ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

ParlaMentis: Agente Conversazionale Retrieval-Enhanced per la Camera dei Deputati

Tesi Triennale in
Programmazione Di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Co-relatore
PhD. Giacomo Frisoni
Dr. Lorenzo Molfetta

Candidato
Giorgio Fantilli

Prima Sessione
Anno Accademico 2023 – 2024

PAROLE CHIAVE

Large Language Model

Retrieve and Generate

Knowledge-Enhanced Question Answering

Multi-Document Reasoning and Acting

Italian Legal Language Processing

This is who I am, Can't be anyone else, so...

Abstract

Il diritto svolge un ruolo centrale e fondamentale nel plasmare la nostra società, influenzando ed insinuandosi in ogni aspetto della vita civile. In questo contesto, le risorse giuridiche fonte di diritto, sia testuali che multimediali, sono ubique, in continua espansione, di ardua interpretazione e intrise di riferimenti incrociati. I moderni sistemi di elaborazione del linguaggio naturale presentano il potenziale per ottimizzare l'attività normativa, permettendo ai legislatori di semplificare la ricerca, comprensione e generazione di documenti, e allo stesso tempo promuovere una maggiore trasparenza verso i cittadini. Strumenti come i Large Language Model sono infatti estremamente efficaci per la consultazione automatica di vasti corpora di documenti legali, possedendo la capacità di estrapolazione selettiva e di correlazione delle informazioni impensabili per un utente umano. A tale scopo proponiamo ParlaMentis, un agente conversazionale multi-lingua e multi-task basato su una innovativa architettura Multi-Document Agent, che combina la potenza dei Large Language Model, tecniche di information retrieval, ed il paradigma Reasoning and Acting. Questi strumenti consentono la costruzione di un modello che è in grado di soddisfare domande riguardanti un vasto corpus di atti di iniziativa della Camera dei Deputati italiana e di video-interventi avvenuti in sedute dell'Assemblea. ParlaMentis fa uso di un Large Language Model appositamente addestrato sul contesto giuridico di riferimento, capace di ragionamento e rielaborazione delle richieste utente, fino alla predisposizione e successiva esecuzione di un dettagliato piano d'azione dinamico volto al selettivo ed efficiente recupero delle nozioni necessarie per generare una risposta quanto più possibile fattuale.

Introduzione

L'elaborazione e la gestione delle informazioni nel contesto legale è una sfida particolarmente ardua e complessa. Le risorse giuridiche sono vastissime e in continua espansione, rendendo difficile per i legislatori accedere e interpretare i dati necessari in modo efficiente. I documenti legali sono spesso intrisi di riferimenti incrociati e richiedono una comprensione approfondita per poter essere utilizzati correttamente. Questo contesto rende essenziale l'adozione di strumenti avanzati che possano supportare il lavoro legislativo in modo efficace.

Per rispondere a questa esigenza, presentiamo ParlaMentis, un agente conversazionale multi-lingua allo stato dell'arte in grado di gestire vasti insiemi di dati strutturati e di soddisfare complessi compiti di accesso ed elaborazione delle informazioni. ParlaMentis, dal latino "per il Parlamento", è stato progettato per migliorare l'efficienza e l'efficacia del lavoro dei deputati italiani nella predisposizione degli atti di iniziativa, quali proposte legislative, emendamenti, mozioni e interrogazioni. Questo strumento fornisce informazioni aggiornate sulla normativa esistente, offre sintesi e comprensione guidata dei documenti prodotti dalla Camera dei Deputati, suggerisce formulazioni per testi legislativi e garantisce l'accesso semantico a fonti multi-modali, inclusi video e trascrizioni degli interventi parlamentari.

La tecnologia alla base di ParlaMentis è una innovativa architettura Multi-Document Agent che combina la potenza di un Large Language Model, specificamente addestrato su di un'ampia gamma di testi legali italiani, con tecniche avanzate di information retrieval ed il paradigma Reasoning and Acting. Il risultato è uno strumento che sfrutta le capacità di agenti ReAct specializzati e gerarchizzati che collaborano tra loro per rispondere a domande complesse e che richiedono l'accesso simultaneo a più fonti informative diverse.

Il sistema è in grado di integrare sinergicamente fasi di ragionamento e azione: elabora un piano d'azione dettagliato basato sulle informazioni disponibili e interagisce con varie risorse per recuperare materiali utili alla generazione di risposte informative e accurate. Il sistema tiene quindi memoria del contesto informativo, esegue una ricerca pianificata delle fonti su un database dedicato, estrae e interpreta i dati pertinenti, e adatta il suo futuro comportamento per completare la conoscenza acquisita. Questo ciclo di ragionamento e di conseguente azione si può ripetere più e più volte, finché il modello non considera la conoscenza recuperata sufficiente per soddisfare le richieste.

Nei prossimi capitoli procediamo con la descrizione approfondita del sistema e del lavoro svolto:

- **Capitolo 1 - Fondamenti Teorici**: introduzione al concetto di NLP e descrizione delle tecnologie utilizzate come i Large Language Model e le tecniche di information retrieval.
- **Capitolo 2 - Lavori Correlati**: descrizione del paradigma ReAct che sta alla base della nostra architettura e panoramica sulla situazione attuale nell'ambito degli LLM specifici per la lingua italiana.
- **Capitolo 3 - Metodo**: descrizione del sistema implementato, con approfondimenti sull'architettura di base Multi-Document Agent, sul lavoro di costruzione dei dataset utilizzati per il retrieval e per l'addestramento del modello, e sul processo di allenamento dello stesso.
- **Capitolo 4 - Metriche e valutazione del Modello**: descrizione dell'approccio di valutazione del sistema e delle metriche utilizzate, studio ablativo e commento dei risultati ottenuti.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Fondamenti Teorici

1.1 Natural Language Processing

Il Natural Language Processing (NLP) è un campo interdisciplinare che combina informatica, intelligenza artificiale e linguistica computazionale. Il suo scopo principale è consentire alle macchine di comprendere, interpretare e generare il linguaggio umano in modo significativo e utile. La capacità delle macchine di interagire con il linguaggio umano apre nuove opportunità, sia nel settore delle tecnologie consumer, come gli assistenti vocali, sia in ambiti più specialistici, come l'analisi automatica di grandi volumi di dati testuali per ricavarne informazioni significative e facilmente interpretabili da un utente umano.

La Text Generation rappresenta una delle sfide più affascinanti ma decisive nell'ambito del NLP. Si tratta della capacità dei sistemi di intelligenza artificiale di generare testo in linguaggio naturale, coerente e rilevante rispetto a un contesto specifico o a un input fornito. Questo compito richiede non solo la comprensione e la modellazione del linguaggio, ma anche la capacità di produrre output che sembri naturale e sensato agli esseri umani.

I primi approcci alla Text Generation si basavano su modelli statistici, che facevano affidamento su regole linguistiche predefinite e dati di addestramento limitati. Questi metodi, sebbene innovativi per l'epoca, presentavano diverse limitazioni: mancavano di flessibilità nel catturare le dipendenze a lungo termine

nel linguagguio e generavano testo che spesso risultava meccanico e incoerente. Con l'avvento delle Recurrent Neural Network (RNN) e delle Long Short-Term Memory (LSTM), la Text Generation ha compiuto un notevole passo avanti. Le RNN sono progettate per gestire dati ordinati mantenendo uno stato interno, che consente loro di considerare la loro dipendenza sequenziale. A differenza delle reti neurali tradizionali, le RNN elaborano un elemento alla volta, aggiornando il loro hidden state in base all'input corrente e agli stati precedenti. Questa architettura le rende particolarmente efficaci nel catturare le dipendenze a breve termine. Tuttavia, possono soffrire di un problema chiamato vanishing gradient, un fenomeno che limita la loro capacità di apprendere invece le dipendenze a lungo termine. Questo fenomeno si verifica durante il backpropagation, quando i gradienti delle funzioni di loss tendono a diventare esponenzialmente piccoli, impedendo al modello di apprendere e immagazzinare conoscenza efficacemente. Per affrontare questo problema, sono state introdotte le LSTM, una variante delle normali RNN progettata specificamente per la corretta gestione delle dipendenze a lungo termine. Le LSTM utilizzano celle di memoria e meccanismi di gating per controllare il flusso di informazioni. Le componenti principali di una cella sono il gate di input, il gate di output e il gate di forget, che lavorano insieme per mantenere e aggiornare le informazioni rilevanti nel tempo. Questo approccio consente alle LSTM di mantenere informazioni per sequenze molto lunghe, superando così le limitazioni delle RNN tradizionali.
Infine, le RNN e le LSTM sono spesso strutturate in modelli sequence-to-sequence, una delle architetture più efficaci per la Text Generation. Un modello sequence-to-sequence è composto da due parti principali: l'encoder e il decoder. L'encoder processa la sequenza di input e la trasforma in una rappresentazione di lunghezza fissa chiamata context vector. Questo viene poi passato al decoder, che genera la sequenza di output basandosi sulle informazioni in esso contenute.

Detto questo, l'approccio di base sequenziale delle RNN e delle LSTM presenta ancora delle sfide, in particolare per quanto riguarda la parallelizzazione durante l'addestramento. Le limitazioni di memoria e le difficoltà nel batching rendono difficile addestrare efficacemente modelli su lunghe sequenze di dati.

1.1.1 Il Meccanismo di Attention

L'Attention è un meccanismo avanzato introdotto per migliorare le prestazioni delle architetture di reti neurali, in particolare nelle architetture encoder-decoder utilizzate per traduzione automatica e per altri compiti di NLP. Questo consente al modello di concentrarsi sulle parti più rilevanti dell'input durante la fase di generazione dell'output, superando le limitazioni delle RNN e delle LSTM, che faticano a gestire sequenze lunghe a causa della loro natura sequenziale.
L'Attention funziona mappando una query e un insieme di coppie chiave-valore a un output, dove query, chiavi, valori e output sono tutti vettori. L'output è calcolato come una somma pesata dei valori, con i pesi che vengono determinati da una funzione di compatibilità tra la query e le chiavi. La forma più comune di attenzione è la Scaled Dot-Product Attention, espressa dalla formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

dove Q rappresenta le query, K le chiavi, V i valori, e d_k è la dimensione delle chiavi. Questo meccanismo permette al modello di valutare l'importanza di ogni elemento della sequenza di input rispetto alla query.
Nell'architettura encoder-decoder originaria, l'encoder codifica l'intera sequenza di input in un singolo context vector, che il decoder utilizza per generare la sequenza di output. Tuttavia, questa rappresentazione fissa diventa inefficace nel mantenere informazioni rilevanti anche su sequenze lunghe. L'Attention risolve questo problema generando un nuovo context vector per ogni passo del decoder, calcolato come una somma pesata delle annotazioni dell'encoder. Ogni annotazione è una rappresentazione della sequenza di input a un determinato passo temporale, prodotta da una rete neurale ricorrente bidirezionale (BiRNN). Le annotazioni forward e backward vengono concatenate per ogni parola di input. Il context vector per ogni parola di output y_i è calcolato come:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

dove h_j è l'annotazione per la parola x_j dell'input e i pesi α_{ij} sono calcolati come:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

Il punteggio di attention e_{ij} è calcolato tramite una funzione di attenzione α:

$$e_{ij} = a(s_{i-1}, h_j)$$

dove s_{i-1} è l'hidden state del decoder al passo precedente.
Questo approccio permette al modello di dare maggiore peso alle parole di input rilevanti per la generazione di ciascuna parola di output, migliorando significativamente la capacità di catturare e rappresentare dipendenze a lungo termine all'interno della sequenza di input. Inoltre, l'Attention consente un maggiore parallelismo nel processamento delle sequenze, poiché il calcolo dei pesi di attention può essere eseguito in parallelo, contrariamente a quanto possibile nelle RNN sequenziali. Questo aumenta notevolmente l'efficienza e la velocità del modello, rendendolo adatto per l'addestramento su grandi dataset e per l'applicazione in tempo reale su compiti di NLP complessi.

1.1.2 Transformer

Il salto significativo nel campo dell'NLP è stato reso possibile dai modelli Transformers, introdotti da Vaswani et al. nel 2017 [3]. Il Transformer è un'architettura rivoluzionaria che utilizza esclusivamente meccanismi di attention, eliminando completamente la ricorrenza e dimostrando prestazioni superiori rispetto ai modelli basati su RNN. L'architettura del Transformer mantiene la struttura encoder-decoder, in cui l'encoder codifica una sequenza di simboli di input in una sequenza di rappresentazioni continue, mentre il decoder utilizza queste rappresentazioni per generare la sequenza di output. La Figura 1.1 mostra l'architettura nella sua interezza.

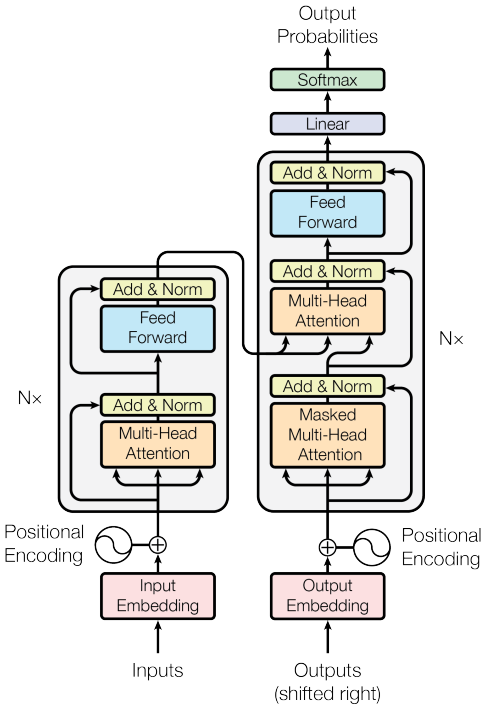

Figura 1.1: Architettura encoder-decoder del modello Transformer.
Fonte: 3, Vaswani et al.

L'encoder è composto da un insieme di N layer identici, ciascuno dei quali ha due sotto-layer principali: un meccanismo di Multi-Head Self-Attention e una rete feed-forward completamente connessa. Ogni sotto-layer è seguito da una normalizzazione dei layer e utilizza connessioni residuali, ovvero l'output di ciascun sotto-layer è LayerNorm(x + Sublayer(x)), dove Sublayer(x) è la funzione implementata dal sotto-layer stesso. Questo design consente una propagazione del gradiente più efficace e migliora la stabilità dell'addestramento.

Una delle componenti chiave è il meccanismo di Self-Attention, il quale consente il calcolo delle rappresentazioni delle sequenze in input considerando globalmente tutte le posizioni della sequenza stessa. Permette al modello di pesare l'importanza di ogni token rispetto agli altri, migliorando così la capacità di catturare dipendenze a lungo termine e di rappresentare informazioni contestuali in maniera più efficiente rispetto alle RNN. La Self-Attention è descritta dalla formula:

$$\text{Self-Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

dove Q (query), K (key), V (value) provengono dalla stessa sequenza. Le query, le chiavi e i valori sono ottenuti proiettando gli input originali tramite matrici di peso apposite. Con questo strumento è possibile calcolare un coefficiente di attenzione ponderato per ogni posizione nella sequenza, facilitando l'apprendimento delle relazioni tra i token senza considerare la loro distanza.

Un altro elemento fondamentale del Transformer è la Multi-Head Attention, che migliora la capacità del modello di catturare diversi tipi di relazioni nella sequenza di input eseguendo in parallelo più meccanismi di attenzione su proiezioni lineari delle query, chiavi e valori. La formula per la Multi-Head Attention è la seguente:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, \ldots, head_h)W^O$$

dove $head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ e W^O è una matrice di proiezione lineare.

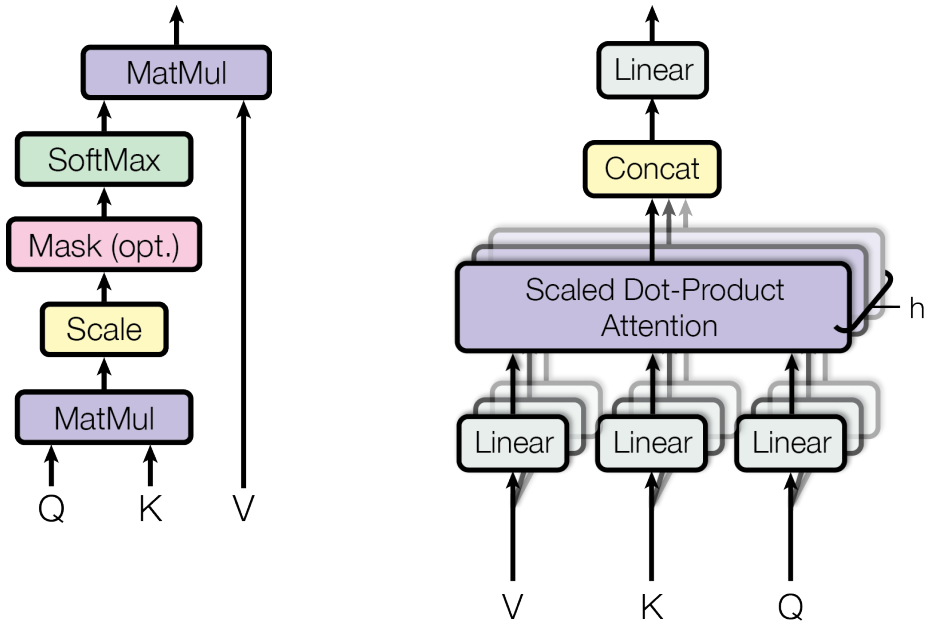

Figura 1.2: La Scaled Dot-Product Attention (a destra) e la Multi-Head Attention (a sinistra).

Fonte: 3, Vaswani et al.

Il meccanismo della Multi-Head Attention consente al modello di focalizzarsi su diversi aspetti dell'input simultaneamente, esplorando diverse relazioni e interazioni tra i token su differenti sottospazi, andando così a migliorare la rappresentazione delle dipendenze complesse e quindi l'effettiva capacità del modello di comprendere e generare linguaggio naturale. La Figura 1.2 illustra graficamente la Multi-Head Attention ed il suo funzionamento interno.

Un'altra innovazione critica dei Transformer è l'uso del Positional Encoding per mantenere l'informazione sull'ordine dei token nella sequenza in input, poiché l'architettura del Transformer non ha meccanismi intrinseci per gestire la sequenzialità come le RNN. Le Positional Encoding vengono rappresentate come vettori da aggiungere agli embedding di input e output, in modo tale da incorporare anche le informazioni sulla posizione relativa o assoluta dei token. Le Positional Encoding (PE) sono definite come:

$$\text{PE}(pos, 2i) = sin(\frac{pos}{10000^{2i/d_{model}}})$$
$$\text{PE}(pos, 2i+1) = cos(\frac{pos}{10000^{2i/d_{model}}})$$

dove pos è la posizione del token nella sequenza e d_{model} è la dimensione del modello. Questi vettori sinusoidali permettono al modello di apprendere relazioni posizionali, migliorando la sua capacità di processare e generare sequenze di lunghezze variabili.

Infine, come già anticipato, l'architettura del Transformer consente una maggiore parallelizzazione rispetto ai modelli ricorrenti, riducendo significativamente il tempo di addestramento e migliorando la scalabilità. L'eliminazione della ricorrenza permette infatti di sfruttare appieno le capacità di calcolo parallelo delle moderne GPU, rendendo possibile l'addestramento su dataset molto grandi e accelerando l'inferenza.

Grazie a queste innovazioni, i Transformer hanno stabilito un nuovo stato dell'arte in molti compiti di NLP, dimostrando al contempo una maggiore efficienza computazionale e flessibilità nell'apprendimento di complessi pattern linguistici.

1.2 Large Language Model

I Large Language Model (LLM) [4] rappresentano una pietra miliare nell'ambito del Natural Language Processing, basandosi su architetture neurali avanzate come i Transformer. Questi modelli si distinguono per la loro capacità di elaborare e generare testi linguistici con una profondità e una precisione sorprendenti, frutto di un intenso processo di pre-addestramento su dataset di

dimensioni enormi. Il termine "large" si riferisce infatti alla vastità del dataset di addestramento, spesso comprendente petabyte di dati, ma anche al numero di parametri che caratterizzano la rete neurale. Ad esempio, modelli come Generative Pre-trained Transformer (GPT) possono contenere centinaia di milioni, se non miliardi, di parametri, che consentono loro di catturare dettagli complessi e sottili delle strutture linguistiche.

Il funzionamento di un LLM si basa su di un intenso periodo di pre-addestramento su testi generici, come libri o articoli, grazie al quale acquisisce una comprensione profonda delle regolarità linguistiche e semantiche del linguaggio naturale. Durante questa fase, il modello viene allenato a predire la parola successiva in una sequenza di testo. Questo processo è supportato da tecniche auto-supervised, che permettono all'LLM di apprendere senza la necessità di etichettature manualima utilizzando il contesto circostante. I Transformer, in particolare, sono stati rivoluzionari in questo senso, permettendo di superare le limitazioni dei modelli precedenti come le reti neurali ricorrenti nella gestione del contesto a lungo termine e nella scalabilità computazionale.

Una volta pre-addestrato, un LLM può essere adattato a compiti specifici di NLP mediante il fine-tuning, venendo ulteriormente addestrato su dataset più piccoli ma appositamente strutturati per il task desiderato. Questa fase utilizza tecniche di apprendimento supervisionato, dove il modello viene guidato da esempi etichettati per apprendere a eseguire compiti precisi con maggiore accuratezza. Ad esempio, per il compito di classificazione del testo, il modello viene addestrato con coppie di testo e relative etichette di categoria.
Un altro approccio emergente è il in-context learning, dove il modello, anziché essere riaddestrato, viene fornito con esempi specifici durante l'esecuzione, utilizzando il contesto fornito dall'utente per adattare la risposta. Questo metodo sfrutta la capacità degli LLM di comprendere e generalizzare dai prompt forniti, rendendo possibile l'adattamento dinamico a una vasta gamma di compiti senza la necessità di un addestramento ulteriore.
Questo addestramento permette agli LLM di eccellere in una vasta gamma di compiti, tra cui classificazione del testo, risposta alle domande, riassunto automatico e generazione di testo. La chiave per ottimizzare le prestazioni di questi modelli risiede nella progettazione di prompt efficaci, un processo noto

con il nome di Prompt Engineering, che determina la qualità e la pertinenza delle risposte generate.

In conclusione, i Large Language Model rappresentano una combinazione avanzata di tecnologie neurali e approcci di apprendimento automatico che hanno rivoluzionato il campo del NLP. Grazie alla loro capacità di apprendere e generalizzare da enormi quantità di dati testuali, questi modelli non solo migliorano la precisione e l'efficienza delle applicazioni linguistiche automatizzate, ma aprono anche nuove frontiere nella comprensione e nell'interazione con il linguaggio naturale. La continua evoluzione degli LLM promette di ampliarne ulteriormente le capacità e le applicazioni, rendendoli strumenti sempre più potenti e versatili nel campo dell'intelligenza artificiale.

1.3 Retrieval-Augmented Generation

Il Retrieval-Augmented Generation (RAG) [5] è un'architettura avanzata di machine learning che combina la potenza dei Large Language Model con sofisticate tecniche di information retrieval. Questa integrazione è progettata per superare le limitazioni degli LLM tradizionali, i quali, sebbene estremamente potenti, possono generare risposte imprecise o non supportate da fatti reali, un fenomeno noto come "allucinazione".
La struttura del RAG può essere scomposta in diverse fasi critiche (visibili anche nella Figura 1.3: indicizzazione, recupero, integrazione e generazione, ognuna delle quali svolge un ruolo fondamentale nel processo complessivo.

