

# INTELLIGENT PARSING

Francesca Erika Rinaldi



Uso di Large Language Models (LLM) in pipeline RAG per migliorare le risposte. Si approfondiscono parsing dei dati da documenti locali.

Github

# Contents

<b>1</b>	<b>Sistema Attuale</b>	<b>3</b>
<b>2</b>	<b>Sistema Proposto</b>	<b>3</b>
<b>3</b>	<b>Specifica PEARS</b>	<b>4</b>
3.1	Performance: . . . . .	4
3.2	Environment: . . . . .	4
3.3	Actuators: . . . . .	5
3.4	Sensors: . . . . .	5
<b>4</b>	<b>Soluzione proposta</b>	<b>6</b>
<b>5</b>	<b>Sviluppo del dataset</b>	<b>6</b>
<b>6</b>	<b>L'importanza del Parsing dei documenti</b>	<b>7</b>
<b>7</b>	<b>Codice</b>	<b>8</b>
7.1	Inizializzazione e Parsing degli Argomenti . . . . .	8
7.2	Query RAG . . . . .	9
7.3	Preprocessing dei documenti . . . . .	10
7.4	Splitting dei documenti . . . . .	11
7.5	Gestione database vettoriale . . . . .	12
<b>8</b>	<b>Esempio di esecuzione completa</b>	<b>13</b>
<b>9</b>	<b>Conclusioni</b>	<b>16</b>
<b>10</b>	<b>Glossario</b>	<b>16</b>

# 1 Sistema Attuale

Il progetto PARSING PDFIA rappresenta davvero un passo avanti nel campo del parsing. Il vantaggio nel contesto attuale del parsing dei documenti risiede principalmente nell'integrazione avanzata di Large Language Models (LLM) con una pipeline di Retrieval Augmented Generation (RAG). Rispetto ai sistemi tradizionali di parsing, che spesso si limitano a estrarre informazioni strutturate o semi strutturate da documenti, l'uso di LLM all'interno di una pipeline RAG consente di gestire grandi quantità di dati non strutturati in modo più efficace. Questa architettura non solo migliora la capacità di comprendere il contenuto semantico dei documenti, ma permette anche di personalizzare le risposte alle query basandosi su contesti specifici e aggiornati, derivanti direttamente dai dati locali. Il progetto facilita la manipolazione e l'interpretazione di dati complessi in modo flessibile, andando oltre le limitazioni degli approcci di parsing tradizionali.

# 2 Sistema Proposto

Lo scopo del mio progetto è quello di realizzare un sistema di analisi di documenti orientato alla RAG specializzato nel parsing di documenti comuni come PDF, DOC e PPTX. Lo sviluppo è focalizzato sulla fase di preprocessing dei documenti eliminando contenuti non testuali come tabelle ed immagini, sostituendoli con descrizioni testuali contenenti le stesse informazioni in modo tale che possano poi essere elaborate dal LLM nella generazione della risposta alla query dell'utente. Nel progetto viene usata una pipeline RAG che si basano fortemente sul Natural Language Processing (NLP). Questi sistemi combinano modelli di retrieval con modelli generativi di NLP per elaborare e rispondere alle query. Il sistema utilizza il NLP per rendere più precisa e mirata l'interpretazione del linguaggio naturale, consentendo all'LLM di generare risposte dettagliate, personalizzate e altamente pertinenti.

## 3 Specifica PEARS

Di seguito ho elencati la specifica P.E.A.S. dell'agente intelligente

### 3.1 Performance:

Le prestazioni del NLP sono valutate attraverso le seguenti misure:

- **Accuratezza nella comprensione delle query:** L'agente deve interpretare correttamente le query dell'utente e rispondere con informazioni pertinenti e coerenti.
- **Efficienza del parsing e della generazione di risposte:** Tempi di risposta brevi per migliorare l'interazione e l'esperienza dell'utente.
- **Precisione nell'estrazione di dati rilevanti :** Capacità d'identificare le informazioni pertinenti all'interno di documenti locali per rispondere efficacemente alle richieste
- **Adattabilità ai formati di documenti diversi:** Capacità di processare vari formati di file (PDF, DOCX,PPTX) con la stessa efficienza e accuratezza.

