Università Politecnica delle Marche

Facoltà di Ingegneria

Corso di Laurea Magistrale in

Ingegneria Informatica e dell'Automazione

Uso di tecniche basate sui Large Language Models per la modellazione tematica di raccolte di documenti

Use of Large Language Models-based techniques for topic modeling of document collections

Relatore:

Prof: Alessandro Cucchiarelli

Candidato:

Manuel Manelli

Correlatore:

Prof: Christian Morbidoni

Anno Accademico 2023-2024

A Francesco, Monica,

Mattia e Cristina.

Rispettivamente Padre, Madre,

Fratello e Compagna di questa Avventura.

Abstract

Gli ultimi anni hanno visto una rivoluzione nel campo dell'intelligenza artificiale grazie all'ascesa dei Modelli di Linguaggio di Grandi Dimensioni (LLM), che hanno migliorato notevolmente l'elaborazione automatica del linguaggio naturale. Tra le applicazioni emergenti, il topic modelling tramite LLM si distingue per la sua capacità di organizzare e sintetizzare enormi quantità di testi in categorie tematiche, facilitando l'accesso e la gestione delle informazioni. Questa tecnologia permette non solo una catalogazione efficace per argomenti e la creazione di etichette riassuntive ma apre anche nuove prospettive nella navigazione di archivi documentali, migliorando l'efficienza nella ricerca di informazioni e stimolando la scoperta di connessioni interdisciplinari. Il topic modelling, quindi, rappresenta un punto di svolta per la disseminazione e l'organizzazione della conoscenza, evidenziando il potenziale degli LLM nell'evoluzione delle pratiche di gestione dei dati non strutturati. In questo lavoro di tesi si è provveduto a confrontare i risultati prodotti da diverse tecniche e da diversi strumenti più o meno sofisticati per implementare un sistema di generazione di etichette a partire da una base documentale costituita da abstract di articoli di ricerca relativi all'ambito del Machine Learning.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Capitolo 1

Introduzione

Negli ultimi anni, i Modelli di Linguaggio di Grandi Dimensioni (LLM) hanno rivoluzionato il campo dell'intelligenza artificiale, segnando un'epoca di progressi senza precedenti nel trattamento del linguaggio naturale. Questi modelli, capaci di comprendere, generare e interpretare testi in maniera sorprendentemente umana, hanno aperto nuovi orizzonti in una varietà di applicazioni, dall'assistenza virtuale alla traduzione automatica, dall'analisi del sentiment alla creazione di contenuti. Grazie alla loro straordinaria capacità di apprendere da enormi dataset, gli LLM hanno mostrato una versatilità e una precisione che superano quanto visto in precedenza, rendendoli protagonisti di una vera e propria rivoluzione tecnologica. L'avvento di questi modelli ha non solo migliorato significativamente le interfacce utente e le esperienze digitali ma ha anche stimolato riflessioni sul futuro dell'interazione uomo-macchina e sulle potenzialità cognitive dell'intelligenza artificiale.

Questa rivoluzione ha portato a una democratizzazione dell'accesso alla tecnologia avanzata di elaborazione del linguaggio, consentendo ad aziende, ricercatori e sviluppatori di sfruttare la potenza degli LLM per creare soluzioni innovative che prima erano impensabili. La capacità di questi modelli di elaborare il linguaggio in maniera così raffinata ha reso possibile la realizzazione di sistemi conversazionali che possono interagire con gli utenti in modo naturale e intuitivo, offrendo risposte e assistenza in tempo reale su una vasta gamma di argomenti. Inoltre, l'impiego degli LLM nel campo educativo e della ricerca ha facilitato l'accesso a conoscenze e informazioni, riducendo le barriere linguistiche e culturali. Gli studenti e i ricercatori

possono ora ottenere riassunti di documenti complessi, generare nuove idee o tradurre testi in diverse lingue con una facilità mai vista prima.

Tuttavia, l'avvento degli LLM solleva anche importanti questioni etiche e sfide, come il rischio di bias nei dataset, la privacy dei dati e la potenziale diffusione di informazioni errate. La comunità scientifica e i regolatori sono quindi chiamati a collaborare per stabilire linee guida che promuovano un uso responsabile e sostenibile di queste tecnologie.