- **Indicizzazione**: Questa fase inizia con la preparazione del corpus di documenti che formeranno la base di conoscenza dalla quale attingere. I documenti vengono suddivisi in unità più piccole, come paragrafi o frasi, che vengono poi convertite in rappresentazioni vettoriali utilizzando modelli di embedding, come ad esempio Bidirectional Encoder Representations from Transformers (BERT). Questi vettori rappresentano i contenuti semantici dei frammenti di testo e sono memorizzati in un database vettoriale, spesso implementato con tecnologie come Facebook AI Similarity Search (FAISS), che supportano ricerche di similarità ad

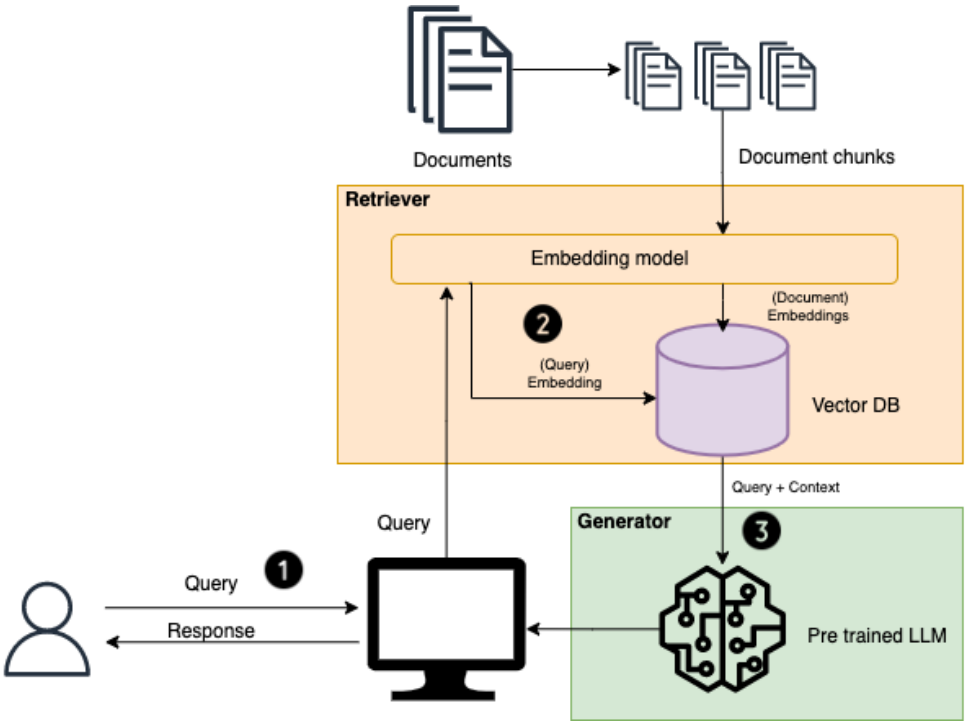

Figura 1.3: Le varie fasi in cui si struttura il Retrieval-Augmented Generation.
Fonte: The ELI5 Guide to Retrieval Augmented Generation

alta velocità. Questo passaggio è cruciale per garantire che i dati possano essere recuperati in modo efficiente e accurato in risposta a query future.

- **Recupero**: Quando una query viene presentata al sistema RAG, essa viene prima trasformata in una rappresentazione vettoriale utilizzando lo stesso modello di embedding adottato durante l'indicizzazione. Successivamente, il sistema calcola la similarità tra il vettore della query e i vettori dei frammenti indicizzati. Le metriche di similarità comunemente utilizzate includono la distanza coseno e la distanza euclidea. I frammenti con i punteggi di similarità più alti vengono recuperati e preparati per essere utilizzati nella fase successiva. Questa fase è cruciale perché la qualità dei frammenti recuperati determina in larga misura la pertinenza e l'accuratezza della risposta finale generata.

- **Integrazione e Generazione**: I frammenti recuperati vengono combinati con la query originale per formare un prompt esteso che viene poi

fornito a un LLM. Questo prompt esteso permette al modello di generare una risposta che non solo riflette la conoscenza pre-esistente nel modello, ma che soprattutto viene arricchita con le informazioni inerenti recuperate. Il processo di integrazione può essere ulteriormente ottimizzato attraverso tecniche come la ponderazione dei frammenti in base alla loro rilevanza o l'utilizzo di approcci di ensemble per combinare le risposte generate da diversi frammenti.

Versioni più avanzate di architetture RAG implementano ottimizzazioni in tutte le fasi del processo. Durante la fase di indicizzazione, possono essere utilizzate tecniche di segmentazione più sofisticate che tengono conto del contesto e della struttura dei documenti. Inoltre, possono essere incorporati metadati aggiuntivi per migliorare la qualità del reperimento di informazioni. Le query possono inoltre essere arricchite o espanse utilizzando informazioni contestuali o conoscenza di dominio per migliorare la precisione della fase di recupero. Infine, nella fase di generazione, possono essere utilizzate tecniche di regolarizzazione per evitare la produzione di contenuti ridondanti o non pertinenti, migliorando così la coerenza e la qualità della risposta finale.

In conclusione, il Retrieval-Augmented Generation rappresenta un passo avanti significativo nella generazione di testo assistita dall'intelligenza artificiale, combinando il meglio degli LLM e delle tecniche di recupero dell'informazione per fornire risposte più accurate, pertinenti e contestualizzate. Il RAG trova applicazione in una vasta gamma di scenari, tra cui sistemi di assistenza virtuale, motori di ricerca avanzati, e applicazioni di business intelligence. Ad esempio, nei sistemi di assistenza virtuale, il RAG può fornire risposte precise e contestualizzate utilizzando sia la conoscenza preaddestrata del modello che informazioni aggiornate recuperate in tempo reale. Nei motori di ricerca, può migliorare la pertinenza dei risultati di ricerca combinando il recupero basato su similarità con la generazione di testo per rispondere in modo più preciso alle query degli utenti.

Nonostante tutto, l'architettura tuttavia presenta ancora diverse sfide. Una delle principali è la gestione delle informazioni da aggiornare in tempo reale, poiché i database vettoriali devono essere costantemente adeguati per riflettere le nuove informazioni. Un'altra sfida riguarda l'efficienza computazionale, poiché

il processo di indicizzazione e recupero richiede risorse significative. Infine la problematica più rilevante risiede l'integrazione di diverse fonti di dati, che richiede avanzate capacità di disambiguazione e fusione delle informazioni per evitare incoerenze e migliorare la precisione delle risposte generate. Le direzioni future della ricerca includono perciò lo sviluppo di modelli di embedding più efficienti, tecniche di aggiornamento incrementale per i database vettoriali e l'integrazione di approcci multimodali che combinano testo, immagini e altri tipi di dati per migliorare ulteriormente la qualità delle risposte generate.

Capitolo 2

Lavori Correlati

2.1 Reasoning and Acting

Il paradigma Reasoning and Acting (ReAct), introdotto per la prima volta da Yao et al. nel 2022 [6], rappresenta un approccio avanzato e innovativo nell'ambito dei Large Language Model, con particolare riferimento alla risoluzione di compiti complessi che richiedono capacità di ragionamento e azione integrata. Mentre i Transformer e i modelli di ragionamento a catena (Chain-of-Thought, CoT) hanno posto solide basi per la comprensione e la generazione del linguaggio naturale, ReAct eleva queste capacità integrando in modo sinergico ragionamento e azione.

ReAct si basa su di una caratteristica unica dell'intelligenza umana: la capacità di combinare senza soluzione di continuità azioni per eseguire compiti con il ragionamento verbale. Questo aspetto è considerato fondamentale nella cognizione umana, poiché permette l'auto-regolazione, la pianificazione strategica e il mantenimento della memoria di lavoro. Tra una specifica azione e l'altra, l'uomo è in grado di ragionare in linguaggio naturale per tenere traccia dei progressi, per gestire eccezioni o adattare il piano alla situazione, e per rendersi conto quando è necessaria un'informazione esterna. Questa stretta sinergia tra "agire" (acting) e "ragionare" (reasoning) permette agli esseri umani di imparare nuovi compiti rapidamente e prendere decisioni robuste, anche in circostanze precedentemente non viste o affrontando incertezze informative.

Allo stesso modo perciò un modello non dovrebbe solo essere in grado di generare testo coerente e rilevante, ma dovrebbe anche essere capace di interagire con risorse esterne, incorporando poi le informazioni acquisite attraverso queste "azioni" nel processo di ragionamento successivo. Nella pratica, ciò significa che il modello alterna fasi di ragionamento interno, dove elabora e pianifica la prossima azione basata sulle informazioni disponibili, e fasi di azione esterna, dove interagisce con varie fonti di dati, come database, API, altre risorse esterne o anche con altri agenti appositamente predisposti, al fine di raccogliere ulteriore materiale utile alla generazione di risposte quanto più informative possibili.

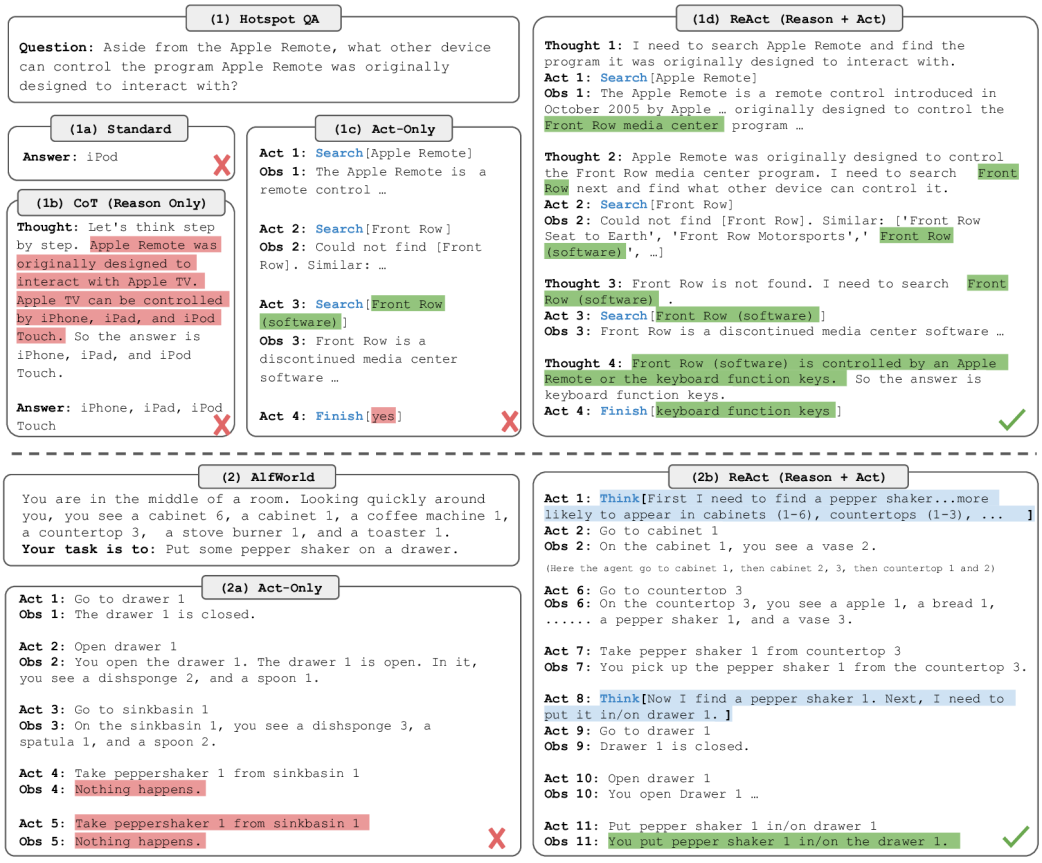

Figura 2.1: (1) Confronto di 4 metodi (a) Standard, (b) Chain-of-thought, (c) Solo azione, e (d) ReAct, nella risoluzione di task di multi-hop QA (Yang et al., 2018 [1]). (2) Confronto tra (a) Solo azione e (b) ReAct per risolvere il gioco AlfWorld (Shridhar et al., 2020b [2]).

Fonte: 6, Yao et al.

Si consideri una generica configurazione di un agente che interagisce con un ambiente per risolvere compiti. Al time-step t, esso riceve una *osservation* $o_t \in O$ dall'ambiente e compie un'azione $a_t \in A$ seguendo una politica $\phi(a_t|c_t)$, dove $c_t = (o_1, a_1, ..., o_t - 1, a_t - 1, o_t)$ è il contesto per l'agente. In questo caso, l'apprendimento di una politica risulta estremamente complesso quando la mappatura $c_t \rightarrow a_t$ è altamente implicita e richiede un'estesa computazione (Figura 2.1 (1c), (2a)).
L'idea che sta alla base di ReAct è quindi quella di aumentare lo spazio delle azioni dell'agente a $\widehat{A} = A \cup \mathcal{L}$, dove $\mathcal{L}$ è lo spazio del linguaggio naturale. Un'azione $\widehat{a}_t \in \mathcal{L}$ nello spazio linguistico, che chiameremo *though* (pensiero), non influenza l'ambiente esterno e quindi non porta ad una osservazione. Al contrario, un pensiero $\widehat{a}_t$ mira a comporre informazioni utili ragionando sul contesto corrente c_t, e aggiornare il contesto $c_{t+1} = (c_t, \widehat{a}_t)$ per supportare il ragionamento o l'azione futura. Come mostrato nella Figura 2.1, ci potrebbero essere vari tipi di pensieri utili, ad esempio decomporre gli obiettivi del compito e creare piani d'azione (2b, Act 1; 1d, Thought 1), integrare conoscenze di buon senso rilevanti per la risoluzione del compito (2b, Act 1), estrarre parti importanti dalle osservazioni (1d, Thought 2, 4), tracciare i progressi e adattare il piano d'azione (2b, Act 8), gestire eccezioni (1d, Thought 3), e così via.

Dal punto di vista tecnico, ReAct sfrutta un'architettura che integra moduli di ragionamento e moduli di azione, orchestrati attraverso un meccanismo di controllo che decide quando e come alternare tra ragionamento e azione. Questo meccanismo di controllo può essere basato su regole predefinite o su apprendimento automatico, dove il modello impara quali strategie di alternanza ottimizzano le prestazioni su specifici compiti attraverso esperienze precedenti e feedback continuo. In quest'ultimo caso si va quindi ad utilizzare un LLM stimolato con pochi esempi contestuali per generare sia azioni specifiche del dominio, sia pensieri linguistici in forma libera per la risoluzione dei compiti (Figura 2.1(1d), (2b)). Ogni esempio è un cammino di azioni, pensieri e osservazioni dell'ambiente per risolvere un'istanza del compito che rispecchia il tipico comportamento umano.

Questo ciclo di ragionamento e azione migliora notevolmente la precisione e l'utilità delle risposte del modello, rendendolo particolarmente adatto per

applicazioni dinamiche e complesse dove la base di conoscenza può cambiare rapidamente. Inoltre, grazie allo spazio di pensiero flessibile e al formato pensiero-azione, il modello riesce ad adattarsi molto facilmente per compiti molto diversi tra loro, con spazi d'azione e necessità di ragionamento distinti. ReAct eccelle anche in termini di interpretabilità e diagnosticabilità. Mentre i modelli che operano con azioni isolate possono rendere difficile per gli esseri umani capire da dove derivano le informazioni utilizzate per una particolare decisione e come sono giunti ad essa, rappresentando una sorta di black box statistiche che rendono impossibile analizzare e spiegarne il comportamento, ReAct facilita questo processo. Permette di distinguere chiaramente le informazioni derivate dalla conoscenza interna del modello rispetto a quelle ottenute attraverso interazioni con risorse esterne, riportando in maniera diretta ed in linguaggio naturale tutte quelle che sono state le sue osservazioni, i suoi ragionamenti, le sue azioni e le informazioni da esse ricavate. Quello che si ottiene rappresenta una sorta di log che dettagliatamente descrive tutti i "pensieri" e le conseguenti azioni e risultati intermedi compiuti dal modello fino alla generazione della risposta finale. Questa caratteristica delinea una forte innovazione rispetto ai tradizionali LLM, che risulta particolarmente importante in quei contesti critici dove la trasparenza e la verificabilità delle decisioni e delle fonti di un modello sono essenziali.

In conclusione, ReAct rappresenta un significativo passo avanti nel campo dgli LLM. Questo approccio non solo migliora le prestazioni e l'accuratezza dei modelli in compiti complessi, ma apre anche nuove possibilità per applicazioni in settori dove la capacità di interagire dinamicamente con l'ambiente esterno è cruciale.

2.2 Panoramica sugli LLM per la Lingua Italiana

2.2.1 Problematiche ed Ostacoli

Gran parte dello sviluppo e della ricerca sui Large Language Model si è fino ad oggi concentrata prevalentemente sulla lingua inglese, creando una disparità linguistica e culturale che risulta funzionalmente problematica. La

necessità di sviluppare LLM specifici per la lingua italiana deriva da una serie di considerazioni linguistiche, culturali e pratiche. L'italiano, come molte altre lingue romaniche, presenta infatti rispetto all'inglese peculiarità sintattiche, morfologiche e lessicali che richiedono un trattamento dedicato per essere adeguatamente rappresentate nei Language Model. La sintassi italiana, ad esempio, è caratterizzata da una struttura flessibile e una ricca morfologia che comprende numerose forme verbali e una complessa concordanza tra sostantivi, aggettivi e verbi. Queste caratteristiche rendono difficile per un modello addestrato prevalentemente su dati in inglese comprendere e generare testo in italiano con la stessa precisione. Per catturare le sfumature e le peculiarità dell'italiano, è essenziale disporre di modelli addestrati su dati specifici di questa lingua, che riflettano non solo le regole grammaticali, ma anche le usanze, le espressioni idiomatiche e il contesto culturale.

Una delle principali difficoltà nello sviluppo di LLM specifici per l'italiano risiede purtropppo anche nella disponibilità di dati di alta qualità per l'addestramento dei modelli. Mentre per l'inglese esistono vasti corpora testuali facilmente accessibili e ben annotati, la quantità di dati disponibili per l'italiano è notevolmente inferiore. Un'altra sfida è rappresentata dalla necessaria valutazione delle prestazioni dei modelli. Gli strumenti e i benchmark esistenti sono stati principalmente sviluppati per l'inglese (come ARC Challenge, HellaSwag, o MMLU) e non sono sufficientemente rappresentativi delle peculiarità della nostra lingua. Valutazioni non accurate o affidabili e difficoltà nei confronti prestazionali, hanno reso molto difficoltoso per i ricercatori valutare accuratamente la qualità dei modelli, ostacolato la capacità di identificare le aree di miglioramento e di far avanzare lo stato dell'arte nel campo degli LLM per l'italiano.

2.2.2 Ricerca e Sviluppi Recenti

Diversi sono stati i Language Model sviluppati specificamente per la lingua italiana. Uno dei primi modelli significativi è stato GePpeTto (De Mattei et al., 2020 [7]). Rappresenta un adattamento di GPT-2 per la lingua italiana, sfruttando la potenza di questo modello per compiti specifici in italiano. Presentava

però limitazioni in termini di capacità di generare testo che catturasse pienamente la complessità morfologica e sintattica dell'italiano. Successivamente, modelli come Alpaca (Taori et al., 2023 [8]) e Camoscio (Santilli e Rodolà, 2023 [9]) hanno tentato di migliorare ulteriormente le capacità degli LLM italiani concentrandosi in particolare sull'uso di adapters. Questi ultimi permettono di personalizzare e fare fine-tuning di LLM preesistenti, per adattarli a compiti specifici senza dover addestrare nuovamente l'intera rete neurale. Gli adapters sono moduli parametrizzati ed allenati sullo specifico task da svolgere, che vengono inseriti tra i layer del modello di base, consentendo un aggiornamento efficiente ma mirato che non modifica in alcun modo gli originali coefficienti interni alla rete. Un altro rilevante LLM nel contesto italiano è stato Fauno (Bacciu et al., 2023 [10]), basato su Baize (Xu et al., 2023 [11]), il quale si proponeva di incrementare le capacità di text generation migliorando la coerenza e la rilevanza del testo prodotto rispetto ai modelli precedenti.

Questi ultimi approcci sono in grado di formulare testo coerente e grammaticalmente corretto, ma spesso mancano della profondità necessaria per catturare le sfumature semantiche e culturali della lingua italiana. Le frasi generate sono generalmente fluide, ma a volte risultavano artificiali o prive di contesto. Inoltre la complessità morfologica dell'italiano, con la sua vasta gamma di coniugazioni verbali e accordi grammaticali, comporta una precisione non sempre ottimale, con errori occasionali nella flessione verbale e nella concordanza tra sostantivi e aggettivi. Nei compiti specifici come la traduzione automatica o la risposta a domande questi i modelli offrono buone prestazioni, ma vengono spesso superati dai loro equivalenti inglesi o multilingua. Questo dovuto in parte alla mancanza di dati di addestramento di alta qualità e in parte alla scarsa ottimizzazione dei benchmark per la lingua italiana.
Receni e significativi passi avanti sono stati fatti in un recentissimo progetto di ricerca di Bacciu et al, 2024 [12], con l'obiettivo di creare un LLM che non solo sia altamente performante, ma anche culturalmente e linguisticamente sensibile: DanteLLM. Quest'ultimo è costruito sulla base di Mistral (Jiang et al., 2023 [13]), un modello con 7 miliardi di parametri che non solo supera il precedente stato dell'arte open-source stabilito da LLaMA2, ma ottiene costantemente prestazioni superiori rispetto a tutti i precedenti LLM italiani, nonostante il

suo pre-addestramento in una lingua inglese. Il team di ricerca ha effettuato un fine-tuning LoRA con quantizzazione a 8 bit del modello di base, preservando il più possibile la conoscenza di Mistral ed allineandolo a comprendere e generare testo italiano fluente. Una delle innovazioni principali riguarda l'utilizzo di un vasto corpus di dati testuali italiani per l'addestramento del modello, raccolti da una varietà di fonti che includono letteratura, giornalismo, documenti tecnici e testi accademici. A differenza della maggior parte degli LLM italiani (es. Fauno e Camoscio) che al contrario sfruttano dataset generati con ChatGPT, rendendoli inoltre inadatti per scopi commerciali. A tale scopo sono state proposte due differenti varianti del modello: OpenDanteLLM, completamente open-source poiché allenato sul dataset Italian SQuAD (Croce et al., 2018 [14]) e su di 25.000 frasi estratte dal dataset Europarl (Koehn, 2005 [15]) (Inglese-Italiano), e DanteLLM, che rappresenta invece il modello con le migliori prestazioni, allenato sui due dataset sopra menzionati e su due dei migliori dataset di addestramento italiani per LLM, ovvero il dataset Quora di Fauno e il dataset Camoscio.

Il progetto di presentazione di DanteLLM incorpora anche lo sviluppo di un benchmark specifico per la lingua italiana per fornire una base solida e standardizzata di valutazione dei modelli LLM italiani. Il primo passo è stato selezionare una serie di benchmark esistenti per la lingua inglese, noti per la loro capacità di coprire un'ampia gamma di task NLP. I prescelti sono stati successivamente tradotti in italiano utilizzando tecniche di traduzione automatica. Questo processo ha necessitati l'uso di modelli di traduzione automatica di alta qualità per garantire che le domande e i task rimanessero fedeli all'originale in termini di difficoltà e contenuto. Per assicurare l'accuratezza e la fedeltà delle traduzioni, sono state eseguite due tipi di analisi della qualità delle traduzioni. Una prima valutazione umana da parte di esperti linguistici che hanno esaminato un campione delle traduzioni per valutarne eventuali problemi o discrepanze, ed una seconda valutazione generale effettuata utilizzando ChatGPT-4, che ha analizzato le traduzioni e fornito un punteggio di qualità basato su criteri predefiniti. Il benchmark italiano comprende diverse metriche, ognuna delle quali mira a testare aspetti specifici:

- **AI2 Reasoning Challenge (ARC Challenge)**: Questo benchmark è

stato tradotto per valutare la capacità dei modelli di risolvere problemi di ragionamento scientifico e logico. ARC Challenge [16] include infatti domande in materia scientifica di livello scolastico. La traduzione in italiano ha mantenuto la complessità e la struttura delle domande originali, assicurando fossero culturalmente e linguisticamente appropriate.