### 3.2 Environment:

L'ambiente in cui opera l'agente intelligente ha le seguenti caratteristiche:

- *Parzialmente osservabile:* l'agente può accedere ai documenti locali e interpretarli, né ha la visibilità sui contenuti esterni ai documenti forniti. Le query dell'utente guidano l'estrazione e la generazione di risposte, quindi, l'osservabilità è limitata ai dati disponibili nei documenti e alle informazioni esplicitate nella query.
- *Non deterministico:* le risposte dell'agente non sono predefinite poiché dipendono dalla varietà di contenuti presenti nei documenti, della formulazione delle query dell'utente e dalle possibili interpretazioni delle stesse. Le risposte possono variare anche con l'aggiornamento dei documenti, li rende l'ambiente meno prevedibile.

- *Sequenziale*: l'agente risponde a una query alla volta, e le risposte possono influenzare le domande successive. L'ordine delle query dell'utente e il contesto generato sono importanti per ottenere risposte significative e correlate.
- *Statico*: dal punto di vista dell'agente, in quanto i documenti locali rimangono invariati durante il processo di risposta a una query specifica. L'ambiente diventa dinamico se i documenti vengono aggiornati o modificati frequentemente, poiché tali cambiamenti influenzeranno le risposte future.
- *Discreto*: le interazioni con l'agente avvengono tramite una serie finita di query e risposte. Ogni interazione è una singola unità che si aggiunge a una sequenza discreta di richieste e risposte, facilitando la gestione e l'interpretazione del contesto.
- *Singolo agente*: l'agente intelligente opera in modo indipendente, senza necessità di collaborare con altri agenti per rispondere alle query dell'utente.

### 3.3 Actuators:

- **Motori di parsing NLP**: consente di analizzare, estrarre e strutturare i dati rilevanti dai documenti.
- **Pipeline di RAG**: Integra il contesto astratto direttamente nella query per migliorare la coerenza e rilevanza della risposta.

### 3.4 Sensors:

- **Rilevatori di struttura testuale e metadati**: Utilizza algoritmi di NLP che identificano ed estraggono sezioni rilevanti e informazioni chiave dei documenti.
- **Monitoraggio delle prestazioni del parsing e del LLM**: Misura l'accuratezza delle estrazioni e la qualità delle risposte generate, permettendo eventuali aggiustamenti e ottimizzazioni per garantire la qualità del servizio.

## 4 Soluzione proposta

Nel progetto è stato utilizzato un algoritmo che sfrutta anche tecniche di Natural language Processing (NLP) per diverse fasi della pipeline di elaborazione. Le tecniche NLP sono impiegate principalmente per estrarre, pulire e preparare il testo dei documenti, convertendo tabelle in linguaggio naturale, estraendo il testo dalle immagini tramite OCR, frammentando il contenuto in chunk di dimensioni gestibili dal modello. Il modello di linguaggio Llama3 utilizza avanzate tecniche NLP per comprendere il contesto e generare risposte coerenti. Questi processi di NLP sono fondamentali per assicurare che il testo fornito come contesto sia chiaro, pertinente e comprensibile per il modello, migliorando l'accuratezza e la coerenza delle risposte prodotte. Il sistema sfrutta l'integrazione tra LLM e un database vettoriale Chroma per migliorare l'efficacia e la pertinenza delle risposte alle query dell'utente. Utilizzando embedding vettoriali per arricchire il contesto di input al modello.

## 5 Sviluppo del dataset

Nel mio codice ho usato un database Chroma, che viene inizializzato con una funzione che viene usata per fare embedding, che genera gli embedding dei testi. Questo serve per trasformare i documenti in vettori che possono essere salvati. Poi calcolo gli ID dei chunk, questo ID è creato sulla base del percorso del documento sorgente e dalla posizione del chunk, così ogni documento può essere tracciato e identificato correttamente nel db. Gli ID dei documenti già presenti nel database vengono recuperati. In Chroma, la persistenza dei dati è gestita automaticamente, quindi non è necessario un comando esplicito di salvataggio o chiusura del database.