1.1 Topic modelling con LLM

Il topic modelling, una delle applicazioni più promettenti degli LLM, rappresenta un ulteriore passo avanti nell'esplorazione delle capacità di questi modelli di intelligenza artificiale. Questa tecnica permette di scoprire, analizzare e categorizzare i temi principali oggetto di grandi volumi di testo, rendendo gli LLM strumenti indispensabili per l'organizzazione e la sintesi delle informazioni in modo automatico ed efficiente.

Attraverso l'impiego degli LLM nel topic modelling, si è aperta la possibilità di gestire dataset testuali di dimensioni precedentemente proibitive, individuando strutture e pattern nascosti all'interno dei dati. Questo ha notevoli implicazioni in vari campi, dalla ricerca accademica al monitoraggio dei social media, dall'analisi di mercato alla gestione documentale, dove la capacità di comprendere rapidamente i contenuti e le tendenze diventa un vantaggio competitivo fondamentale.

L'utilità del topic modelling nell'ambito della catalogazione per argomenti e nella creazione di etichette sintetiche rappresenta un aspetto fondamentale che amplifica il valore di questa tecnologia. Attraverso l'analisi effettuata dagli LLM, è possibile organizzare vasti archivi di documenti e testi in categorie tematiche ben definite, semplificando notevolmente la ricerca e l'accesso alle informazioni. Questa categorizzazione permette agli utenti di filtrare rapidamente i contenuti secondo i loro interessi, migliorando l'efficienza nella gestione delle risorse informative.

Inoltre, la generazione automatica di etichette che sintetizzano i temi principali di ciascun documento o frammento di testo contribuisce a creare un sistema di navigazione intuitivo all'interno di grandi database di conoscenza. Queste etichette fungono da indicatori immediati del contenuto, facilitando la comprensione del contesto anche prima della lettura dettagliata del materiale, e permettendo una classificazione semantica accurata dei dati.

La catalogazione per argomenti e la sintesi attraverso etichette apportano benefici significativi in numerosi settori, dall'editoria digitale alla gestione documentale aziendale, dalla ricerca scientifica alla cura dei contenuti per il web. La capacità di distillare informazioni complesse in categorie e tag comprensibili rende il topic modelling uno strumento prezioso per la disseminazione del sapere, la condivisione delle conoscenze e la creazione di basi dati organizzate secondo criteri logici e intuitivi.

In questo modo, il topic modelling tramite LLM non solo migliora l'accessibilità e la fruibilità delle informazioni ma stimola anche l'interdisciplinarità e la scoperta di connessioni inaspettate tra diversi ambiti di studio o settori di attività, sottolineando il ruolo cruciale dell'intelligenza artificiale nell'evoluzione delle pratiche di gestione della conoscenza.

1.2 Scenari di impiego

Immaginiamo alcuni scenari futuristici in cui il topic modelling con LLM trasforma radicalmente diversi settori:

- **Ricerca Accademica Avanzata**: Ricercatori di varie discipline utilizzano il topic modelling per analizzare rapidamente la letteratura esistente, identificando tendenze emergenti e lacune nella ricerca. Gli LLM semplificano la revisione sistematica di migliaia di pubblicazioni, consentendo agli studiosi di focalizzarsi su nuove aree di indagine e di collaborazione interdisciplinare.
- **Assistenza Sanitaria Personalizzata**: Nel settore sanitario, il topic modelling aiuta i medici a navigare attraverso vasti database di studi clinici e documentazione paziente,

individuando trattamenti e ricerche pertinenti a condizioni specifiche. Ciò consente una diagnosi più rapida e l'elaborazione di piani terapeutici personalizzati, migliorando significativamente l'assistenza al paziente.

- **Analisi di Mercato Dinamica**: Le aziende implementano il topic modelling per analizzare feedback dei clienti, recensioni di prodotti e discussioni sui social media, ottenendo una comprensione profonda delle percezioni del marchio e delle esigenze non soddisfatte. Questo permette alle aziende di adattare rapidamente strategie di marketing e sviluppo prodotto in risposta alle tendenze emergenti.