- **HellaSwag**: Progettato per testare la comprensione contestuale e la generazione del testo, HellaSwag [17] richiede ai modelli di completare frasi e storie in modo coerente e logico. Il benchmark presenta una serie di scenari descrittivi incompleti e il modello deve scegliere la continuazione più appropriata. La traduzione ha adattato questi scenari per preservare il contesto e l'intento originale, rendendo le opzioni di completamento equivalenti e pertinenti in italiano.

- **Massive Multitask Language Understanding (MMLU)**: Questo benchmark copre una vasta gamma di compiti di comprensione linguistica. MMLU [18] include domande su vari argomenti, dalla storia alla biologia, e valuta la capacità del modello di rispondere correttamente a domande di conoscenza generale e specifica. La traduzione ha adattato le domande per essere culturalmente rilevanti e linguisticamente precise, mantenendo l'integrità del contenuto originale.

- **TruthfulQA**: Mirato a testare l'accuratezza e l'integrità delle risposte generate. TruthfulQA [19] presenta presenta affermazioni e domande, progettate al fine di mettere alla prova la capacità del modello nel fornire risposte veritiere e precise a domande complesse. La traduzione ha mantenuto la struttura e la complessità delle domande, assicurando che le risposte richiedessero un ragionamento accurato e un alto livello di comprensione linguistica.

Il benchmark è stato implementato in un ambiente di test standardizzato, permettendo una valutazione oggettiva e comparativa dei modelli LLM. I risultati ottenuti dai vari modelli testati (italiani e non) sono stati poi confrontati per identificare le aree di forza e debolezza. Questo processo ha incluso la

creazione di una leaderboard[1] pubblica, ispirata alla famosa HuggingFace Leaderboard[2], dove i risultati dei modelli sono stati resi liberamente visibili per promuovere la trasparenza e la competizione tra i ricercatori, oltre che in generale supportare lo sviluppo e la ricerca.

DanteLLM ha ottenuto risultati notevoli sul benchmark italiano, dimostrando un netto miglioramento rispetto a tutti i modelli precedenti. Ha evidenziato una superiore capacità di ragionamento scientifico e logico, affrontando le domande con maggiore accuratezza e comprendendo meglio i concetti scientifici. Ha mostrato una coerenza e fluidità superiori, con scelte di completamento più appropriate e contestualmente corrette, indicando un miglioramento nella comprensione contestuale e nella generazione del testo. Inoltre, ha superato gli altri modelli nelle domande di conoscenza generale e specifica, dimostrando una comprensione più profonda e accurata di una vasta gamma di argomenti, il che è particolarmente rilevante per applicazioni che richiedono un alto livello di conoscenza enciclopedica. Infine, le risposte di DanteLLM sono risultate più veritiere e precise, con una significativa riduzione degli errori di fattualità, dimostrando la capacità del modello di fornire informazioni accurate e affidabili in contesti complessi. Questi miglioramenti complessivi rendono DanteLLM particolarmente adatto per applicazioni che necessitano di una comprensione avanzata e di una generazione di testo di alta qualità in italiano.

I risultati ottenuti dimostrano che DanteLLM rappresenta un avanzamento significativo nel campo degli LLM per la lingua italiana. Inoltre, il nuovo benchmark ha fornito una piattaforma essenziale per la valutazione e l'ottimizzazione dei modelli. Tuttavia, rimangono alcune aree che richiedono ulteriore ricerca e sviluppo, come la continua raccolta e integrazione di dati di alta qualità, l'ulteriore raffinamento delle tecniche di adattamento e l'espansione dei benchmark specifici.

[1] https://huggingface.co/spaces/rstless-research/italian_open_llm_leaderboard

[2] https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

Capitolo 3

Metodo

3.1 Architettura Multi-Document Agent

Come già anticipato ParlaMentis si basa su di una innovativa architettura Multi-Document Agent. Possiamo vederla come una sorta di evoluzione della semplice architettura RAG (Sezione 1.3), che tenta di aggirarne e contrastarne i limiti integrandovi il paradigma ReAct (Sezione 2.1).

3.1.1 Limiti delle Soluzioni Precedenti

La più grande sfida da affrontare in questo progetto è infatti rappresentata dalla necessità di gestire un elevato numero di documenti ed informazioni di carattere generico, nel nostro specifico caso rappresentate dalla vastità di atti di iniziativa ed interventi parlamentari. Gli attuali LLM allo stato dell'arte sono capaci di immagazzinare conoscenza ad un volume elevato, grazie ad un altrettanto elevato numero di parametri, il che però comporta sfide significative in termini di risorse computazionali e memoria, rallentando i processi di aggiornamento e miglioramento. D'altra parte, l'uso di un grande numero di parametri non garantisce sempre una comprensione approfondita e contestuale, portando a risposte potenzialmente imprecise.
I modelli RAG rappresentano un'alternativa promettente, poiché invece di memorizzare una vasta quantità di conoscenza all'interno del modello stesso, i RAG combinano la generazione di testo con la capacità di recuperare informa-

zioni da una base di conoscenza esterna. Questo approccio consente di ridurre significativamente il numero di parametri necessari, migliorando l'efficienza e l'aggiornabilità. Questo approccio può però essere fortemente limitato da un recupero eccessivo di contesti, che di conseguenza possono sovraccaricare il modello con informazioni non pertinenti, aumentando la complessità computazionale ma soprattutto la probabilità di generare informazioni errate o ambigue. La situazione diventa ancor più complessa in quei casi, come ad esempio il contesto giuridico, in cui la base di coscienza dalla quale recuperare informazioni per la text generazion risulta fortemente interconnessa ed auto-referenziata.

Questo trova conferma nel fatto che il linguaggio naturale è intrinsecamente ridondante [20], caratteristica che può essere utile per la comprensione umana, ma non necessaria per i LLMs. Jiang et al. [21] dimostrano come la performance di questi ultimi diminuisca all'aumentare del numero di documenti recuperati, perché con loro cresce il rumore introdotto. Figura 3.1 mostra questi risultati sulla generazione multi-documento di domande e risposte, completamento di codice e sintesi astrattiva.

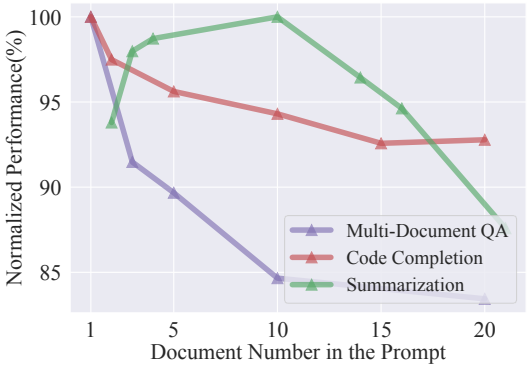

Figura 3.1: Performance dei modelli in funzione del numero di documenti recuperati.

3.1.2 Descrizione Achitetturale

L'architettura Multi-Document Agent è specificatamente progettata per gestire e interrogare vasti insiemi di documenti, andando a sfruttare le capacità di agenti ReAct specializzati, che collaborano tra di loro per rispondere a domande complesse e che richiedono l'accesso simultaneo a più fonti diverse. Questo approccio è particolarmente utile ed efficace per task che richiedono estrazione di informazioni precise o riassuntive da un dato documento, così

come in task che richiedono invece la comparazione o la messa in relazione di più documenti tra loro.

L'architettura si compone innanzitutto di un *top-level agent*, colui il quale interagisce effettivamente con l'utente, ne riceve la domanda ed è incaricato di generare la risposta finale. Quando viene fatta una richiesta all'agente, questo si occupa di interpretarla, valutarne il grado di complessità e, nel caso, scomporla in sotto-problemi più semplici e circoscritti. Completata la fase di osservazione e ragionamento, l'agente elabora e implementa i suoi piani d'azione, adattandoli se necessario e gestendo le eventuali eccezioni. La successiva fase di azione consente all'agente di interagire con varie risorse esterne (chiamate anche *tools*) per acquisire le informazioni che ritiene necessarie ai fini della risposta.

Questi *tools* per il recupero della conoscenza esterna non sono altro che ulteriori sotto-agenti ReAct, ognuno assegnato ad uno dei diversi documenti a disposizione. Ogni *document agent* è responsabile per l'elaborazione e la gestione di quel singolo documento. Questi tool sono in grado di rispondere a domande specifiche o generare riepiloghi dettagliati riguardanti il solo documento assegnato. Il top-agent può quindi scegliere dinamicamente se agire sul *document agent* effettuando una ricerca semantica o riepilogativa, a seconda della natura della query.
Il sotto-agente deve essere accompagnato da una sua precisa descrizione in linguaggio naturale, che riporti in maniera dettagliata i contributi informativi ed i task che l'agente è in grado di soddisfare. La chiarezza e l'esaustività delle varie descrizioni sono di fondamentale importanza per determinare l'efficacia e la qualità del modello, poiché il *top-level agent* andrà ad utilizzare proprio questi testi per determinare a quale *document agent* rivolgersi e quali azioni può soddisfare, al fine di ricostruire il contesto necessario per una risposta verso l'utente quanto più fattuale e completa.

Il *top-level agent* fa uso di un tool retriever interno, che entra in azione subito dopo la prima fase di osservazione e ragionamento post richiesta utente. Essi si occupa di recuperare, tra tutto l'insieme dei tool disponibili nell'architettura, solamente i top-k che raggiungono similarità maggiore con la query. Questo iniziale retrieval dei soli *document agent* ritenuti più adatti e correlati alla domanda viene fatto sulla base del contenuto delle descrizioni degli agenti,

sottolineando ancora una volta la loro fondamentale importanza in termini di chiarezza e completezza al fine di eseguire un retrieval corretto ed esaustivo dei contesti necessari.
Il tool retriever svolge quindi il compito di selezionare a monte un ristretto numero di sotto-agenti che saranno gli unici visibili ed effettivamente utilizzabili dal *top-level agent* per la specifica domanda utente. Differentemente invece da quanto farebbe invece il ReAct agent di base, il quale tenderebbe ad utilizzare o quantomeno vagliare tutti i tool a disposizione, alla ricerca di quelli più adatti. Il tool retriever limita quindi l'effettivo contesto informativo accessibile dal modello per singola query, che altrimenti risulterebbe ragionevolmente troppo ampio e spesso poco utile ai fini della risposta. Questo garantisce un incremento significativo della precisione del modello, come un decremento invece della possibilità di allucinazioni date dall'accesso a contesti troppo vasti o poco pertinenti
Una versione più avanzata dell'architettura Multi-Document Agent prevede che il tool retriever, durante la fase di recupero dei top-k tool da dare in pasto al top agent, aggiunga in coda a questo insieme di *document agent* un ulteriore e nuovo *query planning tool*. Si tratta di un agente ReAct che viene dinamicamente creato ad ogni nuova query sulla base dei tool appena recuperati. Il *query planning tool* viene appositamente predisposto per soddisfare task di tipo comparativo e riassuntivo per compiti che vedono necessario l'accesso simultaneo a documenti multipli. Si occupa di decomporre la richiesta che riceve in sotto-task e di accedere ai vari *document agent* recuperati dal tool retriever, dei quali anche lui tiene un riferimento interno, per riportare al *top-level agent* informazioni generali e di orientamento semantico sul contesto a disposizione. Lo scopo del *query planning tool* è quindi quello di rafforzare la capacità di ragionamento e di accesso pianificato ai tool, mettendo a disposizione un sotto-agente specificatamente incaricato alle analisi comparative ed esplorative ad ampio spettro, su quelle che sono le effettive risorse recuperate ed utilizzabili per una data query. Il *top-level agent* farà uso di questo agente nei casi in cui non è subito in grado di capire verso quale specifico tool appellarsi per recuperare i dati di cui necessita, delegando al *query planning tool* la generazione di un primo piano d'azione dettagliato.

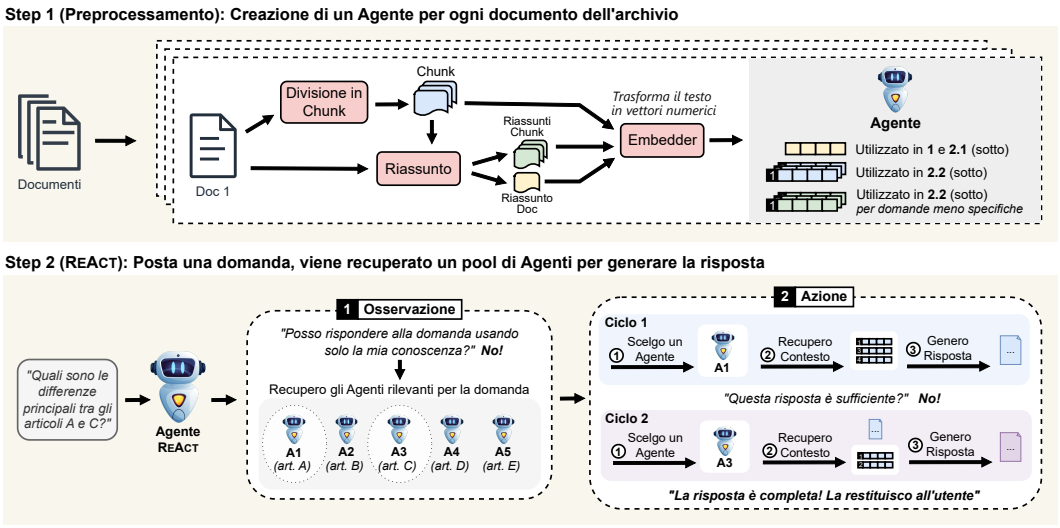

Figura 3.2: Architettura Multi-Document Agent impiegata nel contesto di ParlaMentis. Nell'esempio sono illustrati 5 documenti e 2 cicli di azione.

La Figura 3.2 illustra le fasi del funzionamento dell'architettura:

- **Step 1**: Inizialmente si effettua una prima fase di preprocessamento dei dati. In questa fase, ogni documento viene prima suddiviso in più text chunk di lunghezza fissa e successivamente viene creato il relativo *document agent*. Questo contiene diverse informazioni strutturate a multi-livelli[1]: il riassunto ad alto livello in linguaggio naturale del documento (rappresentata in giallo), una descrizione dettagliata di ciascun chunk (in blu), e un riassunto di ogni chunk (in verde).
  L'architettura prevede che la descrizione da associare al *document agent* venga fatta generare dall'agente stesso, interrogandolo con un apposito prompt mirato a produrre un conciso ma informativo riassunto del documento e del suo contenuto.
  È consigliabile eseguire questa fase di preprocessamento dei dati prima dell'effettiva messa in opera dell'architettura, prevedendo per ogni documento il salvataggio su memoria di massa dei chuck di testo e della descrizione generati. In questo modo, per le successive esecuzioni sarà

[1] Ogni informazione è rappresentata tramite embedding, cioè un vettore numerico ad alta dimensionalità.

possibile ri-creare i vari *document agent* andando a recuperarne questi dati precedentemente generati e totalmente riutilizzabili direttamente da file, senza la necessità di produrli nuovamente di volta in volta. Questo approccio ottimizza di molto le tempistiche e le necessità computazionali dell'architettura.

- **Step 2**: Quando viene formulata una domanda, il *top-level agent* valuta la sua complessità e determina se le sue conoscenze sono sufficienti per fornire una risposta. Se necessario, l'agente consulta il suo database di tool esterni per ottenere informazioni aggiuntive. Questo ciclo di osservazione e azione può ripetersi più e più volte fino a quando il modello non ritiene di aver raggiunto una risposta completa e accurata, o nel caso in cui si rende conto di non avere accesso ad informazioni sufficienti.

L'architettura Multi-Document Agent porta con se numerosi vantaggi. Il primo e più banale sta nel fatto che, rispetto alle normali architetture RAG, si aggiungono tutte le peculiarità del paradigma ReAct. Ad esempio la capacità del modello di alternare fasi di ragionamento interno, dove elabora e pianifica la prossima azione sulla base delle informazioni che ha a disposizione, e fasi di azione esterna, dove interagisce con varie sorgenti di dati per la raccolta di ulteriori contesti utili alla risposta. Il modello risulta inoltre essere capace di riportare in linguaggio naturale tutte le osservazioni, i pensieri, le azioni ed i rispettivi risultati intermedi che genera per giungere alla risposta. Questo consente una facile analisi del suo comportamento ed una altissima interpretabilità del testo generato.
L'architettura gode inoltre di alta efficienza computazionale, data dall'integrazione del modulo di toll retrieving e del *query planning tool*. Questi consentono una pianificazione sistematica del piano d'azione del modello e la selezione di un ristretto insieme di contesti effettivamente accessibili, in relazione alle specifiche necessità della domanda formulata dall'utente.
La Multi-Document Agent assicura anche ampia scalabilità, essendo appositamente progettata per essere in grado di gestire grandi volumi di dati, ma soprattutto risulta facilmente estendibile in base ad eventuali necessità future. L'aggiunta di nuovi documenti, o di nuovi servizi esterni, richiede semplice-

mente la costruzione e la predisposizione di ulteriori sotto-agenti, senza che ciò comporti impatti significativi sulla struttura pre-esistente.
La divisione gerarchica tra *top-level agent* e *document agent* rende infine il sistema altamente modulare. Ogni componente può essere sviluppato, testato e migliorato indipendentemente, facilitando l'aggiornamento e la manutenzione del sistema. Questa modularità rende inoltre l'architettura Multi-Document Agent altamente flessibile ed adattabile a molteplici contesti applicativi, anche molto differenti tra di loro. Nessuno vieta infatti di utilizzare, all'interno dei vari agenti ReAct, LLM diversi, più adeguati, o finemente parametrizzati allo specifico tipo di sotto-task da compiere o alla diversa natura delle informazioni da dover gestire. Sarebbe anche possibile implementare nei diversi agenti dei sotto-moduli o particolari scelte architetturali che meglio si adattano alle situazioni; un esempio è infatti rappresentato dal caso del tool retrieval all'interno del *top-level agent*. Potrebbe inoltre risultare utile la possibilità di aggiungere, nell'insieme dei vari *document agent*, ulteriori e diversi tool o risorse informative, non per forza incarnati da altri agenti ReAct, ma che invece possono essere semplici database vettoriali, API di servizi esterni, strumenti di ricerca sul web, ecc. Le possibilità di personalizzazione sono perciò infinite e permettono di costruire un modello finale totalmente aderente alle proprie necessità.

3.2 Datasets

3.2.1 Dataset della Camera dei Deputati

In questa sezione discuteremo fase di della costruzione dei dataset che compongono la base di conoscenza sulla quale ParlaMentis effettua il retrieval delle informazioni pertinenti alla generazione delle sue risposte.

La Camera dei Deputati italiana rende infatti accessibile il suo vasto patrimonio informativo attraverso l'apposito sito web ufficiale dati.camera.it. Tale piattaforma mette a disposizione la cosiddetta Ontologia della Camera dei Deputati (OCD), uno strumento sviluppato per promuovere la trasparenza e l'accessibilità dei dati pubblici. L'OCD, seguendo le migliori pratiche

internazionali per la pubblicazione di Linked Open Data, offre un catalogo completo di dati e documenti digitali che descrivono tutte le risorse (persone, eventi, atti, ecc.) connesse alla Camera e alla sua attività.

L'OCD è costruita utilizzando il Web Ontology Language[2] (OWL), un linguaggio standard offerto dalla W3C per definire ontologie sul web. OWL permette di descrivere la struttura e la semantica dei dati da catalogare ed archiviare, attraverso un sistema di classi e proprietà, facilitando l'interoperabilità tra diversi sistemi informativi. Le varie risorse disponibili sono composte da una parte di metadati descrittivi e da un'altra parte di riferimenti ad ulteriori risorse interne al dominio o a risorse web esterne. La struttura dei dati è espressa in formato Resource Description Framework[3] (RDF), uno strumento standard di codifica per i Linked Open Data. Si basa su triplette che consentono la rappresentazione di informazioni interconnesse sul web sotto forma di grafo. Ogni tripla RDF è composta da un soggetto, un predicato e un oggetto, formando una struttura progettata per il collegamento semantico di diverse risorse.
L'elemento centrale dell'ontologia è il concetto di "Legislatura", attorno al quale sono organizzate altre entità come organi, gruppi parlamentari, deputati, atti e documenti. Questa organizzazione riflette la complessa attività della Camera dei Deputati e permette di tracciare in modo dettagliato e coerente la storia legislativa e le sue dinamiche istituzionali, dalla I Legislatura del Regno di Sardegna (1848) ad oggi con la XIX Legislatura della Repubblica.

Uno degli aspetti chiave dell'ontologia è l'esposizione di un endpoint[4] SPARQL, che consente l'interrogazione online diretta a tutto il database attraverso query SPARQL, un linguaggio di query appositamente progettato per l'interrogazione di dati in formato RDF. Questo endpoint ci ha permesso quindi di eseguire le necessarie interrogazioni all'OCD per la raccolta massiva dei dati da utilizzare in ParlaMentis. Nello specifico parliamo degli atti di iniziativa prodotti dalla Camera e delle video-registrazioni degli interventi avvenuti nelle varie sedute di dibattito parlamentare.

[2] https://www.w3.org/OWL/
[3] https://www.w3.org/RDF/
[4] https://dati.camera.it/sparql

Le interrogazioni SPARQL producono in output un file di testo, in formato selezionabile tra quelli più utilizzati per la memorizzazione di dati strutturati (nel nostro caso abbiamo scelto il formato JSON). Questo file include vari metadati descrittivi dei record recuperati, ma, al contrario, le eventuali risorse esterne associate ai record vengono riportate come semplici riferimenti URL. Un esempio significativo è rappresentato proprio dagli atti di iniziativa, per i quali è si possibile recuperare le varie informazioni descrittive contenute nei metadati all'interno dell'OCD, tuttavia il testo completo dell'atto non è direttamente presente nel dataset, ma viene al contrario fornito come link esterno che rimanda ad una pagina web dove il documento completo è disponibile come file da scaricare in formato PDF.

Il nostro team ha dovuto perciò svolgere un complesso lavoro di ricostruzione dei dataset necessari per ParlaMentis. Questo ha comportato innanzitutto l'implementazione delle query SPARQL necessarie per il recupero delle risorse della Camera e dei loro metadati. Successivamente, il nostro team ha dovuto sviluppare degli script Python appositamente progettati per ripulire, filtrare ed etichettare correttamente questi dati, ma soprattutto per eseguire operazioni automatiche di web scraping al fine di estrarre, dalle relative pagine web, le risorse esterne associate.

Di seguito andremo ad illustrare dettagliatamente le strategie utilizzate, andando a distinguere tra l'approccio seguito per la costruzione del dataset degli atti di iniziativa e quello invece seguito per il dataset dei video-interventi.

Dataset degli Atti di Iniziativa

Discutiamo quindi la costruzione del dataset contenete gli atti di iniziativa prodotti dalla Camera dei Deputati.

