:** Ottenere in risposta da ogni chiamata un JSON contenente le informazioni identificate nell'immagine.
5. **Unione dei dati nell'indice:** Integrare i dati JSON estratti da ogni pagina all'interno della struttura dell'indice JSON, utilizzando i numeri di pagina come riferimento per l'inserimento.

3.4 Implementazione

L'implementazione di questa pipeline è stata realizzata attraverso una serie di script:

- `convert_pdf.py`: Questo script utilizza la libreria `pdf2image` per prendere in input il file PDF e generare una serie di file immagine in formato JPEG, uno per ogni pagina del documento.
- `gemini_call.py`: Questo modulo contiene la logica necessaria per comunicare con l'API di Gemini 2. Include la definizione della funzione `process_image` che effettua la chiamata all'API per una singola immagine. Il prompt è integrato nello script ed è facilmente modificabile.
- `extract_data.py`: Questo script automatizza l'invio di richieste batch all'API di Gemini 2. Gestisce la frequenza di chiamate al fine per evitare di sovraccaricare l'API e invoca la funzione `process_image` (definita in `gemini_call.py`) per ogni immagine di interesse.
- `fill_index.py`: Questo script ha lo scopo di popolare un indice JSON preesistente con i dati estratti nei singoli JSON relativi ad ogni pagina. Nel caso in esame, la struttura del file JSON di indice è stata semplificata per organizzare le informazioni per capitoli, includendo il loro nome e numero di pagina.

Nota: La scelta di utilizzare l'API di Gemini 2 è stata motivata dalla disponibilità di un piano gratuito che offre un numero di richieste al minuto (15) più che sufficiente per le esigenze del progetto.