## 6 L'importanza del Parsing dei documenti

Il *parsing* dei documenti è un campo di ricerca e sviluppo in continua evoluzione grazie ai progressi nella tecnologia di elaborazione del linguaggio naturale, riconoscimento ottico dei caratteri e apprendimento automatico. Tuttavia, emergono sfide significative come la bassa qualità dei documenti scansionati per le tecniche di OCR, la varietà di layout dei documenti PDF che richiede modelli flessibili in grado di adattarsi ai diversi formati e l'estrazione solo del testo non è sufficiente per comprendere il contesto e la struttura logica dei documenti. A queste sfide ci sono alcune prospettive future molto promettenti, come una maggiore integrazione di tecniche avanzate di intelligenza artificiale per migliorare la comprensione ed estrazione di informazioni, lo sviluppo di strumenti open source, come sistemi automatizzati in grado di gestire l'intero processo, dall'estrazione del testo fino all'analisi delle informazioni. Durante il progetto è stato effettuato uno studio del parsing per poter manipolare non solo il testo di un documento PDF, ma anche le tabelle, trasformate in linguaggio naturale, così che di potesse comprendere al meglio il contesto in cui si sta lavorando. Il parsing dei documenti PDF è il processo di analisi e interpretazione del contenuto di un file PDF. Questo processo consente di estrarre informazioni strutturate e dati da un documento PDF in modo da poterli manipolare. Se non si esegue il parsing di un documento, si rinuncia alla possibilità di estrarre, strutturare e comprendere automaticamente le informazioni contenute al suo interno. In pratica, il documento rimane un semplice file non analizzato e ogni sua parte è tratta come un blocco unico di dati non interpretabile dai sistemi automatizzati. Senza il parsing i modelli di intelligenza artificiale e gli strumenti di analisi non possono accadere direttamente ai contenuti impedendo operazioni di ricerca, analisi o interrogazione basate sul contesto del documento. Inoltre, non si riesce a differenziare i vari elementi presenti, limitando gravemente la possibilità di recuperare informazioni specifiche e aumentando il carico di lavoro manuale. Durante il progetto, ho avuto l'opportunità di studiare diverse librerie per il parsing dei documenti, concentrandomi in particolare sui PDF. Attraverso vari test condotti utilizzando metriche di valutazione come Faithfulness, Correctness e Relevancy, non sono emerse differenze sostanziali nelle prestazioni delle librerie. Tuttavia, sono state osservate alcune differenze a livello di resa grafica nel parsing dei documenti, che variano a seconda della libreria utilizzata. Alla fine, ho scelto di utilizzare PyPDF2, perché a livello grafico era più comprensibile.

## 7 Codice

In questa sezione verranno analizzate le porzioni di codice più rilevanti del prototipo.

### 7.1 Inizializzazione e Parsing degli Argomenti

La funzione *main* inizializza il parsing degli argomenti permettendo agli utenti di resettare il database all'avvio del prototipo. Successivamente carica i documenti, li divide in chunk, li inserisce all'interno del database vettoriale e avvia la chat.

```
def main():
    # Creazione del parser degli argomenti
    parser = argparse.ArgumentParser()

    # Aggiunta dell'argomento per resettare il database
    parser.add_argument("--reset", action="store_true", help="Reset the database.")

    # Parsing degli argomenti dalla riga di comando
    args = parser.parse_args()

    # Controllo sulla richiesta di reset del database
    if args.reset:
        print("💧 Clearing Database")
        clear_database()

    # Caricamento dei documenti, suddivisione ed aggiunta al database vettoriale
    documents = load_documents()
    chunks = split_documents(documents)
    add_to_chroma(chunks)

    # Inizializzazione della chat
    chat()
```

Figure 1: Main



## 7.2 Query RAG

La funzione `query_rag` prende in input due parametri chiamati `query_text` e `context` ed inizia preparando il database ottenendo una funzione di embedding ed inizializzando un oggetto db con la classe Chroma. Successivamente, esplora il database alla ricerca di voci simili al `query_text` restituendo i primi cinque risultati con i rispettivi punteggi di similarità. Aggiunge poi il contesto al prompt combinando il contesto dato con i contenuti delle pagine dei documenti recuperati ed invia questo al modello Llama3 per ottenere una risposta. Infine, formattando le fonti estraendo gli ID dei documenti dai metadati dei risultati e crea una stringa di risposta che include il testo della risposta e le fonti, che viene poi stampata e restituita.

```
#funzione query
def query_rag(query_text: str, context: str):
    # Prepara il DB.
    embedding_function = get_embedding_function()
    db = Chroma(persist_directory=CHROMA_PATH, embedding_function=embedding_function)

    # Esplora DB.
    results = db.similarity_search_with_score(query_text, k=5)

    # Aggiungi il contesto al prompt
    context_text = context + "\n\n" + "\n\n---\n\n".join([doc.page_content for doc, _score in results])
    prompt_template = ChatPromptTemplate.from_template(PROMPT_TEMPLATE)
    prompt = prompt_template.format(context=context_text, question=query_text)

    # Esegui la query con il modello desiderato e la propria API KEY di Groq
    model = ChatGroq(model="llama3-70b-8192", api_key = API_KEY)
    response_text = model.invoke(prompt)

    # Fonti
    sources = [doc.metadata.get("id", None) for doc, _score in results]
    formatted_response = f"Response👉: {response_text}\nSources👉: {sources}\n\n"
    print(formatted_response)

    # Restituisci la risposta del modello
    return response_text
```