La Figura 3.3 sotto mostra la query SPARQL che abbiamo utilizzato all'interno dell'endpoint di accesso ai dati nell'OCD. La query permette il recupero di tutti gli atti prodotti in una specificata legislatura della Repubblica. Per ogni atto abbiamo ottenuto il suo ID univoco, il tipo di iniziativa (parlamentare, governativa, popolare, etc.), il titolo dell'atto, la data di proposta, la data dell'eventuale approvazione dell'atto, la legislatura nella quale è stato discusso

```
SELECT DISTINCT
    REPLACE(?ID, '"', '') AS ?ID,
    REPLACE(?iniziativa, '"', '') AS ?iniziativa,
    REPLACE(?titolo, '"', '') AS ?titolo,
    REPLACE(?dataProposta, '"', '') AS ?dataProposta,
    REPLACE(?dataApprovazione, '"', '') AS
        ?dataApprovazione,
    strafter(str(?rifLegislatura), "_") AS ?legislatura,
    ?risorsa
WHERE{
    ?atto a ocd:atto;
        ocd:iniziativa ?iniziativa;
        dc:identifier ?ID;
        dc:relation ?risorsa;
        ocd:rif_leg ?rifLegislatura;
        dc:date ?presentazione;
        dc:title ?titolo;
        ocd:rif_statoIter ?statoIter.
    ?statoIter dc:date ?dataProposta.
    ?rifLegislatura dc:title ?legislatura.
    OPTIONAL{
        ?votazione a ocd:votazione; ocd:rif_attoCamera
            ?atto;
            ocd:approvato "1"^^xsd:integer;
            ocd:votazioneFinale "1"^^xsd:integer;
            dc:date ?dataApprovazione.
    }
    FILTER(REGEX(?legislatura,'XIX','i'))
}
```

Figura 3.3: Query SPARQL per il recupero degli atti prodotti dalla Camera in una Legislatura specificata. In questo esempio viene presa in considerazione la XIX legislatura.

l'atto, ed infine un link alla pagina web esterna contenente il file PDF con il testo effettivo dell'atto parlamentare.

Quelli che abbiamo recuperato sono due file in formato JSON, i quali contengono tutti gli atti di iniziativa prodotti dalla Camera durante rispettivamente la XIX e XVIII Legislatura. Dato il numero comprensibilmente molto elevato di record recuperati, abbiamo deciso di selezionarne un insieme ristretto ma sufficiente per il corretto funzionamento e la valutazione del prototipo che stiamo implementando. Nello specifico abbiamo cercato di tenere in considerazione gli atti che facessero riferimento ad uno degli argomenti più importati ed al centro del dibattito pubblico negli ultimi anni, ossia il tema del cambiamento climatico ed in generale della salvaguardia ambientale: è stato fatto un filtraggio tramite keyword dei record recuperati, popolando il nostro dataset finale con i soli documenti che riportassero la parola "*ambiente*" all'interno della stringa che rappresenta il titolo dell'atto parlamentare. La Figura 3.4 sotto mostra un

piccolo estratto dei vari record ottenuti.

Nell'insieme degli atti così a disposizione, possono esserci alcuni di questi ai quali viene associata più di una risorsa. Queste corrispondono alle varie versioni dello stesso documento prodotte durante l'attività parlamentare. Un tipico esempio è il caso in cui un dato atto sia stato approvato dalla Camera, oppure sia stato modificato per correggere errori grammaticali o lessicali; queste nuove versioni del testo vengono archiviate all'interno dell'OCD come ulteriori risorse PDF, ma comunque restano associate alla medesima istanza di atto parlamentare.


```
{
    "ID": {
        "type": "literal",
        "value": "2572"
    },
    "iniziativa": {
        "type": "literal",
        "value": "Governo"
    },
    "titolo": {
        "type": "literal",
        "value": "Rendiconto generale dell'Amministrazione dello Stato per l'esercizio finanziario 2019 (2572)"
    },
    "dataProposta": {
        "type": "literal",
        "value": "20200729"
    },
    "legislatura": {
        "type": "literal",
        "value": "18"
    },
    "risorsa": {
        "type": "uri",
        "value": [
            "http://documenti.camera.it/_dati/leg18/lavori/stampati/pdf/18PDL0110800.pdf",
            "http://documenti.camera.it/_dati/leg18/lavori/stampati/pdf/18PDL0110840.pdf",
            "http://documenti.camera.it/_dati/leg18/lavori/stampati/pdf/18PDL0110590.pdf",
        ]
    }
},
```


Figura 3.4: Estratto del dataset contenente gli atti della Camera dei Deputati recuperati dall'OCD.

Il passo successivo è stato quindi quello di sviluppare uno script in linguaggio Python che si occupi, in maniera completamente automatica, di scaricare i file PDF associati agli atti recuperati e filtrati, di leggerne il contenuto, per infine archiviare il testo dell'atto ed i relativi metadati descrittivi all'interno di un unico file JSON finale. Quest'ultimo rappresenta il dataset degli atti di

iniziativa della Camera dei Deputati che viene utilizzato come base di retrieval in ParlaMentis.

Lo script da noi implementato non fa altro che elaborare uno per uno i record a disposizione dopo la precedente fase di filtraggio. Per ogni atto si occupa innanzitutto di controllare quante risorse esterne ha associate. Nel caso in cui, come precedentemente descritto, esistano più versioni dello stesso documento, si procede a selezionare solo la più recente. Il riferimento alla risorsa esterna è infatti un link di accesso diretto al file PDF; questi file vengono denominati con un codice numerico identificativo progressivo che consente una facile identificazione del documento più recente.
Successivamente lo script si occupa di scaricare l'unica risorsa esterna rimasta associata all'atto, utilizzando la libreria `Requests`. Questa è una facile e versatile utility progettata per l'invio di richieste HTTP e per la gestione delle risposte. Il file PDF contenente il testo del documento è stato quindi recuperato effettuando una richiesta GET al link riportato nel dataset. Una volta scaricato il file è necessario leggerne il contenuto e decodificarlo in semplici stringhe di testo, la quali saranno poi date in pasto al nostro modello.

La difficoltà maggiore in questa fase sta nel fatto che PDF è un formato di file finalizzato principalmente alla presentazione visuale di documenti, che si caratterizza da una struttura interna non testuale e molto complessa, adattata ed opportunamente codificata per la costruzione di layout di pagina sofisticati, per la gestione delle immagini, dei font, ecc. Non si tratta sicuramente di un formato progettato per l'archiviazione e l'accesso semplificato a dati testuali.

Un'ultima questione da affrontare riguarda il layout utilizzato in questi file. Il contenuto del documento è infatti composto da più parti, come ad esempio una prima descrizione generale con riferimento ai soggetti che lo hanno proposto, gli eventuali pareri espressi dalle varie Commissioni parlamentari, ed ovviamente il testo vero e proprio dell'atto di iniziativa. Se la versione del documento che abbiamo a disposizione risulta quella che è stata effettivamente approvata dalla Camera dei Deputati, le pagine del file PDF che contengono il testo vero e proprio, solamente quelle pagine, sono scritte in layout a due colonne. Questo poiché nella colonna di sinistra viene riportato il testo orinale dell'atto, mentre nella colonna di destra vengono riportate le modifiche apportate dalla Camera

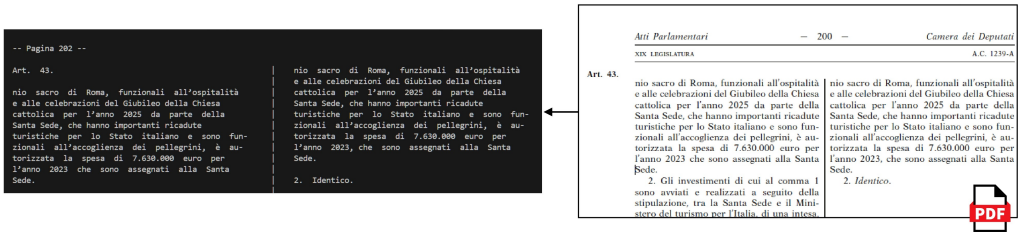

Figura 3.5: Estratto della pagina di uno dei file PDF scaricati, formattata in due colonne e contenente il testo effettivo dell'atto di iniziativa approvato (a destra); la rispettiva stringa di testo estrapolata dalla stessa pagina per mezzo del nostro script Python (a sinistra).

durante il processo di approvazione dello stesso. È importante precisare come la colonna di destra non contiene tutto il testo con tanto di modifiche finali, ma al contrario riporta esplicitamente solo e soltanto quelle che sono le parole, le frasi o i concetti alterati rispetto al testo di sinistra; tutto quello che è invece rimasto invariato viene indicato come appunto "identico". La Figura 3.5 mostra (a destra) un estratto della pagina di uno dei file PDF scaricati, formattata in due colonne e contenente il testo effettivo dell'atto di iniziativa approvato.
Risulta perciò fondamentale, ai fini di una corretta interpretazione da parte del nostro modello del contenuto informativo presente nell'atto parlamentare, riuscire a decodificare e ricostruire la corretta formattazione del testo riportato nei file.

A tale scopo, dopo una serie di prove e ricerche, abbiamo optato per l'utilizzo della libreria `PDFMiner`, dedicata all'estrazione di testo ordinato, e correlato da una serie di metadati descrittivi, dai file in formato PDF. Il nostro script Python è in grado quindi di estrarre il contenuto del documento e leggerlo pagina per pagina. Per ogni pagina dapprima verifica il layout della stessa. Se la pagina risulta scritta in una singola colonna procede con la semplice estrazione sequenziale delle stringhe di testo che la compongono, ed esegue una pulizia generale eliminando eventuali errori di lettura, pagine vuote o header di pagina inutili. Al contrario, se l'algoritmo rileva che la pagina è organizzata in due sezioni destra e sinistra, si vede necessaria una ulteriore elaborazione. La libreria infatti non è in grado di per se di leggere separatamente le due

colonne, ne è in grado di riconoscere la linea separatrice di mezzo. Lo script si occupa perciò di recuperare il contenuto della pagina per singola parola, e di volta in volta valutare se la stessa fa parte di una o dell'altra colonna. Questo è possibile grazie alla serie di metadati che la libreria estrapola ed associa agli elementi (in questo caso la singola parola) letti dal file PDF. Tra questi dati di corredo troviamo le coordinate spaziali dello specifico elemento rispetto allo spazio della pagina, le quali vengono utilizzate dall'algoritmo per determinare se la parola in questione si trova nella metà destra o in quella sinistra. Parola dopo parola lo script ricostruisce l'intera pagina del documento e ne mantiene il layout, per poi archiviarla insieme alle altre per poi continuare con la pagina successiva.
La Figura 3.5 mostra un piccolo estratto della pagina di uno dei file PDF scaricati, formattata in due colonne e contenente il testo effettivo dell'atto di iniziativa approvato (a destra), e la rispettiva stringa di testo estrapolata dalla stessa pagina per mezzo del nostro script (a sinistra).

Al termine di tutto il processo di recupero, filtraggio ed elaborazione degli atti di iniziativa della Camera dei Deputati, abbiamo ottenuto un dataset (archiviato in un apposito file in formato JSON) contenete in totale 68 documenti. Ricordiamo che per ogni documento si è tenuto traccia del suo ID univoco, del tipo di iniziativa (parlamentare, governativa, popolare, etc.), del titolo dell'atto, della data di proposta dell'atto originale, della data dell'eventuale approvazione dell'atto finale, della legislatura nella quale è stato discusso l'atto, ed infine di una lista di stringhe ordinate, ognuna delle quali rappresenta il contenuto delle varie pagine di cui è composto il file PDF associato all'atto.

atti totali	**68**
	Numero di Token
mean	6674
std	13982
min	350
25%	1630
50%	2635
75%	5166
max	73732

Tabella 3.1: Statistiche descrittive delle lunghezze (in token) del corpus di atti della Camera utilizzati in ParlaMentis.

La Tabella 3.1 riporta una serie di statistiche riguardanti le lunghezze dei testi estrapolati dagli atti del nostro dataset finale. La Figura 3.6 mostra con un

istogramma la distribuzione di queste lunghezze. La lunghezza del singolo testo viene calcolata sul numero dei token di cui è composto.
La Figura 3.7 mostra in un istogramma i tempi di costruzione del dataset, di cui lo script Python si è occupato di tenere traccia. Nello specifico riportiamo i tempi del solo download dei file PDF e i tempi totali di elaborazione del singolo atto (comprensivi di download del file ed estrapolazione del testo).

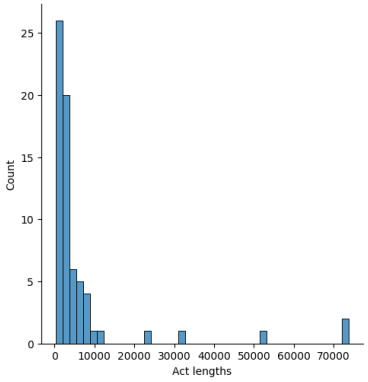

Figura 3.6: Distribuzione delle lunghezze (in token) dei testi degli atti parlamentari recuperati.

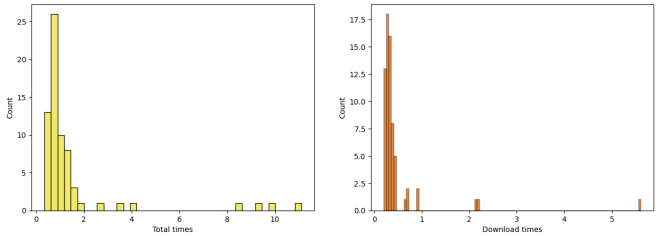

Figura 3.7: Distribuzione dei tempi di costruzione del dataset: a destra i tempi del solo download dei file PDF, a sinistra i tempi totali di elaborazione del singolo atto (comprensivi di download del file ed estrapolazione del testo).

Dataset dei Video-Interventi in Sedute della Camera

Discutiamo di seguito la costruzione del dataset contenete le video-registrazioni degli interventi avvenuti durante le varie sedute della Camera dei Deputati, sul quale poi il nostro modello ParlaMentis andrà a fare retrieval per la generazione delle sue risposte.

La Figura 3.8 sotto mostra le query SPARQL che abbiamo utilizzato all'interno dell'endpoint di accesso ai dati nell'OCD. La prima query (a sinistra) permette il recupero di tutti i deputati che compongono la Camera dei Deputati in una specificata legislatura della Repubblica. Per ogni deputato abbiamo ottenuto il suo nome e cognome, il patito di appartenenza (con indicazione della data di adesione e quella di eventuale abbandono), e l'eventuale incarico governativo ricoperto dal deputato (con indicazione della data di inizio mandato

e quella di eventuale fine). La seconda query invece (a destra) permette il recupero di tutti gli interventi avvenuti durante le sedute dell'Assemblea, filtrati per data e per keyword nel campo `argomento`. Per ogni intervento abbiamo ottenuto il nome e cognome dell'oratore, l'argomento al quale l'intervento fa riferimento, la data dell'intervento, la legislatura nella quale è avvenuto, ed infine un link esterno verso una pagina web.
La scelta di utilizzare due query differenti è stata obbligata poiché, richiedendo anche le informazioni del deputato (gruppo ed incarico) in una unica query di recupero degli interventi, l'endpoint SPARQL della Camera non riesce a gestire la richiesta, probabilmente troppo onerosa, e non genera alcun risultato. Abbiamo optato quindi per la divisione in due query distinte, sarà poi lo script Python a mettere in correlazione i dati .

```
SELECT DISTINCT
    REPLACE(?cognome, '"', '') AS ?cognome,
    REPLACE(?nome, '"', '') AS ?nome,
    REPLACE(?gruppo, '"', '') AS ?gruppo,
    REPLACE(?legislatura, '"', '') AS ?legislatura,
    ?incarico
WHERE {
    ?persona ocd:rif_mandatoCamera ?mandato.
    ?mandato ocd:rif_deputato ?deputatoID.
    ?deputatoID foaf:firstName ?nome;
                foaf:surname ?cognome.
    ?deputatoID ocd:aderisce ?rifGruppo;
                ocd:rif_leg ?idLegislatura.
    ?rifGruppo rdfs:label ?gruppo.
    ?idLegislatura dc:title ?legislatura.
    OPTIONAL {
        ?persona ocd:rif_membroGoverno
                 ?membroGoverno.
        ?membroGoverno dc:title ?incarico.
    }
    FILTER(REGEX(?legislatura,'XIX','i'))
}
```

```
SELECT DISTINCT
    REPLACE(?cognome, '"', '') AS ?cognome,
    REPLACE(?nome, '"', '') AS ?nome,
    REPLACE(?argomento, '"', '') AS ?argomento,
    REPLACE(?testo, '"', '') AS ?risorsa,
    REPLACE(?data, '"', '') AS ?data,
    strafter(str(?legislatura), "_") AS ?legislatura
WHERE {
    ?dibattito a ocd:dibattito;
                 ocd:rif_discussione ?discussione.
    ?discussione ocd:rif_seduta ?seduta.
    ?seduta dc:date ?data.

    ?discussione rdfs:label ?label.
    BIND(str(?label) AS ?argomento)
    FILTER(regex(?argomento,'clima','i'))

    ?discussione ocd:rif_intervento ?intervento.
    ?intervento ocd:rif_deputato ?deputatoId;
                dc:relation ?testo.
    ?deputatoId foaf:firstName ?nome;
                foaf:surname ?cognome;
                ocd:rif_leg ?legislatura.
    FILTER(REGEX(?data, \\
      '^(2023(0[3-9]|1[0-2])|2024(0[1-3]))','i'))
}
```

Figura 3.8: Query SPARQL per il recupero dei deputati della Camera per una data legislatura (a sinistra) e per il recupero degli interventi avvenuti in sedue della Camera (a destra). In questo esempio vengono recuperati i deputati della sola XIX Legislatura e gli interventi dal 03-2023 al 03-2024 con keyword "clima" nel campo `argomento`.

Dato il numero comprensibilmente molto elevato di interventi, abbiamo

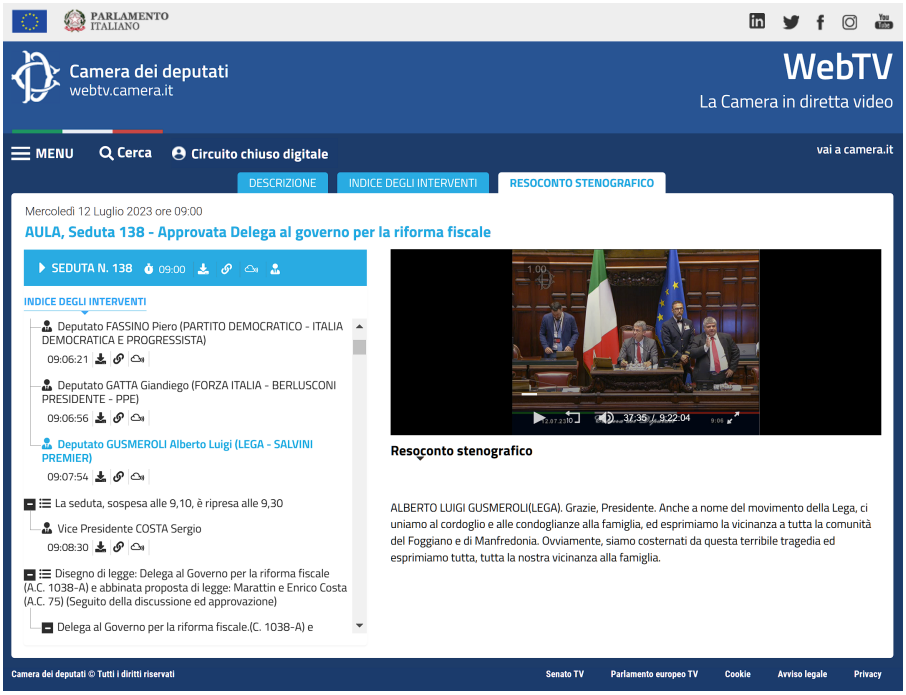

Figura 3.9: Portale WebTV della Camera dei Deputati. Nello specifico vediamo l'esempio di una delle pagine che permettono la visualizzazione e la navigazione delle sedute dell'Assemblea.

deciso di recuperarne un insieme ristretto ma sufficiente per il corretto funzionamento e la valutazione del prototipo che stiamo implementando. Nello specifico abbiamo cercato di tenere in considerazione i soli interventi avvenuti dal 03-2023 al 03-2024, e che riportassero all'interno del loro campo `argomento` le parole "*cliama*", "*rinnovabili*", "*pnrr*", "*immigrazione*", "*rinnovabili*", "*ucraina*". La scelta delle parole è stata fatta sia per fare riferimento ad alcuni dei temi più importati ed al centro del dibattito pubblico negli ultimi anni, ma soprattutto per selezionare gli interventi possibilmente più affini agli atti presenti nel nostro dataset di retrieval.

La Camera dei Deputati, sul sito webtv.camera.it, offre un portale ufficiale attraverso il quale vengono trasmessi in live streaming, e successivamente archiviati, i lavori dell'Assemblea in diretta, le sedute delle Commissioni permanenti, vari eventi culturali, ecc. Le sedute della Camera vengono indicizzate in tempo reale e sono rese disponibili in apposite pagine del sito. Qui è possibile

visionare e navigare la video-registrazione dell'intera seduta dell'Assemblea e viene inoltre messo a disposizione il resoconto stenografico dettagliato della seduta, riportante tutti gli interventi avvenuti. Per ogni intervento è possibile visionare la trascrizione testuale delle parole pronunciate dall'oratore, ed è infine disponibile un apposito tasto che consente il download della sola clip video inerente al dato intervento. La Figura 3.9 mostra uno screenshot esplicativo del portale WebTV.
La risorsa esterna associata all'intervento recuperato dall'OCD non fa però riferimento a questo portale, ma ad una completamente diversa pagina del sito camera.it contenente il solo resoconto stenografico in formato testo dell'intera seduta.

Il passo successivo è stato quindi quello di sviluppare uno script in linguaggio Python che si occupi, in maniera completamente automatica, innanzitutto di associare l'oratore del dato intervento con il suo gruppo politico di appartenenza e con l'eventuale incarico di governo. Successivamente lo script deve recuperare dalla specifica pagina del portale WebTV la trascrizione testuale dell'intervento, scaricare la relativa clip video, per infine archiviare il tutto all'interno di un unico file JSON finale. Quest'ultimo rappresenta il dataset degli interventi nelle sedute della Camera dei Deputati che viene utilizzato come base di retrieval in ParlaMentis.

Lo script da noi implementato non fa altro che elaborare uno per uno i record a disposizione. Per ogni intervento, si occupa per prima cosa di recuperare dal dataset dei deputati il partito politico al quale l'oratore appartiene, oltre che l'eventuale incarico di governo. Questo viene fatto per mezzo di una ricerca sul nome e cognome, e confrontando la data del dato intervento con i periodi di adesione o di incarico riportati nel dataset.

In seguito si passa alla fase di recupero della trascrizione e del video. In questo caso abbiano notato come nel link esterno associato all'intervento e da noi recuperato dall'OCD, sono presenti una serie di ID numerici utilizzati per l'identificazione univoca dello stesso intervento in tutto il dominio web della Camera. Questi stessi ID infatti, se opportunamente inseriti come parametri GET nel link di reindirizzamento al portale WebTV, consentono l'accesso alla pagina contenente la video-registrazione della seduta dell'Assemblea nella quale

```
    # ...
    wd = webdriver.Firefox(options=firefox_options)

    wd.get("https://webtv.camera.it/archivio?NumeroLegislatura="
           + speech_data.legislatura + "&NumeroSeduta="
           + re.search(r'\d+', speech_data.sed).group())

    speech_el = wd.find_element(By.XPATH, f"//*[contains(@data-steno_map,
    ↪  '{speech_data.tit}') and contains(@data-steno_map, '{speech_data.int}')]")


    id  = re.search(r'\d+', speech_el.get_attribute("id")).group()

    wd.execute_script(f"show_popups_resoconto_interventi({id}, 'download')")
    download_box = wd.find_element(By.ID, f"download-box-oratore-resoconto-item_{id}")
    download_link = download_box.find_element(By.TAG_NAME, "a").get_attribute("href")
    wd.quit()

    downloaded_file_name = None
    wd = webdriver.Chrome(service=Service(CHROME_DRIVER))
    wd.get(download_link)

    while True:
        time.sleep(3)
        files = os.listdir(DOWNLOAD_VIDEO_DIR)
        if (len(files) == 1) and (files[0].endswith(".mp4")):
            downloaded_file_name = files[0]
            break
    wd.quit()

    shutil.move(DOWNLOAD_VIDEO_DIR + downloaded_file_name,
                VIDEO_DIR + downloaded_file_name)
    os.rename(VIDEO_DIR + downloaded_file_name,
            VIDEO_DIR + speech_data.ID + ".mp4")
    for file in os.listdir(DOWNLOAD_VIDEO_DIR):
            os.remove(DOWNLOAD_VIDEO_DIR + file)
```

Figura 3.10: Estratto dello script Python di costruzione del dataset degli interventi in sedute della Camera. Nello specifico quella mostrata è la sezione adibita al web scraping automatizzato per il recupero della trascrizione e del file video.