- **Intelligenza Competitiva**: Agenzie governative utilizzano il topic modelling per monitorare sviluppi geopolitici e tendenze economiche globali, analizzando vasti volumi di rapporti, articoli di giornale e comunicazioni digitali. Questo consente di anticipare movimenti strategici e di formulare politiche basate su dati concreti e analisi predittive.

- **Sicurezza Informatica e Prevenzione delle Minacce**: Nel campo della sicurezza informatica, il topic modelling tramite LLM diventa uno strumento chiave per l'analisi e la classificazione di enormi volumi di dati relativi a minacce e vulnerabilità. Organizzazioni e istituzioni possono utilizzare questi modelli per filtrare e interpretare segnalazioni di sicurezza, bollettini di vulnerabilità e discussioni nei forum specializzati, identificando schemi e tendenze nelle tecniche di attacco. Questo consente di anticipare potenziali minacce e di sviluppare strategie di difesa più efficaci, migliorando la resilienza di sistemi e reti di fronte a cyber-attacchi sempre più sofisticati.

In ciascuno di questi scenari, il topic modelling tramite LLM non solo migliora l'efficienza e l'accuratezza dell'analisi dei dati ma apre anche la strada a nuove modalità di interazione con l'informazione, promuovendo un approccio più informato e proattivo alle decisioni in campi diversificati.

1.2.1 Topic modelling in ambito accademico

Come accennato in precendenza, nell'ambito della ricerca accademica avanzata, l'uso degli LLM per il topic modelling si rivela particolarmente prezioso nell'analizzare e sintetizzare

grandi volumi di pubblicazioni scientifiche. Questa tecnologia permette ai ricercatori di identificare rapidamente le tendenze emergenti e le aree meno esplorate all'interno di un campo specifico, facilitando la scoperta di nuove direzioni di ricerca e incentivando collaborazioni interdisciplinari, grazie alla capacità di elaborare e comprendere testi con un livello di profondità e precisione senza precedenti. A tal proposito in rete esistono lavori preliminari che prevedono l'utilizzo di strumenti come BERTopic e Llama2 per estrarre i topic e Llama2 per sintetizzare le etichette. BERTopic è un modello di topic modelling che utilizza tecniche di clustering su rappresentazioni dense del testo per identificare temi latenti nei dati, mentre Llama2 è un avanzato modello di linguaggio sviluppato da Meta, progettato per generare testi coerenti e contestualmente rilevanti, utile per creare etichette sintetiche che riassumano efficacemente i topic individuati.

Capitolo 2

Contesto applicativo

Negli ultimi anni, l'evoluzione dei modelli di linguaggio ha segnato una svolta nel campo del Natural Language Processing (NLP). Partendo da approcci basati su regole e statistiche, si è passati a modelli di apprendimento profondo, come le reti neurali, che hanno permesso un salto qualitativo nella comprensione e generazione del linguaggio naturale. Il lancio di modelli come BERT, GPT-3 e GPT-4 ha ulteriormente spinto i confini, offrendo capacità di comprensione e interattività senza precedenti, aprendo nuove strade per applicazioni pratiche e ricerca.

2.1 Large Language Model (LLM)

Un Large Language Model (LLM) è un tipo avanzato di intelligenza artificiale generativa che utilizza algoritmi di deep learning per cercare di simulare il modo in cui le persone potrebbero pensare. "Large" (Grande) in questo contesto si riferisce alla dimensione del modello, spesso con miliardi di parametri, che influenzano le sue capacità nella generazione di output. Un LLM è di fatto costituito da una Deep Neural Network composta da numerosi layer nascosti in cui i pesi dei singoli neuroni che li compongono (Parametri) vengono attribuiti tramite un processo che prende il nome di addestramento composto da due fasi, pre-training e fine-tuning.