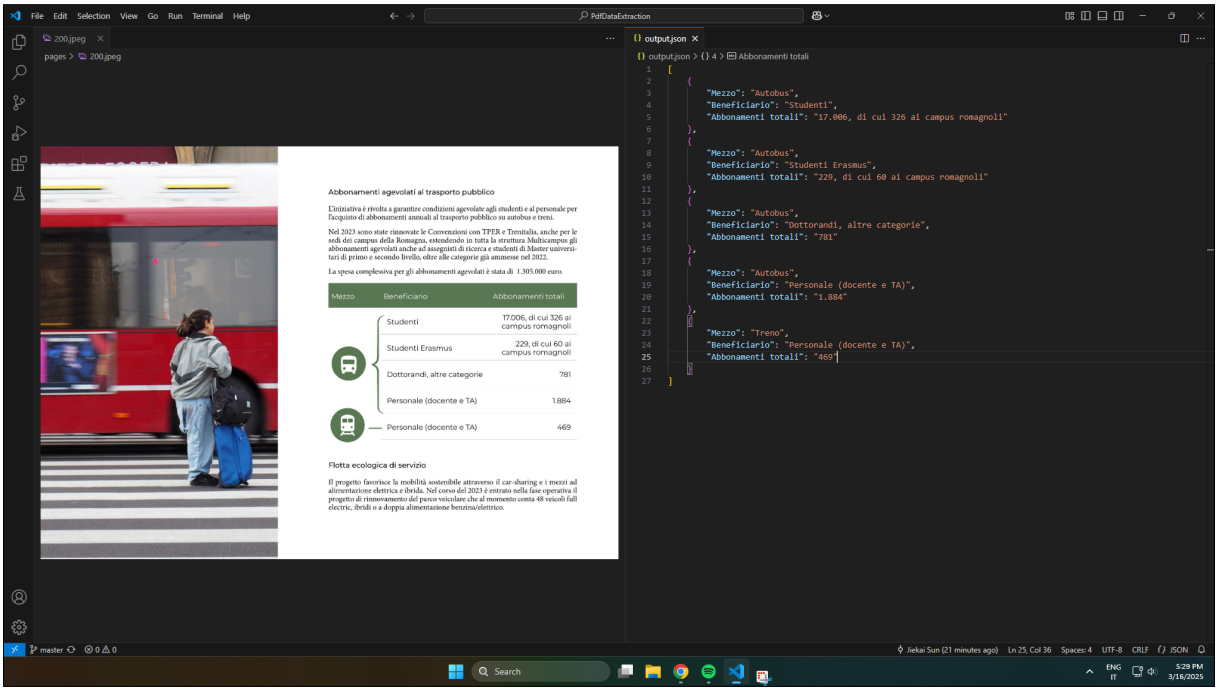


Figura 3: Esempio di output ottenuto con Gemini 2

```
Analizza l'immagine fornita, che fa parte di un bilancio di sostenibilità.
Il tuo obiettivo è estrarre dati strutturati rilevanti per la valutazione delle performance di sostenibilità dell'azienda.
Criteri di rilevanza:
- Concentrati su tabelle, grafici e infografiche che presentano dati numerici (percentuali, valori monetari, indicatori di performance).
- Escludi slide che contengono esclusivamente indici o sommari, informazioni di contatto, introduzioni generiche,
immagini senza dati numerici, solo titoli o intestazioni, crediti o ringraziamenti.
Per quanto riguarda l'output:
- Estrai i dati in JSON valido e ben strutturato.
- Usa le chiavi in snake case, senza spazi o accenti, in lingua italiana. Per i valori non numerici mantieni il formato originale.
- Struttura i dati in modo gerarchico, usando array e oggetti quando necessario.
- Usa interi o float a seconda dei valori numerici che trovi.
- Mantieni le percentuali come stringhe.
- Per le posizioni cardinali usa solo interi.
- Separa le unità di misura dai valori numerici, salvandoli a parte se presenti (`322.561.252,71 euro` -> `322561252.71`).
- Assegna titoli significativi ai dati estratti, evitando prefissi come `"figura"`, `"tabella"`, etc.
- Se l'immagine non soddisfa i criteri di rilevanza, restituisci un JSON vuoto: `{}`.
```

Figura 4: Esempio di prompt utilizzato

BS2024-gemini2-index.json


```json
{
    "capitoli": [
        {
            "numero": 1,
            "nome": "STRATEGIA",
            "pagina": 10,
            "data": {
                "ranking_universitario": [
                    {
                        "nome": "QS World University Ranking",
                        "posizione_italia": 3,
                        "posizione_mondo": 154
                    },
                    {
                        "nome": "Times Higher Education World Ranking",
                        "posizione_italia": 1,
                        "posizione_mondo": 155
                    },
                    {
                        "nome": "QS Sustainability",
                        "posizione_italia": 1,
                        "posizione_mondo": 108
                    },
                    {
                        "nome": "The Impact SDGs 5 UN 2030 (Uguaglianza Di Genere)",
                        "posizione_italia": 1,
                        "posizione_mondo": 2
                    },
                    {
                        "nome": "Green Metric",
                        "posizione_italia": 1,
                        "posizione_mondo": 12
                    },
                    {
                        "nome": "Censis",
                        "posizione_italia": 1,
                        "commento": "Mega Atenei"
                    }
                ],
                "strutture": {
                    "scuole": 5,
                    "dipartimenti": 31
                },
                "costi_per_nome_strategico": {
                    "unita_di_misura": "mln di euro",
                    "ambiti": [
                        {
                            "nome": "Persone (stipendi)",
                            "percentuale": "38%"
```


Figura 5: Porzione del risultato finale

3.5 Considerazioni sulla privacy e utilizzo di LLM locali

Una criticità emersa durante l'attività riguarda la privacy, in quanto il piano gratuito di Gemini 2 prevede l'utilizzo dei dati forniti per l'addestramento dei propri modelli. Per affrontare questa problematica, è stata esplorata la possibilità di utilizzare LLM in locale tramite piattaforme come Ollama e LM Studio.

Entrambe offrono una modesta gamma di LLM, ma si è preferito l'utilizzo di Ollama in modalità CLI (Command Line Interface), per una maggiore flessibilità e controllo.

Sono stati testati diversi modelli open-source con capacità di analisi delle immagini (*vision capable*), tra cui `llava`, `llama3.2-vision`, `minicpm-v` e `gemma3`. Questi modelli differiscono per numero di parametri (misurati in miliardi) e quindi requisiti di memoria. In generale, caricare i modelli interamente sulla GPU è preferibile al fine di evitare notevoli rallentamenti dovuti all'offloading, anche parziale, su CPU e RAM.

I test hanno mostrato che, sebbene promettenti, i modelli locali presentano tempi di inferenza decisamente più lunghi e una qualità delle risposte generalmente inferiore rispetto ai modelli cloud.

In ogni caso è stato sviluppato lo script `extract_data_with_ollama.py` per interagire con modelli fatti girare con Ollama, compito semplificato dalla presenza della libreria ufficiale ollama-python, ma a causa delle limitazioni hardware (GPU con solo 8 GB di VRAM), sono stati eseguiti solo test con modelli leggeri (calibrati intorno a 7-8 miliardi di parametri), che hanno portato risultati inadeguati a fronte della complessità del problema.