è avvenuto il suddetto intervento.

Ottenuto così il giusto riferimento, lo script accede alla risorsa web utilizzando la libreria `Selenium`. Questa è una comoda utility per il web scraping,

che permette il controllo automatizzato di un browser web, potendo simulare le interazioni dell'utente, e consente inoltre l'accesso diretto al Document Object Model (DOM) della pagina, potendo in questo modo selezionare ed interagire con gli elementi HTML che la compongono. Il nostro algoritmo è quindi in grado di aprire uno dei browser installati sulla macchina di esecuzione, per poi visitare ed accedere alla pagina del portale WebTV della Camera contenete l'intervento in elaborazione, come fosse un normale utente umano. Successivamente, accedendo al DOM della pagina, lo script cerca al suo interno il paragrafo HTML che contiene la trascrizione, recuperabile anch'esso per mezzo degli stessi ID univoci identificativi dell'intervento, e ne salva il testo. Nella medesima pagina è possibile ricavare anche l'ora esatta in cui l'intervento è iniziato, dato non presente nei dati recuperati dall'OCD.

Per scaricare invece la clip video, il nostro algoritmo utilizza nuovamente `Selenium` al fine di individuare all'interno della pagina il tasto di download associato all'intervento in elaborazione, anch'esso etichettato dai soiliti ID specifici. Lo script ne ricava poi il link che verrebbe visitato al suo click ed infine utilizza nuovamente il browser per accedere a questo link. Viene in questo modo avviato il download della clip video e lo script Python si occuperà ora di attendere il termine dell'operazione, rinominare opportunamente e spostare il file in una cartella di raccolta generale. La Figura 3.10 mostra un estratto del codice del nostro script, precisamente la sezione che implementa il recupero della trascrizione ed il download della clip video.

interventi totali	**370**
	Numero di Token
mean	137
std	108
min	4
25%	55
50%	121
75%	187
max	685

Tabella 3.2: Statistiche descrittive delle lunghezze (in token) del corpus di trascrizioni degli interventi in sedute della Camera utilizzati in ParlaMentis.

Al termine di tutto il processo di recupero, filtraggio ed elaborazione degli interventi nelle sedute della Camera dei Deputati, abbiamo ottenuto un dataset (archiviato in un apposito file in formato JSON) contenete in totale 370 record. Ricordiamo che

per ogni intervento si è tenuto traccia del suo ID univoco, del nome e cognome dell'oratore, del partito di appartenenza e dell'eventuale incarico governativo ricoperto dall'oratore (al tempo del dato intervento), dell'argomento al quale l'intervento fa riferimento, della data e ora in cui si è tenuto l'intervento, della legislatura nella quale si è tenuto l'intervento, della trascrizione sotto forma di grezza stringa di testo delle parole pronunciate nell'intervento, ed infine della clip video relativa al record.
La Tabella 3.2 riporta una serie di statistiche riguardanti le lunghezze delle trascrizioni degli interventi presenti nel nostro dataset finale. La Figura 3.6 mostra con un istogramma la distribuzione di queste lunghezze. La lunghezza della singola trascrizione viene calcolata sul numero dei token di cui è composta.
La Figura 3.12 mostra in un istogramma i tempi di costruzione del dataset, di cui lo script Python si è occupato di tenere traccia. Nello specifico riportiamo i tempi del solo download delle clip video e i tempi totali di elaborazione del singolo intervento (comprensivi di download del file ed estrapolazione della trascrizione). I dati riportati riguardano un sottoinsieme significativo di 100 eventi presi dal nostro dataset.

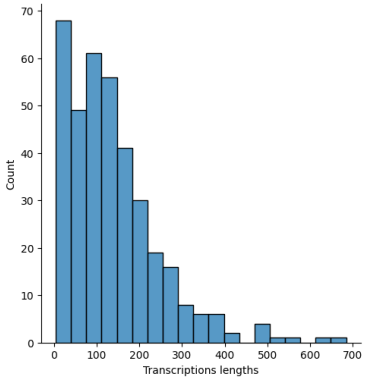

Figura 3.11: Distribuzione delle lunghezze (in token) delle trascrizioni degli interventi recuperati.

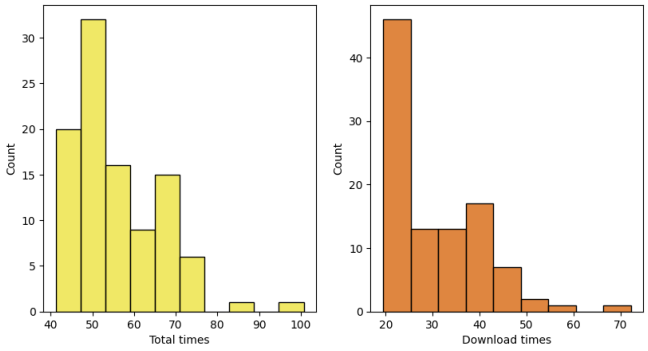

Figura 3.12: Distribuzione dei tempi di costruzione del dataset: a destra i tempi del solo download delle clip video, a sinistra i tempi totali di elaborazione del singolo interventp (comprensivi di download del file ed estrapolazione della trascrizione).

3.2.2 Dataset per l'Addestramento

In questa sezione discuteremo dei dataset sui quali abbiamo basato l'addestramento del Large Language Model integrato all'interno di ParlaMentis. In particolare facciamo distinzione tra i dataset utilizzati per adattare il modello alla lingua italiana, dai dataset utilizzati invece per allenare il modello alla gestione di dati di tipo legale, quindi strettamente connessi allo specifico dominio applicativo.

Padronanza della Lingua Italiana

Il dataset in questione è composto per l'80% da dati esclusivamente per la lingua italiana, mentre il restante 20% è stato riservato per la lingua inglese. Tale scelta è supportata dalle raccomandazioni di Ibrahim et al. [22], i quali suggeriscono che l'inclusione di una modesta frazione dei dati originali di pretraining durane un successivo ri-addestramento possa mitigare il rischio di *catastrophic forgetting*. Questo è un fenomeno comune nelle reti neurali, in cui un modello addestrato su un nuovo set di dati tende a dimenticare le informazioni apprese in precedenza, portando alla loro sovrascrittura con le nuove informazioni. Il nostro bilanciamento 80-20 permette quindi al modello di preservare le competenze acquisite precedentemente e trasferirle con maggior efficacia alla nuova lingua.

In riferimento alla lingua italiana generica (agnostica rispetto al dominio), sono state identificate due fonti:

- **Articoli di Wikipedia**[5] **(40%)**: articoli in lingua italiana provenienti da Wikipedia, accuratamente sottoposti a pulizia.
- **La Gazzetta Ufficiale - Serie Generale**[6] **(5%)**: atti emanati dalle Amministrazioni centrali e periferiche dello Stato, caratterizzati da un linguaggio legale di alto livello.

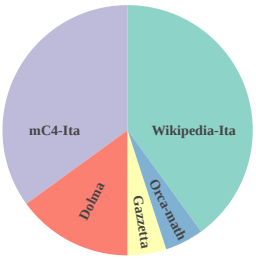

Figura 3.13: Distribuzione dei dataset per il pretraining di ParlaMentis per la lingua italiana.

[5] https://huggingface.co/datasets/wikipedia
[6] https://huggingface.co/datasets/mii-llm/gazzetta-ufficiale

- **mC4**[7] **(35%)**: documenti italiani presenti all'interno di mC4, corpus multi-lingua ottenuto mediante *web crawling*.

Per la lingua inglese invece:

- **dolma**[8] **(15%)**: contenuti web, pubblicazioni accademiche, codice sorgente, libri e materiali enciclopedici.
- **orca-math-word-problems-200k**[9] **(5%)**: problemi di matematica di scuola elementare.

Competenze Legali

Il ruolo dei dati utilizzati per l'addestramento riveste una fondamentale importanza nel processo di addestramento di modelli destinati al contesto legale, determinando non solo il materiale essenziale per l'apprendimento, ma anche la definizione del comportamento del modello stesso. I dati sono determinanti nell'interpretare con precisione le leggi e le sentenze, nonché nell'applicare propriamente il linguaggio giuridico. Un corpus ampio e ben strutturato agevola il processo di apprendimento del modello, consentendogli di assimilare le specificità linguistiche proprie del diritto italiano, comprese le terminologie tecniche e le espressioni idiomatiche.

Un'analisi esaustiva della letteratura scientifica [23] dimostra che l'inglese costituisce inequivocabilmente il linguaggio predominante nel campo della NLP legale, con una prevalenza del 56%, seguito dal cinese (~10%).

La digitalizzazione dei corpora legali italiani è cruciale nell'addestramento dei modelli, garantendo l'utilizzo di un lessico specifico e la conformità alle impostazioni desiderate. Senza un corpus adeguato, i modelli potrebbero non cogliere le sfumature del linguaggio giuridico, compromettendo la loro affidabilità e utilità.

Tuttavia, nonostante contributi preliminari (es. La Gazzetta Ufficiale, menzionata in precedenza), persiste una significativa carenza di risorse digitalizzate

[7] https://huggingface.co/datasets/gsarti/clean_mc4_it

[8] https://huggingface.co/datasets/allenai/dolma

[9] https://huggingface.co/datasets/microsoft/orca-math-word-problems-200k

nel dominio legale italiano. Sebbene siano stati sviluppati nuovi portali per l'accesso a corpora legali, questi sono spesso limitati alla consultazione Web.

Per queste ragioni, abbiamo intrapreso un'importante opera di digitalizzazione dei testi legali italiani. Tale processo è iniziato con operazioni di scraping orientate alla raccolta estensiva di dati, e si è concretizzato nel recupero delle seguenti risorse:

- Codice degli Appalti D.Lgs. 36/2023.
- Codice Civile, aggiornato a Dicembre 2023.
- Codice Penale, aggiornato a Marzo 2024.
- Linked Open Data della Camera dei Deputati (dei quali abbiamo discusso nella precedente Sezione 3.2.1).

Successivamente, è stata applicata ai dataset una fase di parsing per estrarre le informazioni rilevanti e conferire maggiore struttura alle istanze, rispettando la gerarchia e l'organizzazione delle fonti. Ciò ha richiesto un'attenta analisi manuale delle tipologie di documento presenti in ogni fonte, facendo attenzione a preservare struttura logica e giuridica (es. divisione in articoli, sezioni, commi, paragrafi).

Infine, si è vista necessaria un'ultima fase di elaborazione dei dataset legali, al fine di avere a disposizione un corpus adeguato all'addestramento di un modello di tipo instruct. Un modello instruct differisce dalla sua versione base principalmente per la sua capacità di interpretare e rispondere a istruzioni specifiche, come ad esempio quelle che possono essere delle normali richieste utente, formulate ovviamente in linguaggio naturale. I modelli standard vengono generalmente addestrati su una vasta gamma di testi per generare risposte coerenti e contestualmente appropriate. Tuttavia, questi modelli sono ottimizzati semplicemente per la comprensione del lessicale e per l'assorbimento di conoscienza, e non per eseguire correttamente compiti specifici richiesti dagli utenti. Al contrario, i modelli instruct vengono addestrati con dati formattati come coppia istruzione-risposta, il che li rende capaci di comprendere meglio il contesto delle richieste utente e di fornire risposte più pertinenti e accurate quando applicati in task reali.

La scelta di utilizzare un modello instruct deriva quindi dal bisogno di garantire che il modello potesse gestire efficacemente compiti specifici e dettagliati, tipici del linguaggio giuridico. Questo è poi particolarmente rilevante nel nostro caso specifico, volendo noi utilizzare il modello all'interno di una architettura Multi-Document Agent (Sezione 3.1). Questa combina la formulazione di "pensieri" in linguaggio naturale con successive "azioni" di recupero delle informazioni rilevanti, richiedendo perciò un modello capace di seguire istruzioni complesse, cosa che i modelli standard potrebbero non fare efficacemente.

Questa fase di trasformazione dei dataset ha comportato quindi la conversione dei testi legali grezzi in formati strutturati come coppie domanda-risposta, termine-descrizione e testo-riassunto. Questa trasformazione è stata realizzata utilizzando un modello avanzato, Mixtral-7x8B[10], con 46 miliardi di parametri, per garantire l'alta qualità dei dati trasformati. L'uso di un modello di questa scala ha infatti assicurato che la trasformazione dei dati mantenesse un elevato livello di accuratezza e aderenza al linguaggio giuridico, minimizzando le imprecisioni e le ambiguità. Questo ha permesso di creare dataset che rispecchiano fedelmente le modalità di utilizzo previste, consentendo al modello di apprendere non solo le risposte specifiche, ma anche il contesto e la modalità di risposta più appropriati per ogni tipo di istruzione. Questo approccio, noto come instruction tuning, migliora significativamente le capacità del modello di generalizzare le risposte a istruzioni non viste durante l'addestramento, rendendolo più versatile e affidabile per applicazioni pratiche. La Tabella 3.3 illustra il prompt utilizzato ed i risultati ottenuti in questa fase.

Quelli appena descritti sono dataset che considerimao perciò contenenti "gold answer", proprio perché le risposte, le descrizioni o i riassunti sono stati prodotti con un modello molto grande ed avanzato. Abbiamo quindi costruito un ulteriore dataset contenente in questo caso le rispettive "rejected answer" delle stesse istruzioni (domanda, termine da descrivere, testo da riassumere). Questo dataset è stato creato utilizzando dunque gli stessi prompt usati in precedenza per Mixtral-7x8B, così da avere le rispettive controparti delle gold answer, ma con un modello con molti meno parametri e molto più vecchio, Mistral-7B-v1.0. La logica alla base di questa scelta è che la vetustà e la

[10] https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

Prompt

```
You are an Italian legal expert model. Given the following document text,
generate some possible questions ad the related answers, extract the main
concepts and their definitions, and genrate a summary.

{{context text}}
```

Dataset Generati

```
'question': "Che cos'è la capacità giuridica?"
'answer': 'La capacità giuridica è la possibilità di essere titolare di
diritti e di assumere obblighi. Si acquista dal momento della nascita.'
```

```
'term': 'Usufrutto'
'definition': 'Un diritto reale che permette ad una persona di godere e
disporre di un bene, senza esserne il proprietario, a condizione di non
mutarne la sostanza e di non ledere il diritto di proprietà del titolare.'
```

```
'term': 'Obbligazioni'
'definition': 'Titoli di debito emessi da una società che attribuiscono
al sottoscrittore il diritto al rimborso del capitale e al pagamento
degli interessi.'
```

Tabella 3.3: La tabella mostra il prompt utilizzato per la generazione dei dataset instruct ed un piccolo estratto dei risultati ottenuti.

limitata capacità di quest'ultimo dovrebbero produrre risposte molto meno accurate o errate.

Ulteriore nota a margine va fatta per il dataset relativo ai Linked Open Data della Camera dei Deputati, questo poiché ha subito un processo di rielaborazione leggermente diverso. In questo caso infatti è stato costruito solo il dataset delle "gold answer", andando a selezionare manualmente 5 tra i gli atti di iniziativa precedentemente recuperati, e successivamente dando in pasto al modello GPT-4o il relativo file PDF contenente il testo dell'atto. Per ogni atto abbiamo chiesto all'LLM di generare 20 domande pertinenti e le rispettive risposte. Ottenendo quindi alla fine un totale di 100 coppie domande-risposte, utilizzate anch'esse insieme ai precedentemente descritti dataset in fase di addestramento del nostro modello, ma impiegate anche e soprattuto durante la fase di valutazione (Capitolo 4).

3.3 Large Language Model

3.3.1 Scelta del Modello

LLaMA-3 è la più recente famiglia di LLM open-source sviluppata da Meta. Tali modelli, disponibili nelle varianti da 8 e 70 miliardi di parametri, sono addestrati su un vasto corpus di testo comprendente 15 trilioni di token. Un volume di dati significativamente superiore a quello impiegato per l'addestramento di LLaMA-2 nel 2023, limitato a 2 trilioni. I modelli LLaMA-3 presentano inoltre una lunghezza del contesto aumentata fino a 8.192 token, in confronto ai 4.096 token di LLaMA-2, con una capacità di scalare fino a 32.000 token, il che li rende particolarmente idonei per applicazioni di Retrieval-Augmented Generation (RAG).

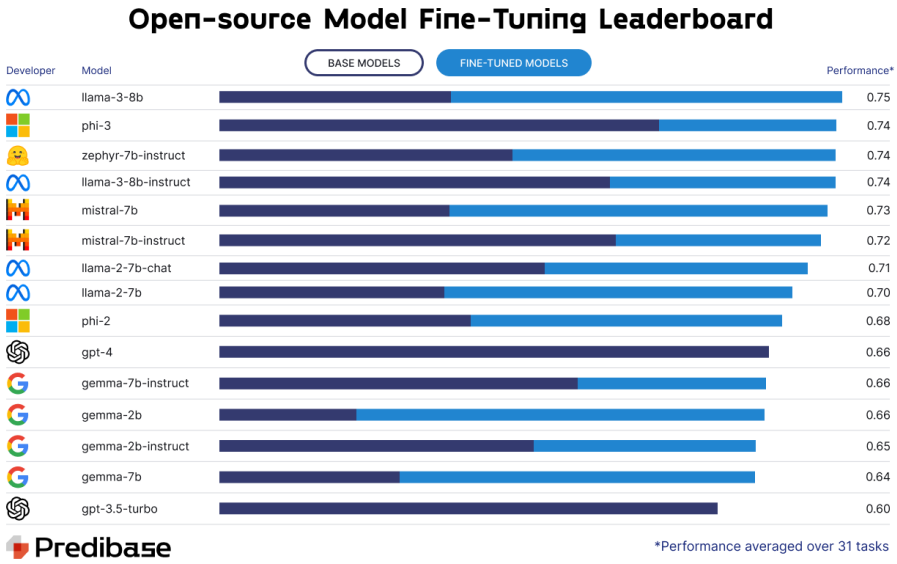

Figura 3.14: Prestazioni pre e post fine-tuning di LLM popolari, aggregate su 31 task NLP eterogenei.
Fonte: `https://predibase.com/fine-tuning-index`

Presentano un vocabolario ampliato a 128.000 token, riducendo del 15% il numero di token necessari per la codifica del testo. Tale espansione giustifica l'incremento da 7 miliardi a 8 miliardi di parametri per la versione minore. Di notevole rilevanza è il fatto che LLaMA-3 sia concepito per applicazioni

multi-lingua: oltre il 5% del dataset di preaddestramento è costituito da dati di alta qualità in lingue diverse dall'inglese, coprendo più di 30 idiomi. Di conseguenza, il modello possiede già una solida conoscenza della lingua italiana, suscettibile di ulteriore affinamento per usi specifici di dominio.

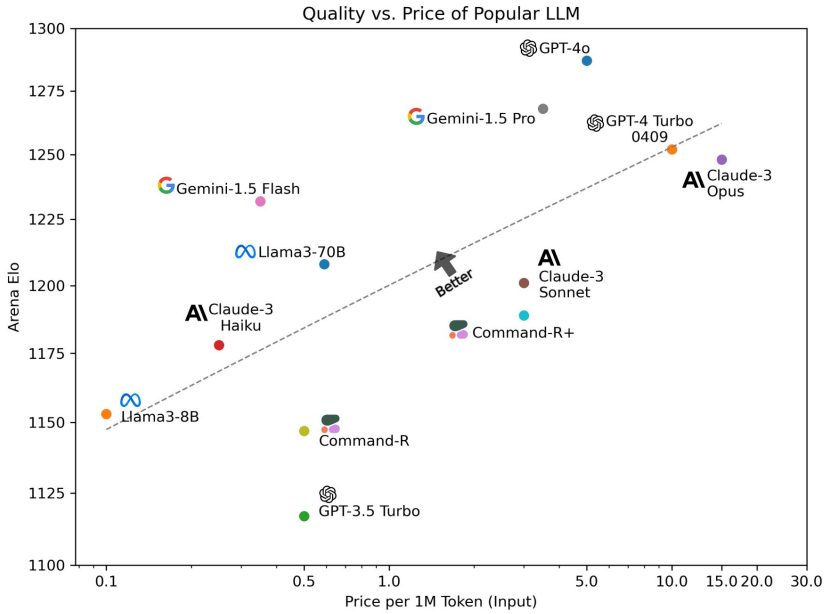

Figura 3.15: Punteggio Arena Elo di LLM popolari a seguito di >800.000 annotazioni umane di preferenza binaria con prompt eterogenei raccolti via crowdsourcing, rapportato ai costi segnalati nelle pagine dei provider.
Fonte: `https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard`

Il modello con 8 miliardi di parametri (LLaMA-3 8B) si è rivelato essere la scelta ottimale rispetto ad altri modelli della medesima dimensione, divenendo la nostra base per ParlaMentis. Recentemente, Predibase ha introdotto il Fine-tuning Index [24], progettato per assistere i team di IA aziendali nella selezione dei modelli open-source più adatti alle loro esigenze specifiche. Esso valuta le prestazioni di 13 popolari LLM open-source su 31 differenti task, comparandoli con i principali modelli commerciali. I modelli open-source ottimizzati tramite fine-tuning risultano non soltanto più economici di modelli closed-source con piani di pagamento a consumo, ma più celeri da addestrare ed implementare. Essi si distinguono anche per l'eccellenza in compiti specializzati,

quali la revisione di contratti legali, superando GPT-4 nell'85% dei benchmark considerati. LLaMA-3 8B, in particolare, si configura come il modello open-source ad oggi più performante per il fine-tuning (Figura 3.14). A seguito di un'estensiva valutazione umana [25], LLaMA-3 si dimostra inoltre essere un LLM conversazionale con un rapporto qualità/costi altamente competitivo (Figura 3.15).

3.3.2 Addestramento

In questa sezione discutiamo del lavoro di addestramento di nostri modelli nello specifico dominio applicativo. Nello specifico questa fase si è composta da due pipeline distinte che hanno rispettivamente portato una alla costruzione di due modelli base, mentre l'altra di due modelli instruct.

Addestramento dei modelli base

La prima pipeline di allenamento dei modelli base si compone di una iniziale fase di Continual Pre-Training (CPT), durante la quale il modello base viene addestrato alla next token prediction su di un normale testo non formattato. Questa fase ha lo scopo di far assimilare conoscenza all'LLM di quelle che sono le informazioni di dominio e le specificità di linguaggio. Successivamente lo stesso modello viene nuovamente allenato con la tecnica del Supervised FineTuning (SFT), questa volta allo scopo di adattare il modello al contesto applicativo e al formato delle istruzioni utente e delle rispettive risposte attese. Per questa seconda fase è quindi necessario il dataset per l'addestramento che abbiamo discusso nella Sezione 3.2.2. Quelle che ci servono sono coppie istruzione-risposta sulle quale eseguire sempre una next token prediction, ma questa volta solo sulla risposta e fornendo la domanda con rispettivo input.