2.1.1 Pre-training

Il processo di addestramento[3] di un Large Language Model (LLM) inizia con il Pre-training (Fig. 1), dove il modello viene esposto a grandi quantità di testi, noti come corpus, che possono estendersi fino a diversi terabyte. Durante questa fase, il modello apprende le regole grammaticali, i modelli linguistici, come prevedere la sequenza di parole che compongono una frase, come riempire gli spazi vuoti delle parole in un testo, senza ulteriori informazioni di classificazione (etichette) delle singole frasi del testo. Di fatto si tratta di una fase di addestramento non supervisionato. Questo processo è fondamentale per consentire al modello di catturare i pattern intrinseci e le relazioni tra le parole proprie del linguaggio e richiede un'ampia capacità di calcolo, spesso possibile solo tramite supercalcolatori. Esistono vari LLM, alcuni commerciali come GPT di OpenAI e altri disponibili come Llama2 di Meta, adattabili per diversi scopi. Una volta che la fase di pre-training è stata ultimata, l'insieme dei parametri ottenuti prende il nome di modello. I modelli che si ottengono dalla fase di addestramento sono modelli molto generici, e possono trovare diversi utilizzi.

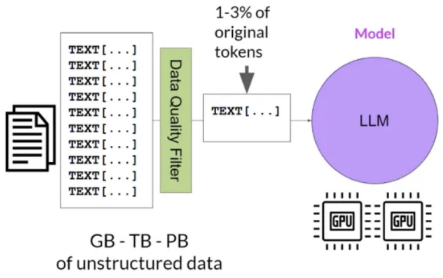


Figura 1: *Pre-training di un LLM*

2.1.2 Fine-tuning

Per poter trovare impiego in ambiti specifici è richiesta un ulteriore fase dopo l'addestramento che prende il nome di fine-tuning (Fig. 2). Tale fase ha lo scopo di sottoporre al modello, dopo la fase di addestramento iniziale, un insieme di testi di dimensione notevolmente ridotte ma molto specifici per un determinato ambito applicativo con l'obbiettivo di focalizzare il modello generico in un determinato dominio applicativo. Quando il modello elabora i dati specifici del dominio, calcola la differenza tra le sue previsioni e i risultati effettivi. Questa

differenza, nota come gradiente, guida le regolazioni dei parametri. Le tecniche di ottimizzazione utilizzano quindi queste informazioni sul gradiente per perfezionare iterativamente i parametri del modello. In questo modo si riducono al minimo gli errori di previsione e si migliora la competenza specifica del LLM.

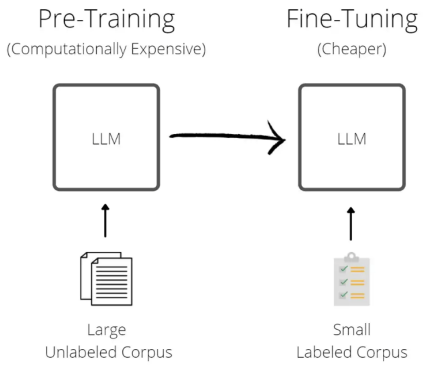


Figura 2: *Fine-tuning di un LLM*

2.1.3 In-context learning

L'apprendimento in contesto si riferisce a un metodo per guidare il comportamento del modello in base al contesto specifico che gli è stato fornito durante la formulazione della richiesta. A differenza del fine-tuning, l'apprendimento in-context non richiede la modifica dei parametri del modello o l'addestramento del modello su un set di dati specifico. Piuttosto, si fornisce al modello una richiesta o una serie di istruzioni all'interno dell'interazione stessa per influenzare le sue risposte. Il modello utilizza questa richiesta per condizionare il suo output, generando una risposta appropriata al contesto specificato. Su questa metodologia di apprendimento si basa il prompt engineering.

2.1.4 Principali utilizzi

Uno degli utilizzi principali di questi LLM è la generazione di testo, che rappresenta un'impresa notevole nel campo dell'intelligenza artificiale. Questi modelli utilizzano i dati appresi in fase di addestramento per creare testi che spaziano dallo svolgimento di compiti specifici alla composizione di contenuti creativi. Operando con miliardi di parametri, gli LLM generano testo mediante la previsione del token successivo in una sequenza, basandosi sul

contesto fornito dall'utente. Questo processo consente ai modelli di produrre risposte, idee e contenuti che appaiono simili a quelli prodotti dagli umani e contestualmente rilevanti. Gli algoritmi generativi non sono una novità nel campo dell'elaborazione del linguaggio naturale. In passato, si utilizzavano modelli basati su reti neurali ricorrenti (RNN), che però erano limitati dalla necessità di ampie risorse computazionali e di memoria per eseguire compiti generativi. Le RNN, nonostante la loro potenza, mostravano limiti nella previsione del testo, specialmente con contesti più ampi. Tuttavia, con l'introduzione dell'architettura Transformer nel 2017[10], si è assistito a un salto qualitativo significativo. Questa nuova architettura ha permesso una più efficiente scalabilità, elaborazione parallela dei dati e una capacità avanzata di "attenzione" al significato delle parole, portando a progressi notevoli nell'intelligenza artificiale generativa.