Figure 2: `Query_rag`

### 7.3 Preprocessing dei documenti

Questo frammento di codice fa parte della funzione `load_documents` che si occupa di preelaborare i documenti di formato PDF, DOCX, PPTX presenti in una directory specificata e di caricarli nel database. In particolare, questo frammento si occupa di gestire i documenti PDF dall'estrazione del contenuto testuale tramite il metodo `PdfReader` all'estrazione delle tabelle con `tabula` e alla loro rielaborazione in linguaggio naturale.

```
if file_extension == '.pdf':
    # Carica file PDF e estrai testo
    with open(file_path, 'rb') as f:
        reader = PdfReader(f)
        text += "\n".join([page.extract_text().strip() for page in reader.pages])

    # Utilizza tabula-py per estrarre le tabelle
    tables = tabula.read_pdf(file_path, pages='all')

    # Converti ogni tabella in linguaggio naturale
    for table in tables:
        prompt = "transform the following table into natural language including in every row transformed" \
            "all the informations about what that value rappresents:" + table.to_string(index=False)
        model = ChatGroq(model="llama3-70b-8192", api_key=API_KEY)
        chain = model | StrOutputParser()
        response_text = chain.invoke(prompt)
        text += f"{response_text}\n"
```

Figure 3: PDF

## 7.4 Splitting dei documenti

La funzione *split\_documents* prende come parametro una lista di documenti. Nel corpo della funzione viene creato un oggetto *text\_splitter* utilizzando la classe *RecursiveCharacterTextSplitter* con i parametri specificati. La funzione restituisce il risultato della chiamata al metodo *split\_documents* dell'oggetto *text\_splitter* che divide i documenti forniti in frammenti secondo i parametri specificati.

```
#funzione per splittare i documenti
def split_documents(documents: list[Document]):

    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=800,
        chunk_overlap=80,
        length_function=len,
        is_separator_regex=False,
    )
    return text_splitter.split_documents(documents)
```

Figure 4: split\_documents

## 7.5 Gestione database vettoriale

La funzione *add\_to\_chroma* prende come parametro una lista di chunk di documenti come parametro. Si inizia creando un oggetto db utilizzando la classe Chroma e calcola poi gli ID dei chunk chiamando la funzione *calculate\_chunk\_ids*. La funzione continua recuperando gli ID degli elementi esistenti del database e li memorizza per verificare che i chunk non siano già presenti nel db. Se ci sono nuovi frammenti li aggiunge al database, altrimenti stampa un messaggio indicando che non ci siano nuovi documenti da aggiungere

```
#funzione per il database vettoriale
def add_to_chroma(chunks: list[Document]):
    db = Chroma(
        persist_directory=CHROMA_PATH, embedding_function=get_embedding_function()
    )

    #calcola gli IDs delle pagine
    chunks_with_ids = calculate_chunk_ids(chunks)

    #aggiunge o aggiorna i documenti
    existing_items = db.get(include=[]) #IDs sempre inclusi di default
    existing_ids = set(existing_items["ids"])
    print(f"Number of existing documents in DB: {len(existing_ids)}")

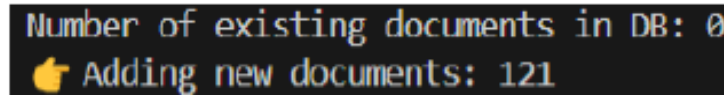
    #aggiunge i documenti che non esistono nel DB
    new_chunks = []
    for chunk in chunks_with_ids:
        if chunk.metadata["id"] not in existing_ids:
            new_chunks.append(chunk)

    if len(new_chunks):
        print(f"👉 Adding new documents: {len(new_chunks)}")
        new_chunk_ids = [chunk.metadata["id"] for chunk in new_chunks]
        db.add_documents(new_chunks, ids=new_chunk_ids)
        db.persist()
    else:
        print("✅ No new documents to add\n")
```