Il processo ha visto l'addestramento del modello Meta-Llama-3-8B con tecnica CPT e di seguito SFT, ed ha infine prodotto due modelli personalizzati che differiscono per base di conoscenza utilizzata in allenamento:

- **ItalianLegalExpert-Llama3-8B**: è stato allenato con tecnica CPT sui dataset di addestramento di testo non formattato per la lingua italiana e

per il contesto legale, per la seguente fase SFT sono invece stati utilizzati i dati in formato istruzione-risposta inerenti il solo generico contesto legale.

- **ItalianLegalExpert-CuriaParlaMentis**: ha subito il medesimo allenamento del precedente modello, l'unica differenza sta nel fatto che la fase di addestramento con SFT è stata fatta sul dataset di domande-risposte estrapolate direttamente dagli atti di iniziativa della Camera.

Addestramento dei modelli instruct

Tradizionalmente, l'adattamento di LLM a task di dominio specifico si sviluppa in: (1) fine-tuning supervisionato (SFT) sulle istruzioni per adattare il modello al dominio di destinazione, seguito da (2) metodi di allineamento delle preferenze, come il Reinforcement Learning with Human Feedback (RLHF) o la Direct Preference Optimization (DPO), mirati a favorire la generazione di risposte coerenti con le preferenze umane. Sebbene SFT adatti efficacemente il modello al dominio desiderato, esso comporta anche un incremento non intenzionale della probabilità di predire risposte indesiderate oltre a quelle preferite. Per questo motivo, la fase di allineamento delle preferenze è necessaria per ampliare il divario tra le probabilità di output desiderati e non. Introdotto da Hong et al. [26], ORPO offre una elegante soluzione a tale problematica, combinando SFT e allineamento delle preferenze in un unico processo di addestramento monolitico (Figura 3.16). Mentre le tecniche sopra citate richiedono un ampia e onerosa fase di raccolta ed etichettature dati per determinare la qualità di un'istanza in termini di preferenza, l'algoritmo ORPO sfrutta la differenza relativa tra un esempio "negativo" (o "rifiutato") ed uno "positivo" (o "accettato"). Una tale divisione binaria dei dati risulta essere molto più efficiente rispetto ad una fase di raccolta che prevede l'assegnazione di punteggi.

La decisione di adottare ORPO come tecnica di fine-tuning è stata guidata da due principali motivazioni: la riduzione delle risorse computazionali e del tempo di addestramento necessari, e risultati empirici che attestano la superiorità di ORPO rispetto ad altri metodi di allineamento su diversi benchmark.

Contrariamente alle tecniche di preaddestramento che possono avvalersi di dati grezzi, come interi libri di testo, gli algoritmi basati su coppie istruzione-

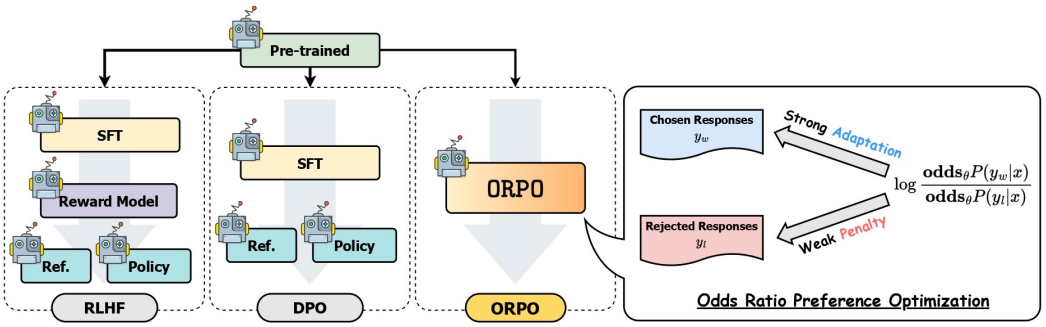

Figura 3.16: Il processo di allenamento standard di un modello linguistico comprende tre fasi: due di addestramento con dati etichettati e una per allineare lo stile delle risposte con un modello di riferimento. ORPO accorpa il fine-tuning e l'allineamento, senza usare un modello di reward.

risposta (Instruction Fine-tuning) o sull'allineamento alle preferenze degli utenti richiedono dati specificamente strutturati e formattati. A questo scopo abbiamo utilizzato i dataset per l'addestramento, discussi nella Sezione 3.2.2, che abbiamo precedentemente costruito utilizzando modelli di linguaggio. Questi sono costituiti da una serie di triplette della forma instruzione-gold answer-rejected answer

La funzione di ottimizzazione di ORPO (presentata in Equazione 3.1) è composta da due parti: una loss di fine-tuning supervisionato ($\mathcal{L}_{SFT}$) e una loss di rapporto tra le entità positive e negative ($\mathcal{L}_{OR}$), ponderate con λ.

$$\mathcal{L}_{ORPO} = \mathbb{E}_{(x,y_w,y_l)} \left[\mathcal{L}_{SFT} + \lambda \cdot \mathcal{L}_{OR} \right] \tag{3.1}$$

$\mathcal{L}_{SFT}$ segue la modellazione convenzionale di causal language modeling per generare il token successivo della sequenza dati i token precedenti. $\mathcal{L}_{OR}$ (rappresentata in Equazione 3.2) massimizza invece il rapporto tra la probabilità di generare la rispota etichettata come positiva y_w rispetto alla risposta negativa y_l. La funzione log-sigmoide è dunque utilizzata per la minimizzazione di $\mathcal{L}_{OR}$ mediante l'aumento del rapporto delle probabilità logaritmiche tra y_w e y_l.

$$\mathcal{L}_{OR} = -\log \sigma \left(\log \left(\frac{\text{odds}_\theta(y_w|x)}{\text{odds}_\theta(y_l|x)} \right) \right) \tag{3.2}$$

Il processo ha visto l'addestramento del modello Meta-Llama-3-8B-Instruct con tecnica ORPO, ed ha infine prodotto due modelli personalizzati che differiscono per base di conoscenza utilizzata in allenamento:

- **IalianLegalExpert-CuriaParlaMentis-ORPO**: è stato allenato sui dataset generati a partire dai testi legali generici, come Codice Civile e Penale. Il modello dovrebbe perciò aver appreso queste fonti ed essere in grado di districarsi tra le specificità linguistiche proprie del diritto italiano.

- **IalianLegalExpert-CuriaParlaMentis-ORPO-finale**: è stato anch'esso allenato sui dataset generati a partire dai testi legali generici, ma in questo caso anche sui dati generati dagli atti della Camera. Il modello, oltre ad avere la capacità di gestire testi legali italiani, dovrebbe essere quindi già allineato al formato specifico delle informazioni che recupererà dalla nostra base di retrieval e che dovrà utilizzare, promettendo una maggiore abilità di adattamento durante l'utilizzo reale.

3.4 Experimental Setup

3.4.1 Ambiente di Sviluppo

La fase di addestramento del nostro modello è stata condotta su una GPU NVIDIA A100 con 80GB di memoria vram. La versione del driver è 535.171.04 e la versione di CUDA è 12.2. Il sistema operativo di riferimento è Ubuntu 23.10. Il nostro ambiente di sviluppo è costruito sopra un container Docker utilizzando l'immagine nvidia/cuda:12.3.2-devel-ubuntu22.04

Ogni esperimento di inferenza e valutazione di ParlaMentis è stato condotto su una GPU NVIDIA GeForce RTX 3090 con 24GB di memoria vram. La versione del driver è 550.54.14 e la versione di CUDA è 12.4. Il sistema operativo di riferimento è Ubuntu 20.04.6 LTS. Di seguito riportiamo il dockerfile utilizzato per la costruzione dell'immagine Docker sopra al quale sono stati eseguiti tutti i test.

```
FROM huggingface/transformers-pytorch-latest-gpu

RUN pip install nest_asyncio
RUN pip3 install accelerate==0.28.0
RUN pip3 install peft==0.9.0

RUN pip3 install tqdm~=4.63.1 spacy~=3.3.0 nltk~=3.7 gensim~=4.2.0 tensorboard~=2.9.0
    protobuf~=3.19.0 scikit-learn~=1.1.1 seqeval~=1.2.2 datasets==2.11.0
    bitsandbytes==0.43.0 sentence-transformers==2.2.2

RUN pip3 install --upgrade nvidia-ml-py3==7.352.0
RUN pip3 install --upgrade wandb==0.15.7

RUN pip install torch==2.1.2

RUN pip install llama-index==0.9.48
RUN pip install langchain
RUN pip install pydantic==1.10.8
RUN pip install jinja2==3.1.3
RUN pip install ipywidgets
RUN pip3 install transformers==4.38.0 urllib3==1.26.7

RUN pip install rouge-score
RUN pip install bert-score
```

Figura 3.17: Dockerfile per la costruzione dell'immagine Docker di esecuzione dei test di ParlaMentis.

3.4.2 Iperparametri

Per garantire la replicabilità, vengono forniti tutti gli iperparametri. La Tabella 3.4 elenca tutti gli iperparametrici esaminati nelle fasi di inferenza e di addestramento del modello, evidenziando i valori selezionati.

Iperparametri	**Spazio di riceca**
inference	
`max_new_tokens`	{32, 64, 128∗, 256}
`max_input_size`	{2048, 4096∗}
`max_chunk_overlap`	{0∗, 0.1, 0.2}
`contex_window`	8192
`temperature`	{0∗, 0.1, 0.2}
`top_k`	{5, 10, 15∗, 20}
`max_iteration`	{30, 40, 60∗, 80}
`seed`	42
training	
`max_seq_length`	8192
`packing`	True
`per_device_train_batch_size`	10
`gradient_accumulation_steps`	4
`num_train_epochs`	1
`learning_rate`	3e-4
`fp16`	False
`bf16`	True
`logging_steps`	1
`optim`	adamw_torch_fused
`max_grad_norm`	1.0
`warmup_ratio`	0.1
`weight_decay`	0.1
`lr_scheduler_type`	cosine
`seed`	42
`save_steps`	10

Tabella 3.4: Iperparametri esplorati nello spazio di ricerca empirico.

∗ *etichetta il valore selezionato.*

3.4.3 Scelte Implementative

Vediamo ora con maggiore dettaglio come abbiamo applicato l'architettura Multi-Document Agent (Sezione 3.1) per implementare il nostro ParlaMentis.

Il modello è stato sviluppato in linguaggio Python utilizzando la utility `LlamaIndex`[11] v0.9.48. Questa è una libreria open-source progettata per facilitare lo sviluppo di applicazioni di intelligenza artificiale generativa basate su LLM, offrendo un robusto framework per la creazione di applicazioni che integrano l'elaborazione del linguaggio naturale con dati strutturati e non strutturati. Uno degli obiettivi principali di LlamaIndex è proprio la context augmentation, ossia l'abilità di arricchire i modelli linguistici con informazioni aggiuntive provenienti da varie fonti di dati.

Preprocessamento dei Dati e Costruzione dei Document Agent

Nella prima fase di esecuzione, l'algoritmo di occupa di preprocessare i dati per il retrieval e di costruire i relativi *document agent*. I documenti vengono quindi recuperati dal dataset degli atti di iniziativa della Camera dei Deputati (Sezione 3.2.1) ed ognuno incapsulato con i rispettivi metadati in un nodo `Document`. Ogni nodo viene successivamente suddiviso in chunk più piccoli da un oggetto della classe `SentenceSplitter`. Come dice il nome essa si occupa di dividere il nodo in input in più sezioni da dimensione e overlap specificato, evitando però di frammentare le singole frasi.

I segmenti del documento vengono poi utilizzati nella costruzione del rispettivo agente. Quest'ultimo infatti utilizza al suo interno due tipi di database vettoriali, entrambi popolati con i nodi dell'atto. Si tratta di un `VectorIndex`, per il recupero di informazioni puntuali sull'atto, e di un `SummaryIndex`, per il recupero di informazioni più generali e riassuntive. È proprio in questa fase che viene estratto il riassunto in linguaggio naturale del documento, che verrà successivamente utilizzato come descrizione identificativa dello stesso *document agent*. Questo riassunto viene generato con un apposito prompt al modello eseguendo retrieval direttamente sul `SummaryIndex`. Viene infine creato il sotto-

[11] https://docs.llamaindex.ai/en/v0.9.48/

agente ReAct appositamente configurato, andando anche a definire il suo compito di assistere nelle risposte a domande riguardanti il singolo documento che gli è stato assegnato. Come detto in precedenza, è preferibile archiviare su memoria di massa i chunk del documento ed il riassunto appena generato, per poi recuperare questi dati alle successive esecuzioni, con un significativo guadagno di efficienza. La Figura 3.18 mostra un estratto del codice appena descritto, la Tabella 3.5 integra il codice mostrando i prompt utilizzati.

```
    # ...
    vector_index = VectorStoreIndex(document_chunk, service_context=service_context)
    summary_index = SummaryIndex(document_chunk, service_context=service_context)

    vector_query_engine = vector_index.as_query_engine()
    summary_query_engine = summary_index.as_query_engine(
        response_mode="tree_summarize",
        use_async=True,
    )

    summary = str(await summary_query_engine.aquery("Summary prompt"))

    query_engine_tools = [
        QueryEngineTool(
            query_engine=vector_query_engine,
            metadata=ToolMetadata(
                name=f"vector_tool_{file_base}",
                description=f"Useful for questions related to specific facts",
            ),
        ),
        QueryEngineTool(
            query_engine=summary_query_engine,
            metadata=ToolMetadata(
                name=f"summary_tool_{file_base}",
                description=f"Useful for summarization questions",
            ),
        ),
    ]

    doc_agent = ReActAgent.from_tools(
        query_engine_tools,
        llm=service_context.llm,
        verbose=True,
        max_iterations=80,
        system_prompt=f"""System Prompt""",
    )
    # ...
```

Figura 3.18: Estratto di codice adibito alla costruzione dei *document agent*.

Summary Prompt

```
Extract from the first/second page of the document the name (consist of date and number)
of the inner law.
You have to specify also the general purpose of this parliamentary act.  You have to be
concise.
Do not add your own explanations or comments to the answer.
```

System Prompt

```
You are a specialized agent designed to answer queries about the {file_base} part of the
LlamaIndex docs.  You must ALWAYS use at least one of the tools provided when answering a
question; do NOT rely on prior knowledge.
Note:  when you work on a Chamber's documents you should read all the act's pages that
are related to the user question.  Within the pages of the documents the character "|" is
the column separator.  If a page contain the actual text of the law with the respective
articles, and that page is written in two columns, you should consider the two columns as
separeted mirrored text.  The left collumn ALWAYS contains ONLY the originally proposed
text of the law, while the right collumn ALWAYS contains ONLY the equivalent of the left
column but in the finally modified and approved version of the law.
```

Tabella 3.5: La tabella mostra i prompt utilizzati nella costruzione dei *document agent*.

Integrazione dei Video-Interventi della Camera

Per far si che il modello possa fare retrieval anche sulle video-registrazioni degli interventi in sedute della Camera (Sezione 3.2.1) abbiamo sfruttato l'alta flessibilità e possibilità di personalizzazione dell'architettura Multi-Document Agent. Abbiamo infatti considerato come se le trascrizioni degli interventi fossero riportate una di seguito all'altra all'interno di un nuovo documento fittizio. Ogni record accompagnato dai rispettivi metadati appositamente etichettati. Questo "resoconto" degli interventi è stato successivamente normalmente passato per la pipeline di preprocessamento, fino alla costruzione del nuovo *document agent* dedicato. L'unica differenza rispetto alla normale procedura di creazione degli agenti associati agli atti sta nel fatto che, in questo caso, non viene eseguita la fase di generazione del relativo riassunto identificativo da parte del modello. Al contrario, la descrizione del tool è stata scritta e assegnata manualmente, esplicando proprio il fatto che il sotto-agente contiene le informazioni relative a tutti gli interventi che si sono tenuti durante sedute

della Camera dei Deputati italiana. Secondo la nostra soluzione quindi, quando il *top-level agent* dell'architettura dovrà rispondere ad una domanda che si riferisce a questi interventi, esso troverà, nella totalità dei suoi tool disponibili per il recupero di informazioni esterne, quell'unico *document agent* che potrà soddisfare la sua richiesta.

Costruzione del *Top-Level Agent*

Creato il pool di *document agent* contenenti tutto il dataset di informazioni, l'algoritmo procede con la costruzione del *top-level agent*. Come vediamo nella Figura 3.19 esso non vi accede direttamente, ma al suo interno utilizza un tool retrieval custom che si occupa di recuperare le top-15 risorse maggiormente inerenti alla query dell'utente, rendendole poi accessibili per il retrieval da parte del top-agent. La Tabella 3.6 mostra il system prompt utilizzato nel *top-level agent*.

```
    # ...
    tool_mapping = SimpleToolNodeMapping.from_objects(document_agents)
    obj_index = ObjectIndex.from_objects(
        all_tools,
        tool_mapping,
        VectorStoreIndex,
        service_context=service_context,
    )

    custom_obj_retriever = CustomObjectRetriever(
        retriever=obj_index.as_node_retriever(similarity_top_k=15),
        object_node_mapping=tool_mapping,
        llm=service_context.llm,
        service_context=service_context,
    )

    top_agent = ReActAgent.from_tools(
        tool_retriever=custom_obj_retriever,
        system_prompt="""System Prompt.""",
        llm=service_context.llm,
        verbose=True,
        max_iterations=80,
    )
```

Figura 3.19: Estratto di codice adibito alla costruzione del *top-level agent*.

System Prompt
`You are an agent designed to answer queries about the documentation, using the italian language. You are a helpful, respectful and honest, helping the parlamentarians of the italian Chamber of Deputies. Your role is to support the work of preparing legislative proposals, and guidance and control acts.` `Your answers should not include any harmful, unethical, racist, sexist, toxic, dangerous, or illegal content. Please ensure that your responses are socially unbiased and positive in nature. If a question does not make any sense, or is not factually coherent, explain why instead of answering something not correct. If you don't know the answer to a question, please don't share false information.` `Always answer as helpfully as possible, while being safe. Make sure you always use the italian language.`

Tabella 3.6: La tabella mostra il system prompt utilizzato nella costruzione del *top-level agent*.

In Figura 3.20 mostriamo la nostra specifica implementazione del tool retriever. Questo conserva al suo interno tutti i *document agent* precedentemente creati. Espone il metodo `retrieve` grazie al quale il top-agent recupera i top-k tool più inerenti alla richiesta utente. Il `top-level` agent potrà quindi accedere ed utilizzare esclusivamente questi ultimi, limitando di fatto le risorse e quindi i contesti recuperabili.

La Figura 3.20 mostra inoltre come, in coda alla lista dei tool da restituire al `top-level`, il tool retriever si occupa anche di aggiungere un ulteriore strumento, quello che abbiamo chiamato il *query planning tool*. Questo è rappresentato da un oggetto della classe `SubQuestionQueryEngine`, appositamente progettato per la gestione di compiti che richiedono l'accesso simultaneo a multiple fonti di dati. Si occupa innanzitutto di scomporre la richiesta in sotto-task più piccoli e riferiti ad una singola risorsa, ottenere poi le risposte intermedie, per infine sintetizzarle nella risposta finale. Il *query planning tool* viene dinamicamente creato dal tool retriever ad ogni chiamata di metodo, facendo si che la sua base di conoscenza su cui fare retrieval siano proprio i top-k *document agent* recuperati. Necessita per ultima istanza di una apposita descrizione della risorsa, che abbiamo scritto manualmente definendo il suo compito di tool adibito esclusivamente a compiti di tipo riassuntivo e comparativo tra più documenti.

```python
class CustomObjectRetriever(ObjectRetriever):
    def __init__(...):
        #...

    def retrieve(self, query_bundle):
        nodes = self._retriever.retrieve(query_bundle)
        tools = [self._object_node_mapping.from_node(n.node) for n in nodes]
        self.retrieved_nodes = nodes

        sub_question_engine = SubQuestionQueryEngine.from_defaults(
            query_engine_tools=tools, service_context=self._service_context
        )
        sub_question_tool = QueryEngineTool(
            query_engine=sub_question_engine,
            metadata=ToolMetadata(
                name="compare_tool",


            ),
        )
        return tools + [sub_question_tool]
                ↪  for any queries involving multiple documents.""",
```

Figura 3.20: Estratto di codice che implementa il tool retriever ed il *query planning tool*.

3.5 Web App

Per una facile ed intuitiva interazione con il nostro strumento ParlaMentis, abbiamo sviluppato una dedicata interfaccia utente in stile chatbot. Si tratta di una web app che consente all'utente di interrogare l'agente come se stesse scrivendo in una normale chat di messaggistica. Il front-end dell'applicazione

è stato realizzato utilizzando REACT[12], un framework JavaScript ampiamente diffuso per la costruzione di interfacce utente interattive e dinamiche. La sua architettura si basa su un approccio dichiarativo, consentendo la definizione dell'aspetto e del comportamento dell'interfaccia in modo chiaro e conciso.
REACT è progettato per facilitare la creazione di componenti UI riutilizzabili e modulari, ciascuno responsabile di una parte specifica dell'interfaccia utente. Ad esempio, possiamo avere componenti dedicati alla barra di input del messaggio, alla visualizzazione della conversazione e ai pulsanti di controllo. Un componente REACT è una funzione o una classe che accetta input (noti come props) e restituisce un elemento REACT che descrive cosa dovrebbe apparire sullo schermo. Questi ultimi vengono implementati in JSX, un'estensione di sintassi che consente di scrivere elementi UI in un formato simile a HTML dentro JavaScript. Ogni componente può avere uno stato interno, è il cambiamento dello stato di un componente che provoca il re-rendering del componente stesso. Invece di manipolare direttamente il DOM, React gestisce un Virtual DOM, una rappresentazione in memoria del DOM reale, che permette di aggiornare solo le componenti che effettivamente vanno aggiornate. Quando lo stato di un componente cambia, REACT crea un nuovo albero del Virtual DOM e lo confronta con l'albero precedente (un processo noto come diffing). Vengono quindi calcolate le differenze, e il DOM reale viene aggiornato solo dove necessario. Questo processo ottimizza le operazioni di manipolazione del DOM, che possono essere costose in termini di prestazioni.

La nostra web app è stata realizzata come un chatbot, un'interfaccia che simula una conversazione con un altro utente umano. Questo tipo di interazione rende l'utilizzo dell'applicazione più intuitivo e accessibile, permettendo di interagire con il sistema attraverso comandi testuali e ricevendo risposte in tempo reale. La UI da noi costruita si compone di una ulteriore sezione laterale alla chat, che ha lo scopo di contenere le principali fonti utilizzate dal modello per generare una data risposta. PARLAMENTIS infatti allega alla sequenza di output i chunck di testo che ha recuperato dalla sua conoscenza e sui quali ha basato la sua elaborazione. Dai metadati di queste informazioni si può facilmente risalire alla fonte originale, atto o intervento parlamentare che sia.

[12] https://react.dev/

Quando l'interfaccia REACT riceve l'output del modello ad una richiesta utente, si occupa di aggiornare dinamicamente la chat facendovi apparire un messaggio di testo contenente il testo generato, ed al contempo popola la sezione laterale della UI con le fonti delle informazioni nella risposta. L'utente è quindi in grado, per ogni interazione con lo strumento, di visionare comodamente il testo contenuto nei file PDF o il file video degli interventi alla Camera maggiormente utilizzati e considerati dal sistema inerenti alla sua richiesta. Lo scopo è quello di permette all'utilizzatore di PARLAMENTIS una personale revisione manuale delle fonti di informazione, così da constatare in qualunque momento in prima persona se la conoscenza generata dal modello risulta o meno ancora e supportata da fatti reali, o se invece è frutto di allucinazioni o di retrieval di dati non pertinenti.