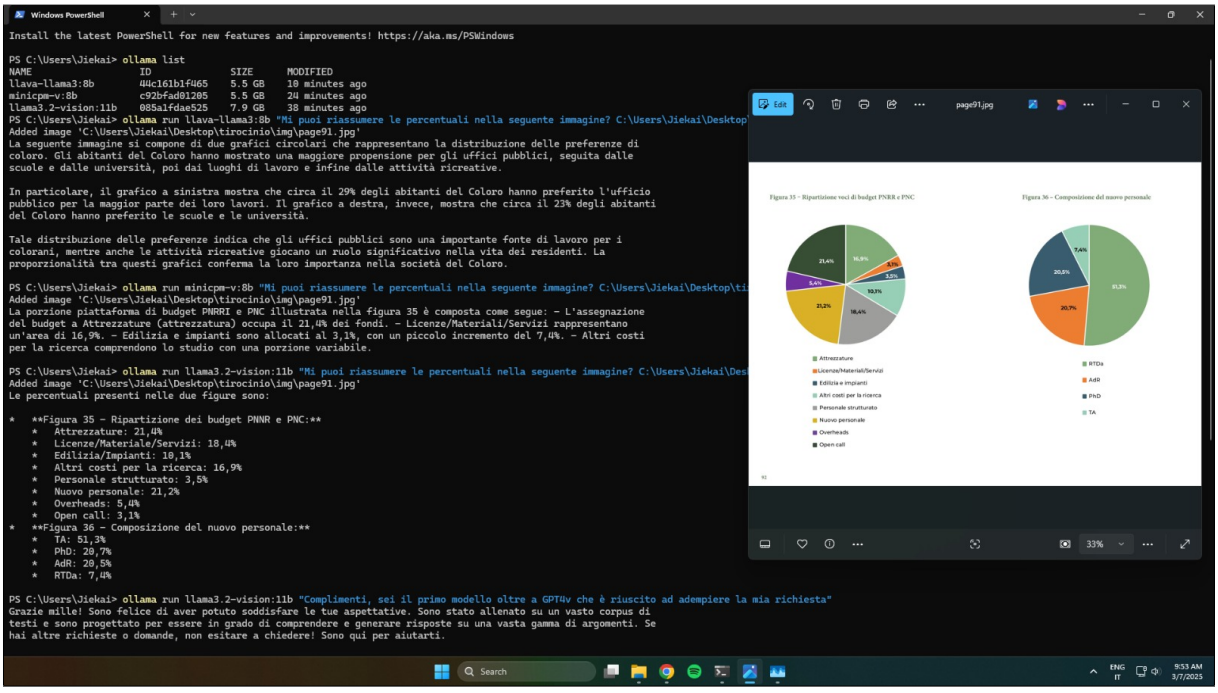


Figura 6: Risultati di tre modelli locali

4 Conclusioni

Sfortunatamente, l'obiettivo iniziale di automatizzare completamente il processo di estrazione dati non è stato pienamente raggiunto. Tuttavia, la soluzione implementata, basata su una pipeline di script Python e sull'utilizzo dell'API di Gemini 2, ha permesso di automatizzare circa il 65-70% del flusso di lavoro. Il restante 30-35% ha richiesto un intervento manuale per la revisione e la formattazione degli output ottenuti. Uno dei problemi principali è stata l'interpretazione di pagine in serie che condividono un contesto comune, in quanto la strategia adottata analizza ogni pagina individualmente, senza mantenere una memoria del contesto precedente. Un altro problema è stato l'impossibilità di ottenere uno stile di output coerente, in quanto ogni chiamata all'API restituiva un risultato indipendente dalle altre, pur utilizzando lo stesso prompt.

L'esperienza ha evidenziato che, allo stato attuale non è possibile affidarsi completamente all'intelligenza artificiale per un compito del genere, ma il suo contributo può ridurre significamente i tempi di estrazione. In prospettiva futura, si prevede un continuo miglioramento di tali LLM, sia commerciali che open-source, con un aumento delle prestazioni, riduzione dei consumi energetici e una maggiore compatibilità con hardware di tipo consumer-level.

5 Risorse

I codici sviluppati e i risultati ottenuti durante il tirocinio sono disponibili al seguente repository: `https://github.com/until-it-breaks/PdfDataExtraction`

Il backend utilizzato da Ollama e LM Studio è `llama.cpp`, il quale è stato adattato per sfruttare diverse risorse hardware tramite tecniche quali CPU, ROCm, CUDA oppure Vulkan. In assenza di accelerazione GPU, Ollama di default utilizza la CPU per l'inferenza, il che porta ad un notevole rallentamento del processo. È importante notare che Ollama, nella sua versione ufficiale, non supporta alcune GPU, anche recenti. La GPU in propria dotazione non rientrava tra quelle supportate e, per superare questa limitazione, sono stati usati fork di Ollama che abilitano tale supporto tramite Vulkan o forzando l'utilizzo di ROCm.