Figure 5: add\_to\_chroma

## 8 Esempio di esecuzione completa

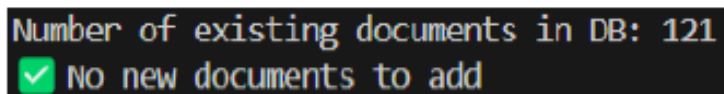
Dopo che l'utente ha inserito i documenti nella cartella "data" dell'applicazione, all'avvio dovrà attendere qualche secondo in quanto i nuovi documenti inseriti devono essere aggiunti al database vettoriale.

A terminal window with a black background and yellow text. The first line reads "Number of existing documents in DB: 0". The second line shows a yellow hand icon followed by "Adding new documents: 121".

```
Number of existing documents in DB: 0
👉 Adding new documents: 121
```

Figure 6: nuovi documenti da aggiungere al db

Nel caso in cui l'utente riavvia l'applicazione e non aggiunge nuovi documenti a quelli già presenti non dovrà attendere.

A terminal window with a black background and yellow text. The first line reads "Number of existing documents in DB: 121". The second line shows a green checkmark icon followed by "No new documents to add".

```
Number of existing documents in DB: 121
✅ No new documents to add
```

Figure 7: nessun documento da aggiungere

Dopo aver aggiunto i documenti al database andremo adesso ad analizzare come si comporta il sistema rispetto alle domande dell'utente da informazioni contenute nei documenti. Uno dei casi che andremo ad analizzare è la richiesta di spiegazioni su un gruppo di regole del Monopoly chiamate *speed die rules* che nel documento sono spiegate come mostra la figura 8.

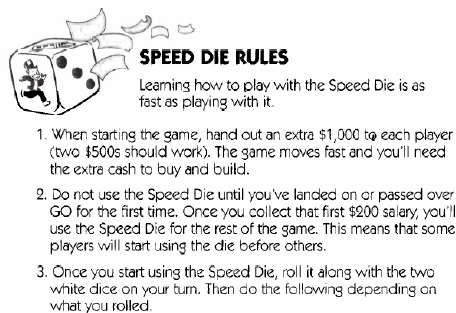


Figure 8: Speed die rules

La figura 9 illustra la risposta del sistema che è composta di due parti:

- La parte esplicativa contenete le informazioni recuperate dal documento e riorganizzate da Llama3.
- La parte Sources che contiene i riferimenti alle parti del documento delle quali sono state recuperate le informazioni.

```
You 🐼 tell me about speed die rules.
Response 🦙: According to the provided context, here are the Speed Die rules:

1. When starting the game, hand out an extra $1,000 to each player (two $500s should work). The game moves fast and you'll need the extra cash to buy and build.
2. Do not use the Speed Die until you've landed on or passed over GO for the first time, once you collect that first $200 salary, you'll use the Speed Die for the rest of the game. This means that some players will start using the die before others.
3. Once you start using the Speed Die, roll it along with the two white dice on your turn. Then do the following depending on what you rolled:

    * 1, 2, or 3: Add this number to the roll of the two white dice. You'll zoom around the board.

For a roll of Bus: This lets you "get off the bus early." Look at the two white dice. You can move the value of one die, the other die, or the sum of both dice.
sources: ['C:\\Users\\genna\\Desktop\\tirocinio\\progetti\\PRAG\\data\\monopoly.pdf:None:24', 'C:\\Users\\genna\\Desktop\\tirocinio\\progetti\\PRAG\\data\\monopoly.pdf:None:2', 'C:\\Users\\genna\\Desktop\\tirocinio\\progetti\\PRAG\\data\\monopoly.pdf:None:8', 'C:\\Users\\genna\\Desktop\\tirocinio\\progetti\\PRAG\\data\\monopoly.pdf:None:1', 'C:\\Users\\genna\\Desktop\\tirocinio\\progetti\\PRAG\\data\\monopoly.pdf:None:23']
```

Figure 9: Query sulla speed die rule

Il prossimo documento di esempio è utile al fine di mostrare come si comporta il sistema rispetto ad una domanda su informazioni contenute in una tabella.