La Figura 3.21 mostra l'interfaccia implementata, in un esempio di utilizzo reale.

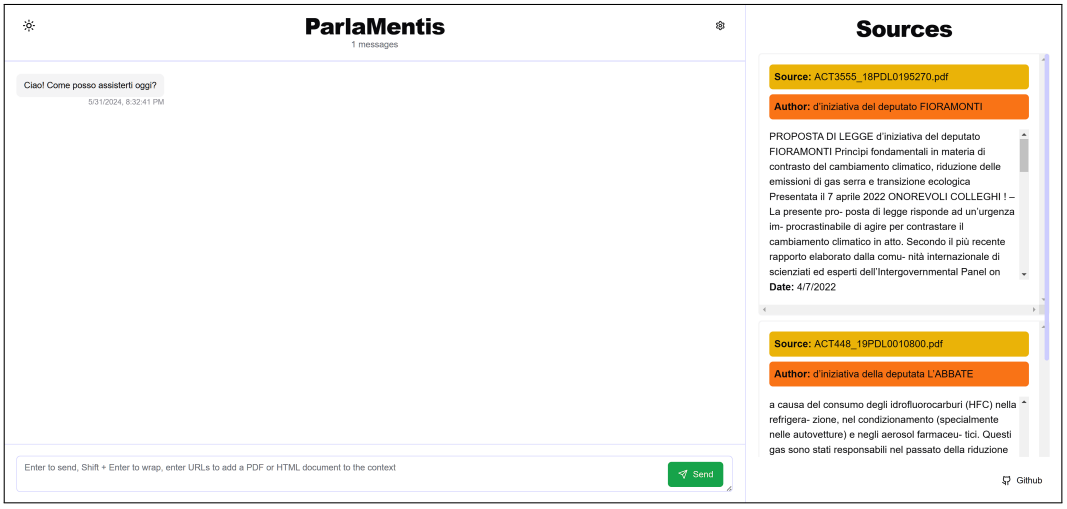

Figura 3.21: Interfaccia in stile chatbot realizzata per PARLAMENTIS, in un esempio di utilizzo reale.

Capitolo 4

Metriche e Valutazione del Modello

Nel seguente capitolo discutiamo dell'operazione di valutazione della qualità e delle prestazioni di ParlaMentis. Come già anticipato nella Sezione 3.2.2, a tale scopo abbiamo costruito un apposito dataset composto da 100 coppie domanda-risposta generate dal modello GPT-4o, prendendo come contesto 5 documenti manualmente selezionati tra i precedentemente recuperati atti di iniziativa della Camera. Il sistema è stato valutato sia in task puramente generativi, andando poi a confrontare la risposta generata con la risposta target per mezzo di metriche automatiche, che in task multiple choice.

Al fine di valutare selettivamente ogni contributo del nostro lavoro, procediamo con uno studio ablativo del sistema, mettendo alla prova il comportamento di tutta la famiglia di LLM da noi selezionati ed allenati sul corpus legale (sia instruct che non). Le prestazioni dei modelli sono state inoltre valutate sia nel caso di applicazione della nostra architettura Multi-Document Agent (Sezione 3.1) e sia nel caso di utilizzo della tradizionale architettura RAG, al fine di mettere in luce le differenze qualitative tra le due soluzioni.

Come vedremo in seguito, la valutazione dei modelli non instrcut è stata possibile esclusivamente se applicati alla tradizionale architettura RAG. La Multi-Document Agent richiede infatti elevate capacità di ragionamento, interpretazione e rielaborazione delle richieste utente, che poco si sposa con la natura dei modelli base. Questi sono infatti stati addestrati al solo fine di imparare la semantica e le relazioni di dipendenza intrinseche del linguaggio naturale, o al massimo per assorbire internamente la generale conoscenza disponibile

nel contesto applicativo. Non sono purtroppo in grado, poiché mai allenati a farlo, di soddisfare precise istruzioni e di elaborare una risposta coerente e ben allineata allo stile umano.

4.1 Metriche Automatiche

Una valutazione approfondita di ParlaMentis è essenziale per garantirne l'efficacia, efficienza e affidabilità in contesti reali. In tale prospettiva, in confronto alle strategie attualmente riportate in letteratura, occorre osservare che la capacità di generare testo artificiale indistinguibile da quello umano è oggi superiore rispetto alla capacità di valutarlo. Tuttavia, è imperativo riconoscere che la qualità del linguaggio generato è un concetto poliedrico, che richiede un'analisi su molteplici dimensioni intrinseche, quali informatività, fattualità, fluidità, coerenza e adeguatezza. Inoltre, possono esistere risposte diverse ma ugualmente plausibili o di pari significato per lo stesso input dell'utente.

Sebbene la valutazione umana sia considerata lo standard di riferimento, progettare esperimenti di crowdsourcing con linee guida dettagliate è un processo costoso e caratterizzato da elevata latenza, non facilmente integrabile in una pipeline di sviluppo quotidiana che richiede benchmarking e addestramento periodico su vasta scala. Con il progresso architetturale dei modelli generativi, i valutatori devono confrontarsi con passaggi di testo più estesi e complessi, condizionati da un ampio contesto. In questi scenari, gli errori sono spesso legati al contenuto semantico, come inesattezze fattuali o incoerenze contestuali [27, 28], piuttosto che alla fluidità, rendendo insufficienti valutazioni superficiali con annotatori inesperti [29].

Date queste problematiche, la comunità di ricerca NLP ha proposto una varietà di metriche di valutazione automatica, le quali computano un punteggio olistico o specifico per dimensione (G: generative, R: retrieval):

G **ROUGE** [30] $[0, 1], \uparrow$

Recall-Oriented Understudy for Gisting Evaluation è un insieme di metriche di valutazione utilizzate per misurare la sovrapposizione lessicale (%) tra

risposte risposte generate dal modello e risposte target. Le principali varianti di ROUGE includono:

- **ROUGE-N**: Misura il numero di n-grammi (sequenze contigue di parole) sovrapposti tra il testo generato e il testo di riferimento. Le varianti più comuni sono ROUGE-1 (unigrammi) e ROUGE-2 (bigrammi).

$$\text{ROUGE-N} = \frac{\sum_{S \in \text{RefSumm}} \sum_{\text{gram}_n \in S} \min(\text{Count}_{\text{match}}(\text{gram}_n), \text{Count}_{\text{cand}}(\text{gram}_n))}{\sum_{S \in \text{RefSumm}} \sum_{\text{gram}_n \in S} \text{Count}(\text{gram}_n)} \tag{4.1}$$

- **ROUGE-L**: Utilizza la Longest Common Subsequence (LCS), cioè la sotto-sequenza comune più lunga tra il testo generato e il testo di riferimento. Questo metodo tiene conto dell'ordine delle parole e della lunghezza delle corrispondenze.

$$\text{ROUGE-L} = \frac{LCS(\text{RefSumm}, \text{CandSumm})}{\text{Len}(\text{RefSumm})} \tag{4.2}$$

- **ROUGE-Lsum**: si tratta di una versione aggregata di ROUGE-L che viene utilizzata in contesti di riassunto dove ci sono più documenti di riferimento.

$$\text{ROUGE-Lsum} = \frac{\sum_{i=1}^{m} LCS(\text{RefSumm}_i, \text{CandSumm})}{\sum_{i=1}^{m} \text{Len}(\text{RefSumm}_i)} \tag{4.3}$$

  Dove $LCS(\text{RefSumm}_i, \text{CandSumm})$ è la lunghezza della sottosequenza comune più lunga tra il sommario di riferimento i-esimo e il sommario candidato, $\text{Len}(\text{RefSumm}_i)$ è la lunghezza del sommario di riferimento i-esimo, e m è il numero totale di sommari di riferimento.

G **BERTScore** [31] $[-1, 1], \uparrow$

Il BERTScore valuta l'allineamento semantico tra una sequenza di risposta generata e una sequenza target. Un modello rappresentazionale (transformer encoder-only) preaddestrato basato su BERT è impiegato per trasformare ogni token nelle sequenze di testo (sia nella risposta generata che nel target) nel rispettivo vettore di embedding, che catturando il contesto bidirezionale del token all'interno della frase. Per ogni token y_i nella sequenza target y, si calcola

la similarità (prodotto scalare) con il token $\hat{y}_j$ nella sequenza generata $\hat{y}$ che massimizza questa similarità: Questo passaggio è formulato come:l'allineamento (prodotto scalare) medio tra le rappresentazioni dei token nelle due sequenze, pesando i token rispetto a una stima della loro rilevanza:

$$\max_{\hat{y}_j \in \hat{\boldsymbol{y}}} \mathrm{e}(y_i)^{\mathsf{T}} \cdot \mathrm{e}(y_j) \tag{4.4}$$

La similarità massima tra $\hat{y}_j$ e $\hat{y}$ indica quanto bene il token generato corrisponde al token target.
Per attribuire maggiore importanza ai token meno frequenti e potenzialmente più informativi, il BERTScore utilizza la Inverse Document Frequency (IDF). La IDF di un token t è definita come:

$$\mathrm{idf}(t) = \log\left(\frac{N}{|d \in D : t \in d|}\right) \tag{4.5}$$

Qui, N è il numero totale di documenti nel corpus D, e $|d \in D : t \in d|$ è il numero di documenti in cui il token t appare. Questa pesatura assicura che i token comuni abbiano un peso minore rispetto a quelli rari.

$$BeS = \frac{\sum_{y_i \in \boldsymbol{y}} \mathrm{idf}(y_i) \cdot \max_{\hat{y}_j \in \hat{\boldsymbol{y}}} \mathrm{e}(y_i)^{\mathsf{T}} \cdot \mathrm{e}(\hat{y}_j)}{\sum_{y_i \in \boldsymbol{y}} \mathrm{idf}(y_i)} \tag{4.6}$$

G **Perplexity** [32] $[0, \infty[$, $\downarrow$

La Perplexity (PPL) è una misura comunemente utilizzata in linguistica computazionale e nell'addestramento dei modelli di linguaggio per valutare la naturalezza di un testo generato rispetto ai dati su cui il modello è stato addestrato. È particolarmente utile per quantificare quanto bene un modello probabilistico di linguaggio riesca a prevedere una sequenza di parole. La formula di calcolo è:

$$ppl(\boldsymbol{y}) = \exp\left(\frac{1}{|\boldsymbol{y}|} \sum_{i}^{|\boldsymbol{y}|} \log p_\theta(y_i | \boldsymbol{y}_{<i})\right) \tag{4.7}$$

dove $\boldsymbol{y}$ rappresenta il testo da valutare e $p_\theta(y_i|\boldsymbol{y}_{<i})$ è la probabilità prevista dal modello per il token y_i dati i token precedenti $\boldsymbol{y}_{<i}$.

La Perplexity può essere interpretata come il numero medio di scelte che il modello considera plausibili per la continuazione di una sequenza di parole.

G **BARTScore-F** [33] $[-\infty, 0], \uparrow$

Un modello BART [34] preaddestrato con pesi θ è utilizzato per stimare la probabilità che la risposta inferita e quella target siano parafrasi reciproche:

$$BaS = \sum_{t=1}^{|\boldsymbol{y}|} \log p(y_t | \boldsymbol{y}_{<t}, \hat{\boldsymbol{y}}, \theta) \tag{4.8}$$

RAGAS[1]

- R **Context Recall** $[0, 1], \uparrow$ Misura quanto il contesto recuperato sia allineato con la risposta target. Ogni frase nella risposta target viene analizzata per determinare se può essere attribuita al contesto recuperato o meno.

$$CR = \frac{|\text{Frasi target attribuibili al contesto}|}{|\text{Frasi target}|} \tag{4.9}$$

- R **Context Precision** $[0, 1], \uparrow$ Data una domanda, valuta quanti chunk target rilevanti vengono restituiti tra i primi risultati dal modulo di retrieval. Idealmente, tutti i chunk rilevanti devono comparire nei ranghi più alti.

$$CP@K = \frac{\sum_{k=1}^{K} P@k \cdot v_k}{|\text{Chunk rilevanti nei primi } K \text{ risultati}|} \tag{4.10}$$

$$P@k = \frac{\text{veri positivi}@k}{(\text{veri positivi}@k + \text{falsi positivi}@k)} \tag{4.11}$$

- G **Faithfulness** $[0, 1], \uparrow$ Esamina la coerenza fattuale della risposta generata rispetto al contesto fornito in termini deduttivi.

$$F = \frac{|\text{Affermazioni nella risposta generata inferibili dal contesto}|}{|\text{Affermazioni nella risposta generata}|} \tag{4.12}$$

[1] https://docs.ragas.io/en/stable/index.html

4.2 Analisi dei Risultati

4.2.1 Valutazione Multiple Choice

Una prima valutazione del modello è stata effettuata ponendo un test di tipo multiple choice. Il sistema è stato opportunamente interrogato per rispondere alle 100 domande contenute nel nostro dataset per la valutazione. In questo caso, non abbiamo però interrogato il sistema per generare una risposta completa, ma al contrario abbiamo fornito nel prompt della richiesta 4 possibili risposte numerate. Abbiamo quindi chiesto al modello di scegliere, tra le scelte disponibili, quella che secondo lui rappresenta la soluzione corretta per la domanda, e di riportare il rispettivo numero. Ci siamo assicurati di formulare un prompt che induca l'LLM a non rispondere andando per esclusione delle scelte meno pertinenti, ma che invece lo forzi a formulare la sua personale risposta, per poi confrontarla con quelle possibili e selezionare quella che più si avvicina.

Ovviamente tra le 4 risposte presentate solo una è quella effettivamente corretta, le altre sono state selezionate randomicamente sempre dal nostro dataset di valutazione. Al fine di garantire una maggiore affidabilità della valutazione, le 3 possibilità errate sono state estratte considerando il solo sotto-insieme di risposte che fanno riferimento al medesimo atto parlamentare. Questo per evitare che il modello riuscisse con troppa facilità ad individuare la scelta corretta nel caso in cui le restanti opzioni facessero riferimento ad argomenti e contesti totalmente differenti rispetto al tema della domanda. La Tabella 4.1 mostra un esempio dello User Promt utilizzato in questa fase di valutazione. Nella Tabella 4.2 sono visionabili tutti i risultati intermedi e finali ottenuti.

User Prompt

```
Answer the following multiple choice question:
Qual è l'obiettivo principale del decreto-legge 3555?

Choose the correct answer from the following:
1) Implementare strategie per il monitoraggio e il controllo delle emissioni di gas serra a livello
nazionale.
2) Introdurre misure urgenti in materia di contrasto del cambiamento climatico, delle emissioni di gas
serra, e transizione ecologica.
3) Stabilire l'obbligo per lo Stato di adottare misure atte a rispettare gli impegni internazionali
dell'Accordo di Parigi.
4) Coordinare le politiche ambientali e climatiche a livello nazionale.

You should ask yourself the question, generate your personal answer and finally select the one that
comes closest from the options.
You should answer ONLY indicating the number corresponding to your choice.
Do not rewrite the answer too.  Do not add explanations or introductions, only and exclusively the
number.
```

Tabella 4.1: Un esempio di User Prompt utilizzato nella valutazione Multiple Choice del nostro sistema. La possibile risposta evidenziata in arancione è quella effettivamente corretta per questo caso.

		Multiplechoice evaluation tests					
ARCHITECTURE	MODEL	**ACT_3555**	**ACT_611**	**ACT_69**	**ACT_1458**	**ACT_907**	**TOT.**
Simple RAG	**Meta-Llama-3-8B**	13 / 20	14 / 20	13 / 20	10 / 20	14 / 20	64 / 100
	ItalianLegalExpert-Llama-3-8B	16 / 20	15 / 20	14 / 20	8 / 20	11 / 20	64 / 100
	ItalianLegalExpert-CuriaParlaMentis	15 / 20	16 / 20	11 / 20	13 / 20	10 / 20	65 / 100
	Meta-Llama-3-8B-Instruct	15 / 20	15 / 20	10 / 20	13 / 20	12 / 20	65 / 100
	ItalianLegalExpert-CuriaParlaMentis-ORPO	10 / 20	15 / 20	15 / 20	12 / 20	14 / 20	66/ 100
	ItalianLegalExpert-CuriaParlaMentis-ORPO-final	17 / 20	16 / 20	14 / 20	13 / 20	12 / 20	72 / 100
Custom RAG	**Meta-Llama-3-8B-Instruct**	14 / 20	17 / 20	14 / 20	14 / 20	15 / 20	74 / 100
	ItalianLegalExpert-CuriaParlaMentis-ORPO	16 / 20	16 / 20	15 / 20	13 / 20	17 / 20	77 / 100
	ItalianLegalExpert-CuriaParlaMentis-ORPO-final	14 / 20	20 / 20	15 / 20	12 / 20	17 / 20	**78 / 100**

Tabella 4.2: Risultati ottenuti nella valutazione Multiple Choice del nostro sistema.

Si può notare come le prestazioni già raggiungibili dalla semplice architettura RAG utilizzando il modello non instruct e non allenato sul contesto legale rappresentino una buona base di partenza, raggiungendo una correttezza delle risposte selezionate che supera il 60%. Questo a testimonianza dell'elevata qualità del modello LLaMA-3 8B da noi scelto, il quale risulta capace di elevata flessibilità applicativa e di gestire senza difficoltà critiche il contesto legale italiano. Detto ciò, si riscontra un incremento prestazionale significativo apportato sia dalla messa in opera dei modelli sempre più allineati e dedicati allo specifico task e dominio informativo, ma soprattutto un incremento derivante dal supporto fornito da parte dell'architettura Multi-Document Agent in fase di retrieval ed inference. La nostra architettura permette ad esempio al modello Meta-Llama-3-8B-Instruct, che quindi non è appositamente progettato per il nostro specifico contesto e non ha al suo interno alcuna conoscenza legale italiana, di eguagliare se non superare i risultati ottenuti dal sistema operante su RAG tradizionale ma sfruttando la potenza del modello talianLegalExpert--CuriaParlaMentis-ORPO-final, completamente addestrato e personalizzato sul dominio di riferimento.

In conclusione, il test ablativo multiple choice ha riscontrato un incremento prestazionale, apportato dai nostri contributi architetturali e di addestramento dell'LLM ItalianLegalExpert- -CuriaParlaMentis-ORPO-final alla base, di circa il 22% rispetto al riferimento di partenza.

4.2.2 Valutazione Generativa

Una seconda valutazione del modello è stata effettuata ponendo un test generativo, che analizzi la qualità e la similarità delle risposte generate rispetto a delle "gold answer". Il sistema è stato opportunamente interrogato per rispondere alle 100 domande contenute nel nostro dataset per la valutazione. In questo caso, abbiamo chiesto al sistema di generare una risposta completa in linguaggio naturale, per poi confrontare quest'ultima con la rispettiva risposta presente nel nostro dataset per mezzo di metriche automatiche.

Ci siamo assicurati di formulare un prompt che induca l'LLM a generare testo conciso e preciso nella risposta, cercando di farlo somigliare in forma

quanto più possibile alle gold answer a nostra disposizione. Questo poiché le metriche automatiche di valutazione non possono ovviamente essere precise tanto quanto una valutazione manuale, offrendo un certo grado di approssimazione e incertezza, che possono portare a score di accuratezza inconsistenti se tra la sequenza generata e la sequenza target ci sono importanti disallineamenti non tanto contenutistici e informativi ma quando in temrini stile e di formattazione della risposta. La Tabella 4.3 mostra un esempio di User Promt utilizzato in questa fase di valutazione. Nelle Tabella 4.4, Tabella 4.5, e Tabella 4.5 sono visionabili tutti i risultati intermedi e finali ottenuti.

User Prompt

```
Answer the following question:
Qual è l'obiettivo principale del decreto-legge 3555?

Provide a dry and concise answer, without reintroducing the context of the question. Be very concise
and precise in the answer. Answer using a maximum of 15/20 words.
Make sure to answer in italian correct language.
```

Tabella 4.3: Esempio di User Prompt utilizzato nella valutazione generativa del nostro sistema.

Generative evaluation tests
SIMPLE RAG NOT INSTRUCT

MODEL	ACT_3555	ACT_611	ACT_69	ACT_1458	ACT_907	TOT.
Meta-Llama-3-8B	BERT-SCORE Precision: 0.4823 Recall: 0.6094 F1: 0.5247 ROUGE-SCORE rouge1: 0.1006 rouge2: 0.0243 rougeL: 0.0836 rougeLsum: 0.0943	BERT-SCORE Precision: 0.5066 Recall: 0.4543 F1: 0.4376 ROUGE-SCORE rouge1: 0.0776 rouge2: 0.0250 rougeL: 0.0708 rougeLsum: 0.0704	BERT-SCORE Precision: 0.4842 Recall: 0.4900 F1: 0.4575 ROUGE-SCORE rouge1: 0.1092 rouge2: 0.0351 rougeL: 0.0929 rougeLsum: 0.0902	BERT-SCORE Precision: 0.5166 Recall: 0.4056 F1: 0.4285 ROUGE-SCORE rouge1: 0.0924 rouge2: 0.0184 rougeL: 0.0591 rougeLsum: 0.0632	BERT-SCORE Precision: 0.5180 Recall: 0.5753 F1: 0.5285 ROUGE-SCORE rouge1: 0.1913 rouge2: 0.0728 rougeL: 0.1320 rougeLsum: 0.1614	BERT-SCORE Precision: 0.50154 Recall: 0.50692 F1: 0.47536 ROUGE-SCORE rouge1: 0.11422 rouge2: 0.03512 rougeL: 0.08767 rougeLsum: 0.09587
ItalianLegalExpert--Llama3-8B	BERT-SCORE Precision: 0.4169 Recall: 0.6219 F1: 0.4978 ROUGE-SCORE rouge1: 0.0579 rouge2: 0.0084 rougeL: 0.0531 rougeLsum: 0.0531	BERT-SCORE Precision: 0.5058 Recall: 0.6559 F1: 0.5661 ROUGE-SCORE rouge1: 0.1140 rouge2: 0.0631 rougeL: 0.1032 rougeLsum: 0.1065	BERT-SCORE Precision: 0.4773 Recall: 0.6250 F1: 0.5369 ROUGE-SCORE rouge1: 0.1169 rouge2: 0.0283 rougeL: 0.0813 rougeLsum: 0.0851	BERT-SCORE Precision: 0.5483 Recall: 0.6136 F1: 0.5768 ROUGE-SCORE rouge1: 0.2371 rouge2: 0.0754 rougeL: 0.1548 rougeLsum: 0.1558	BERT-SCORE Precision: 0.5086 Recall: 0.6279 F1: 0.5575 ROUGE-SCORE rouge1: 0.1593 rouge2: 0.0718 rougeL: 0.1214 rougeLsum: 0.1243	BERT-SCORE Precision: 0.49138 Recall: 0.62886 F1: 0.54702 ROUGE-SCORE rouge1: 0.13704 rouge2: 0.04939 rougeL: 0.10277 rougeLsum: 0.10495
ItalianLegalExpert--CuriaParlaMentis	BERT-SCORE Precision: 0.5000 Recall: 0.5996 F1: 0.5378 ROUGE-SCORE rouge1: 0.1059 rouge2: 0.0292 rougeL: 0.0918 rougeLsum: 0.0939	BERT-SCORE Precision: 0.5391 Recall: 0.6467 F1: 0.5790 ROUGE-SCORE rouge1: 0.2209 rouge2: 0.1358 rougeL: 0.2049 rougeLsum: 0.2074	BERT-SCORE Precision: 0.5705 Recall: 0.6327 F1: 0.5918 ROUGE-SCORE rouge1: 0.2148 rouge2: 0.0984 rougeL: 0.1766 rougeLsum: 0.1766	BERT-SCORE Precision: 0.6017 Recall: 0.6351 F1: 0.6171 ROUGE-SCORE rouge1: 0.2780 rouge2: 0.1246 rougeL: 0.1901 rougeLsum: 0.1886	BERT-SCORE Precision: 0.5962 Recall: 0.6442 F1: 0.6125 ROUGE-SCORE rouge1: 0.2007 rouge2: 0.1103 rougeL: 0.1721 rougeLsum: 0.1728	**BERT-SCORE** **Precision: 0.56150** **Recall: 0.63166** **F1: 0.58764** **ROUGE-SCORE** **rouge1: 0.20406** **rouge2: 0.09963** **rougeL: 0.16711** **rougeLsum: 0.16786**

Tabella 4.4: Risultati ottenuti nella valutazione generativa dell'architettura RAG utilizzando modelli non instruct.