Global warming is a phenomenon primarily caused by the increase in concentrations of greenhouse gases in the atmosphere. Greenhouse gases such as carbon dioxide (CO<sub>2</sub>), methane (CH<sub>4</sub>), and nitrous oxide (N<sub>2</sub>O) trap the sun's heat in the atmosphere and contribute to the rise in global average temperatures. This phenomenon is primarily caused by human activities such as the burning of fossil fuels (coal, oil, and gas), deforestation, and agricultural activities.

The consequences of global warming are numerous and include rising sea levels, melting polar ice, changes in precipitation and temperature patterns, extreme weather events such as floods and droughts, and a negative impact on biodiversity and crops.

Here is a table with the countries that pollute the most and their respective percentages of CO<sub>2</sub> emissions relative to the global total:

Country	CO <sub>2</sub> Emissions (Million Metric Tons)	Percentage of Total Emissions
China	10065.0	41.63%
United States	5417.2	22.41%
India	2611.0	10.80%
Russia	1674.2	6.93%
Japan	1162.7	4.81%
Germany	759.6	3.14%
Iran	720.0	2.98%
South Korea	611.0	2.53%
Saudi Arabia	582.4	2.41%
Canada	571.6	2.36%

These countries are responsible for a significant portion of global CO<sub>2</sub> emissions, but it's important to note that all countries have a role to play in combating climate change and reducing greenhouse gas emissions.

Figure 10: Documento sulle emissioni di CO2

Come illustra la prossima figura recupera tutte le informazioni contenute nella tabella del documento e le riformula per rispondere alla domanda dell'utente.

```
You 🧑: Which countries are most responsible for carbon dioxide emissions?
Response 🤖: According to the table, the top four countries in terms of carbon dioxide (CO2) emissions are:

1. China - approximately 41.63% of global total CO2 emissions (10065.0 million metric tons)
2. United States - approximately 22.41% of global total CO2 emissions (5417.2 million metric tons)
3. India - approximately 10.80% of global total CO2 emissions (2611.0 million metric tons)
4. Russia - approximately 6.93% of global total CO2 emissions (1674.2 million metric tons)

These countries are responsible for a significant portion of global CO2 emissions, but it's important to note that a
ll countries have a role to play in combating climate change and reducing greenhouse gas emissions.
Sources 📄: ['C:\\Users\\genna\\Desktop\\tirocinio\\progetti\\PRAG\\data\\GlobalEmissions.docx:None:1', 'C:\\Users\\genna\\Desktop\\tirocinio\\progetti\\PRAG\\data\\GlobalEmissions.docx:None:2', 'C:\\Users\\genna\\Desktop\\tirocinio\\progetti\\PRAG\\data\\GlobalEmissions.docx:None:0', 'C:\\Users\\genna\\Desktop\\tirocinio\\progetti\\PRAG\\data\\GlobalEmissions.docx:None:3']
```

Figure 11: Query sui paesi responsabili per le emissioni

## 9 Conclusioni

Il lavoro svolto ha portato alla realizzazione di un sistema di analisi documentale orientato alla RAG che dimostra come l'integrazione di LLM e tecniche di information retrieval possa notevolmente migliorare l'efficienza e l'accuratezza nell'estrazione e nell'elaborazione delle informazioni da documenti complessi. Il prototipo è stato testato con una varietà di documenti contenenti testo, immagini, tabelle e risultato dei test hanno evidenziato una buona performance del sistema nel rispondere alle domande dell'utente, mantenendo la coerenza e la rilevanza nelle risposte generate.

## 10 Glossario

Termini specifici utilizzati:

- **Database:** archivio centralizzato di dati persistenti, accessibile per il recupero e la modifica delle informazioni salvate.
- **Chunk:** sono porzioni di dati più grandi suddivisi in blocchi più piccoli per semplificarne l'elaborazione, la trasmissione o la gestione. Nel contesto degli NLP, il testo lungo viene suddiviso in "chunk" (frasi o paragrafi) per renderlo gestibile da modelli che hanno limiti di lunghezza di input.
- **Embedding:** sono rappresentazioni numeriche di dati, spesso usati in intelligenza artificiale e machine learning per convertire informazioni complesse in vettori di numeri reali che possono essere gestiti dai modelli di apprendimento automatico.
- **Dataset:** insieme di dati su cui un algoritmo opera durante l'analisi o l'addestramento.
- **Algoritmo:** soluzione struttura a un problema, comporta da operazioni e condizioni ben definite.