Generative evaluation tests
SIMPLE RAG INSTRUCT

MODEL	ACT_3555	ACT_611	ACT_69	ACT_1458	ACT_907	TOT.
Meta-Llama-3-8B-Instruct	BERT-SCORE Precision: 0.6037 Recall: 0.6687 F1: 0.6331 ROUGE-SCORE rouge1: 0.1826 rouge2: 0.0567 rougeL: 0.1601 rougeLsum: 0.1601	BERT-SCORE Precision: 0.6710 Recall: 0.7181 F1: 0.6914 ROUGE-SCORE rouge1: 0.3344 rouge2: 0.2247 rougeL: 0.3016 rougeLsum: 0.3024	BERT-SCORE Precision: 0.5617 Recall: 0.6229 F1: 0.5857 ROUGE-SCORE rouge1: 0.2585 rouge2: 0.1481 rougeL: 0.2351 rougeLsum: 0.2172	BERT-SCORE Precision: 0.6297 Recall: 0.5804 F1: 0.6034 ROUGE-SCORE rouge1: 0.2720 rouge2: 0.1291 rougeL: 0.2042 rougeLsum: 0.1906	BERT-SCORE Precision: 0.6445 Recall: 0.6582 F1: 0.6481 ROUGE-SCORE rouge1: 0.3230 rouge2: 0.1863 rougeL: 0.2831 rougeLsum: 0.2766	BERT-SCORE Precision: 0.62212 Recall: 0.64966 F1: 0.63234 ROUGE-SCORE rouge1: 0.27409 rouge2: 0.14898 rougeL: 0.23682 rougeLsum: 0.22936
ItalianLegalExpert-CuriaParlaMentis-ORPO	BERT-SCORE Precision: 0.6166 Recall: 0.6770 F1: 0.6447 ROUGE-SCORE rouge1: 0.1996 rouge2: 0.0694 rougeL: 0.1841 rougeLsum: 0.1841	BERT-SCORE Precision: 0.6766 Recall: 0.6987 F1: 0.6860 ROUGE-SCORE rouge1: 0.2775 rouge2: 0.1492 rougeL: 0.2464 rougeLsum: 0.2464	BERT-SCORE Precision: 0.6597 Recall: 0.6672 F1: 0.6602 ROUGE-SCORE rouge1: 0.2834 rouge2: 0.1584 rougeL: 0.2385 rougeLsum: 0.2385	BERT-SCORE Precision: 0.6699 Recall: 0.5940 F1: 0.6292 ROUGE-SCORE rouge1: 0.2460 rouge2: 0.1201 rougeL: 0.2059 rougeLsum: 0.2059	BERT-SCORE Precision: 0.6744 Recall: 0.6605 F1: 0.6650 ROUGE-SCORE rouge1: 0.2685 rouge2: 0.1327 rougeL: 0.2178 rougeLsum: 0.2178	BERT-SCORE Precision: 0.65944 Recall: 0.65948 F1: 0.65702 ROUGE-SCORE rouge1: 0.25499 rouge2: 0.12597 rougeL: 0.21853 rougeLsum: 0.21853
ItalianLegalExpert-CuriaParlaMentis-ORPO-final	BERT-SCORE Precision: 0.6051 Recall: 0.6639 F1: 0.6326 ROUGE-SCORE rouge1: 0.1802 rouge2: 0.0692 rougeL: 0.1596 rougeLsum: 0.1596	BERT-SCORE Precision: 0.7125 Recall: 0.7189 F1: 0.7145 ROUGE-SCORE rouge1: 0.3528 rouge2: 0.2459 rougeL: 0.3172 rougeLsum: 0.3172	BERT-SCORE Precision: 0.6762 Recall: 0.6567 F1: 0.6632 ROUGE-SCORE rouge1: 0.2554 rouge2: 0.1479 rougeL: 0.2371 rougeLsum: 0.2371	BERT-SCORE Precision: 0.6773 Recall: 0.5914 F1: 0.6307 ROUGE-SCORE rouge1: 0.2593 rouge2: 0.1208 rougeL: 0.2069 rougeLsum: 0.2069	BERT-SCORE Precision: 0.6778 Recall: 0.6399 F1: 0.6566 ROUGE-SCORE rouge1: 0.2416 rouge2: 0.1115 rougeL: 0.2063 rougeLsum: 0.2063	**BERT-SCORE** **Precision: 0.66978** **Recall: 0.65416** **F1: 0.65954** **ROUGE-SCORE** **rouge1: 0.25786** **rouge2: 0.13905** **rougeL: 0.22542** **rougeLsum: 0.22542**

Generative evaluation tests
CUSTOM RAG INSTRUCT

MODEL	ACT_3555	ACT_611	ACT_69	ACT_1458	ACT_907	TOT.
Meta-Llama-3-8B-Instruct	BERT-SCORE Precision: 0.6638 Recall: 0.6951 F1: 0.6782 ROUGE-SCORE rouge1: 0.1716 rouge2: 0.0986 rougeL: 0.1571 rougeLsum: 0.1571	BERT-SCORE Precision: 0.6992 Recall: 0.6647 F1: 0.6792 ROUGE-SCORE rouge1: 0.2570 rouge2: 0.1175 rougeL: 0.2128 rougeLsum: 0.2128	BERT-SCORE Precision: 0.7307 Recall: 0.6602 F1: 0.6903 ROUGE-SCORE rouge1: 0.2629 rouge2: 0.1372 rougeL: 0.2283 rougeLsum: 0.2283	BERT-SCORE Precision: 0.7213 Recall: 0.6216 F1: 0.6659 ROUGE-SCORE rouge1: 0.2597 rouge2: 0.1249 rougeL: 0.2011 rougeLsum: 0.2011	BERT-SCORE Precision: 0.6936 Recall: 0.6545 F1: 0.6721 ROUGE-SCORE rouge1: 0.2144 rouge2: 0.1090 rougeL: 0.1839 rougeLsum: 0.1839	BERT-SCORE Precision: 0.70172 Recall: 0.65922 F1: 0.67714 ROUGE-SCORE rouge1: 0.23309 rouge2: 0.11743 rougeL: 0.19665 rougeLsum: 0.19665
ItalianLegalExpert-CuriaParlaMentis-ORPO	BERT-SCORE Precision: 0.6617 Recall: 0.7570 F1: 0.7037 ROUGE-SCORE rouge1: 0.2432 rouge2: 0.1299 rougeL: 0.2221 rougeLsum: 0.2221	BERT-SCORE Precision: 0.7468 Recall: 0.7206 F1: 0.7303 ROUGE-SCORE rouge1: 0.3128 rouge2: 0.1911 rougeL: 0.2765 rougeLsum: 0.2765	BERT-SCORE Precision: 0.6972 Recall: 0.6954 F1: 0.6936 ROUGE-SCORE rouge1: 0.3077 rouge2: 0.1771 rougeL: 0.2754 rougeLsum: 0.2754	BERT-SCORE Precision: 0.7206 Recall: 0.6329 F1: 0.6721 ROUGE-SCORE rouge1: 0.2627 rouge2: 0.1388 rougeL: 0.2188 rougeLsum: 0.2188	BERT-SCORE Precision: 0.7094 Recall: 0.6910 F1: 0.6963 ROUGE-SCORE rouge1: 0.2905 rouge2: 0.1474 rougeL: 0.2369 rougeLsum: 0.2376	BERT-SCORE Precision: 0.70714 Recall: 0.69938 F1: 0.69920 ROUGE-SCORE rouge1: 0.28338 rouge2: 0.15686 rougeL: 0.24595 rougeLsum: 0.24610
ItalianLegalExpert-CuriaParlaMentis-ORPO-final	BERT-SCORE Precision: 0.7500 Recall: 0.8056 F1: 0.7765 ROUGE-SCORE rouge1: 0.3531 rouge2: 0.1710 rougeL: 0.3265 rougeLsum: 0.3265	BERT-SCORE Precision: 0.7755 Recall: 0.7762 F1: 0.7752 ROUGE-SCORE rouge1: 0.3667 rouge2: 0.2146 rougeL: 0.3127 rougeLsum: 0.3127	BERT-SCORE Precision: 0.7578 Recall: 0.7605 F1: 0.7575 ROUGE-SCORE rouge1: 0.3427 rouge2: 0.1942 rougeL: 0.2813 rougeLsum: 0.2813	BERT-SCORE Precision: 0.7407 Recall: 0.6644 F1: 0.6998 ROUGE-SCORE rouge1: 0.2995 rouge2: 0.1555 rougeL: 0.2462 rougeLsum: 0.2439	BERT-SCORE Precision: 0.7549 Recall: 0.7268 F1: 0.7392 ROUGE-SCORE rouge1: 0.3241 rouge2: 0.1873 rougeL: 0.2772 rougeLsum: 0.2772	**BERT-SCORE** **Precision: 0.7558** **Recall: 0.7467** **F1: 0.7496** **ROUGE-SCORE** **rouge1: 0.3372** **rouge2: 0.1845** **rougeL: 0.2888** **rougeLsum: 0.2883**

Tabella 4.5: Risultati ottenuti nella valutazione generativa utilizzando modelli instruct, sia per l'originale architettura RAG (tabella sopra) e sia per la nostra architettura Multi-Document Agent (tabella sotto).

In questo caso teniamo maggiormente in considerazione per le nostre valutazioni i punteggi BERT-SCORE ottenuti dal sistema, poiché essenzialmente rappresentativi dell'allineamento semantico tra risposta generata e taget, contrariamente ai ROUGE-SCORE, i quali risultano relegati alla misura della pura sovrapposizione lessicale. Questo poiché è ragionevole aspettarsi che modelli differenti, come GTP-4o, utilizzato per generare le gold answer, e quelli da noi utilizzati per la valutazione, non forniscano risultati equiparabili in termini di semplice forma e stile.

I risultati ottenuti nel test generativo confermano quanto emerso dal precedente test multiple choice. Anche in questo caso, si può notare infatti come le prestazioni raggiungibili dalla semplice architettura RAG utilizzando il modello generale Meta-Llama-3-8B rappresentino una buona base di partenza. Si continua a riscontrare il significativo incremento prestazionale apportato sia dalla messa in opera dei modelli dedicati allo specifico dominio applicativo, ma soprattutto il contributo dell'architettura Multi-Document Agent. Anche in questo caso da sottolineare come la nostra architettura permette al modello non addestrato sulla conoscenza legale italiana di eguagliare se non superare i punteggi ottenuti dal sistema con RAG tradizionale ma sfruttando la potenza di un modello completamente allineato dominio di riferimento.

In conclusione, il test ablativo generativo ha riscontrato un incremento prestazionale, apportato dai nostri contributi architetturali e di addestramento dell'LLM ItalianLegalExpert- -CuriaParlaMentis-ORPO-final, di circa il 58% rispetto al riferimento di partenza. Per questi risultati va ancora una volta ricordato come le metriche automatiche di valutazione non sono esenti da errori di incertezza e vengono influenzate non dal solo contenuto informativo della risposta generata, ma in parte anche dalla rappresentazione lessicale di queste informazioni.

Conclusioni e Lavori Futuri

In conclusione, i risultati di ParlaMentis hanno dimostrato l'efficacia del metodo proposto, con performane competitive rispetto allo stato dell'arte. Questo strumento è stato progettato per assistere i deputati italiani nel processo legislativo. Il sistema si avvale delle potenzialità dei Large Language Model e integra tecniche di tecniche di IR all'interno del paradigma Reasoning and Acting, rendendo ParlaMentis capace di gestire e rispondere domande complesse che richiedono la selezione e l'accesso ad un vasto insieme di sorgenti informative diverse, in modo parallelo ed efficiente. Il nostro obiettivo principale era migliorare la gestione e l'accesso ai documenti legislativi da parte dei parlamentari della Camera italiana, riducendo il tempo necessario per la ricerca e l'analisi delle informazioni ed offrendo loro uno strumento capace di supportare con efficacia la fase di predisposizione degli atti di iniziativa.

I risultati ottenuti dimostrano che ParlaMentis è in grado di aumentare significativamente l'efficacia del lavoro legislativo, questo grazie all'integrazione sinergica di tecniche di ragionamento e azione che permettono al sistema ampia efficacia nel recupero e nell'analisi automatica di grandi volumi di documenti e di dati multimediali provenienti dal lavoro della Camera, mantenendo un elevato livello di accuratezza e coerenza.

Tuttavia, il lavoro svolto rappresenta solo un punto di partenza. Ci sono diverse direzioni future che possono essere esplorate per migliorare ulteriormente i contributi che abbiamo apportato. Innanzitutto il sistema potrebbe essere arricchito con l'utilizzo di fonti multimodali: la versione attuale di ParlMentis utilizza infatti le sole trascrizioni dei video-interventi avvenuti in sedute della Camera dei Deputati come base per il retrieval, implementando nei fatti un modello unimodale. In futuro sarebbe possibile dotare l'architettura di

strumenti di analisi diretta di audio e video delle intere registrazioni delle sedute d'Assemblea. Ciò consentirebbe al sistema non solo di essere indipendente da altri modelli di linguaggio, che oggi si vedono necessari nel caso in cui si voglia integrare file video di cui non si ha la trascrizione, ma soprattutto lo renderebbe capace di estrapolare informazioni che vanno al di là delle semplici parole degli oratori intervenuti nelle sedute. Sarebbe infatti in grado di interpretare anche le azioni compiute dai deputati, le reazioni dell'Assemblea, le espressioni, il tono di voce utilizzato, ed in generale tutto quello che è stato registrato nelle dirette della Camera.

Una possibile evoluzione della nostra architettura potrebbe prevedere l'integrazione di altri tipi di risorse esterne di informazioni. Multi-Document Agent descrive infatti una struttura molto flessibile ed altamente personalizzabile, cosa che abbiamo anche noi sfruttato per l'implementazione di ParlaMentis, in grado di accogliere con facilità nuovi strumenti di recupero dei dati, cosi come strumenti capaci di offrire nuovi servizi avanzati (es. ricerca di informazioni sul web). Portando all'estremo questo concetto sarebbe possibile integrare ulteriori moduli di ragionamento sulle richieste ed elaborazione del contesto recuperato, che fanno uso di LLM o architetture dedicate alla scomposizione delle domande utente e alla generazione di piani d'azione specifici per il contesto applicativo e per i dati a disposizione.

Sottolineiamo infine ancora una volta la necessità in campo di ricerca di avere a disposizione un corpus di dati strutturati ancora più vasto e diversificato per quanto concerne l'ambito della lingua italiana, soprattutto se si parla di fonti legislative e giuridiche. Questo poiché è ben chiaro come la quantità e la qualità dei dati utilizzati nell'addestramento riveste un ruolo chiave nel processo di formazione dei modelli destinati all'ambito legale, non solo facilitando l'apprendimento massivo del conteso informativo, ma anche determinando la capacità del modello di corretta interpretazione ed applicazione del linguaggio giuridico. Questo si collega con la necessità generale di un investimento maggiore nella creazione di modelli di linguaggio di qualità, pre-addestrati specificamente per le specificità del linguaggio italiano. Questi modelli potrebbero migliorare notevolmente la comprensione e la generazione di testo giuridico, riducendo gli errori e aumentando la rilevanza delle risposte generate.

Il progetto ParlaMentis ha dimostrato di essere un valido e innovativo strumento per assistere i deputati italiani, migliorando significativamente l'accesso e la gestione dei documenti legislativi. Nonostante le sfide legate alla specificità del linguaggio giuridico e alle necessità del contesto legale, le prospettive di innovazione rimangono molteplici. Siamo fiduciosi che i futuri sviluppi nel campo dell'NLP e l'ampliamento delle risorse informative permetteranno di superare questi ostacoli, rendendo i sistemi di supporto legislativo ancora più efficaci e precisi.

Giorgio Fantilli
Luglio, 2024

Bibliografia

[1] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *CoRR*, abs/1809.09600, 2018. URL `http://arxiv.org/abs/1809.09600`.

[2] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *CoRR*, abs/2010.03768, 2020. URL `https://arxiv.org/abs/2010.03768`.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html`.

[4] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *CoRR*, abs/2307.06435, 2023. doi: 10.48550/ARXIV.2307.06435. URL `https://doi.org/10.48550/arXiv.2307.06435`.

[5] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *CoRR*, abs/2312.10997, 2023. doi: 10.48550/ARXIV.2312.10997. URL `https://doi.org/10.48550/arXiv.2312.10997`.

[6] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *CoRR*, abs/2210.03629, 2022. doi: 10.48550/ARXIV.2210.03629. URL `https://doi.org/10.48550/arXiv.2210.03629`.

[7] Lorenzo De Mattei, Michele Cafagna, Felice Dell'Orletta, Malvina Nissim, and Marco Guerini. Geppetto carves italian into a language model. *CoRR*, abs/2004.14253, 2020. URL `https://arxiv.org/abs/2004.14253`.

[8] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori Hashimoto. Alpaca: A strong, replicable instruction-following model. *Center for Research on Foundation Models*, 2023.

[9] Andrea Santilli and Emanuele Rodolà. Camoscio: An italian instruction-tuned llama. In Federico Boschetti, Gianluca E. Lebani, Bernardo Magnini, and Nicole Novielli, editors, *Proceedings of the 9th Italian Conference on Computational Linguistics, Venice, Italy, November 30 - December 2, 2023*, volume 3596 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL `https://ceur-ws.org/Vol-3596/paper44.pdf`.

[10] Andrea Bacciu, Giovanni Trappolini, Andrea Santilli, Emanuele Rodolà, and Fabrizio Silvestri. Fauno: The italian large language model that will leave you senza parole! In Franco Maria Nardini, Nicola Tonellotto, Guglielmo Faggioli, and Antonio Ferrara, editors, *Proceedings of the 13th Italian Information Retrieval Workshop (IIR 2023), Pisa, Italy, June 8-9, 2023*, volume 3448 of *CEUR Workshop Proceedings*, pages 9–17. CEUR-WS.org, 2023. URL `https://ceur-ws.org/Vol-3448/paper-24.pdf`.

[11] Canwen Xu, Daya Guo, Nan Duan, and Julian J. McAuley. Baize: An open-source chat model with parameter-efficient tuning on self-chat data. *CoRR*, abs/2304.01196, 2023. doi: 10.48550/ARXIV.2304.01196. URL https://doi.org/10.48550/arXiv.2304.01196.

[12] Andrea Bacciu, Cesare Campagnano, Giovanni Trappolini, and Fabrizio Silvestri. Dantellm: Let's push italian LLM research forward! In Nicoletta Calzolari, Min-Yen Kan, Véronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue, editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pages 4343–4355. ELRA and ICCL, 2024. URL https://aclanthology.org/2024.lrec-main.388.

[13] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *CoRR*, abs/2310.06825, 2023. doi: 10.48550/ARXIV.2310.06825. URL https://doi.org/10.48550/arXiv.2310.06825.

[14] Danilo Croce, Alexandra Zelenanska, and Roberto Basili. Neural learning for question answering in italian. In Chiara Ghidini, Bernardo Magnini, Andrea Passerini, and Paolo Traverso, editors, *AI*IA 2018 - Advances in Artificial Intelligence - XVIIth International Conference of the Italian Association for Artificial Intelligence, Trento, Italy, November 20-23, 2018, Proceedings*, volume 11298 of *Lecture Notes in Computer Science*, pages 389–402. Springer, 2018. doi: 10.1007/978-3-030-03840-3_29. URL https://doi.org/10.1007/978-3-030-03840-3_29.

[15] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of Machine Translation Summit X: Papers, MTSummit 2005, Phuket, Thailand, September 13-15, 2005*, pages 79–86, 2005. URL https://aclanthology.org/2005.mtsummit-papers.11.

[16] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457, 2018. URL `http://arxiv.org/abs/1803.05457`.

[17] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/V1/P19-1472. URL `https://doi.org/10.18653/v1/p19-1472`.

[18] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=d7KBjmI3GmQ`.

[19] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 3214–3252. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.ACL-LONG.229. URL `https://doi.org/10.18653/v1/2022.acl-long.229`.

[20] Claude E Shannon. Prediction and entropy of printed english. *Bell system technical journal*, 30(1):50–64, 1951.

[21] Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*, 2023.

[22] Adam Ibrahim, Benjamin Thérien, Kshitij Gupta, Mats L. Richter, Quentin Anthony, Timothée Lesort, Eugene Belilovsky, and Irina Rish. Simple and scalable strategies to continually pre-train large language models. *CoRR*, abs/2403.08763, 2024. doi: 10.48550/ARXIV.2403.08763. URL https://doi.org/10.48550/arXiv.2403.08763.

[23] Daniel Martin Katz, Dirk Hartung, Lauritz Gerlach, Abhik Jana, et al. Natural language processing in the legal domain. *CoRR*, abs/2302.12039, 2023. doi: 10.48550/arXiv.2302.12039. URL https://doi.org/10.48550/arXiv.2302.12039.

[24] Justin Zhao, Timothy Wang, Wael Abid, Geoffrey Angus, Arnav Garg, Jeffery Kinnison, Alex Sherstinsky, Piero Molino, Travis Addair, and Devvret Rishi. Lora land: 310 fine-tuned llms that rival gpt-4, a technical report, 2024.

[25] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael I. Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference. *CoRR*, abs/2403.04132, 2024. doi: 10.48550/ARXIV.2403.04132. URL https://doi.org/10.48550/arXiv.2403.04132.

[26] Jiwoo Hong, Noah Lee, and James Thorne. ORPO: monolithic preference optimization without reference model. *CoRR*, abs/2403.07691, 2024. doi: 10.48550/ARXIV.2403.07691. URL https://doi.org/10.48550/arXiv.2403.07691.

[27] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *CoRR*, abs/2311.05232, 2023. doi: 10.48550/ARXIV.2311.05232. URL https://doi.org/10.48550/arXiv.2311.05232.

[28] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. Siren's song in the AI ocean: A survey on hallucination in large language models. *CoRR*, abs/2309.01219, 2023. doi: 10.48550/ARXIV.2309.01219. URL `https://doi.org/10.48550/arXiv.2309.01219`.

[29] Elizabeth Clark, Tal August, Sofia Serrano, Nikita Haduong, Suchin Gururangan, and Noah A. Smith. All that's 'human' is not gold: Evaluating human evaluation of generated text. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7282–7296, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.565. URL `https://aclanthology.org/2021.acl-long.565`.

[30] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL `https://aclanthology.org/W04-1013`.

[31] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with BERT. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=SkeHuCVFDr`.

[32] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, 1977.

[33] Weizhe Yuan, Graham Neubig, and Pengfei Liu. Bartscore: Evaluating generated text as text generation. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Sy-*

stems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, pages 27263–27277, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/e4d2b6e6fdeca3e60e0f1a62fee3d9dd-Abstract.html.

[34] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.ACL-MAIN.703. URL https://doi.org/10.18653/v1/2020.acl-main.703.