2.2 Transformer Architecture

L'architettura Transformer, utilizzata negli LLM, ha segnato una svolta nel campo dell'elaborazione del linguaggio naturale, superando le limitazioni delle precedenti reti neurali ricorrenti (RNN). La sua efficacia risiede nell'apprendere il contesto e la rilevanza di tutte le parole in una frase (Fig. 3), attraverso un meccanismo chiamato self-attention. Questo meccanismo consente al modello di valutare l'importanza di ogni parola rispetto alle altre in un contesto, indipendentemente dalla loro posizione. Durante la fase di addestramento il modello appren-

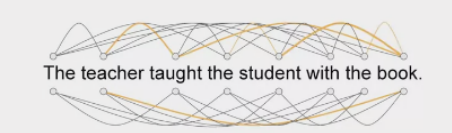


Figura 3: *Rilevanza delle parole nell'insieme della frase*

de i così detti attention weights. Essi, nell'architettura Transformer, sono fondamentali per determinare come le diverse parole in una frase influenzino l'una l'altra. Questi pesi permettono al modello di focalizzarsi su parti specifiche del testo, assegnando maggiore importanza a certe parole rispetto ad altre nel contesto. In pratica, gli attention weights (rappresentati in giallo in Fig. 4) aiutano il modello a capire la relazione e l'importanza relativa tra le parole, migliorando così la qualità della generazione e comprensione del linguaggio. Questo consente

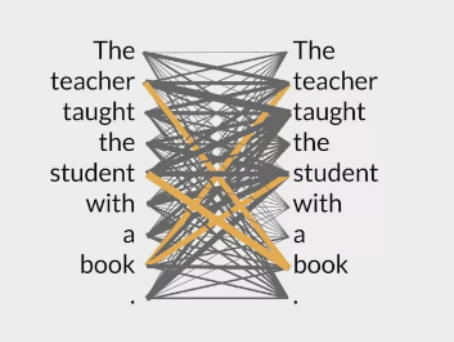


Figura 4: *Attention weights*

all'algoritmo, ad esempio, di rispondere meglio in casi di sinonimia oltre che di avere contezza di quali parole sono vicine tra loro.

2.2.1 Descrizione dell'architettura

L'architettura del modello Transformer, rappresentata ad alto livello in Fig. 5 e nel dettaglio in Fig. 7, è un'innovazione chiave nel campo dell'elaborazione del linguaggio naturale. Essa si distingue per la sua struttura divisa in due parti principali: l'encoder e il decoder. L'encoder processa il testo di input, mentre il decoder genera l'output. Come si può osservare nella Fig. 5, in ingresso all'encoder e al decoder sono presenti due componenti denominati embeddings. Nel dettaglio, gli embeddings catturano le relazioni semantiche tra le parole, riflettendo le loro similitudini e differenze in uno spazio vettoriale multidimensionale. Ciò significa che parole con significati simili sono rappresentate da vettori vicini nello spazio degli embeddings, mentre parole con significati diversi si trovano a distanze maggiori. Questi vettori, come rappresentato in Fig. 6 catturano il significato semantico e le relazioni sintattiche delle parole attraverso uno spazio vettoriale multidimensionale. Gli embeddings sono fondamentali per trasformare i testi, che sono dati non strutturati, in formati che i modelli di machine learning possono elaborare permettendo al modello di interpretare il linguaggio in modo più efficace e sfumato. Terminato il processo di embeddings i vettori vengono passati ad uno strato interno dei encoder/decoder chiamato self-attention (come si può vedere in Fig. 7) che consente al modello di analizzare la frase da più punti utilizzando altre informazioni apprese in precedenza durante la fase di addestramento del modello. Questo passo non è fatto una volta sola ma più volte in parallelo in modo che ogni processo chiamato self-attention

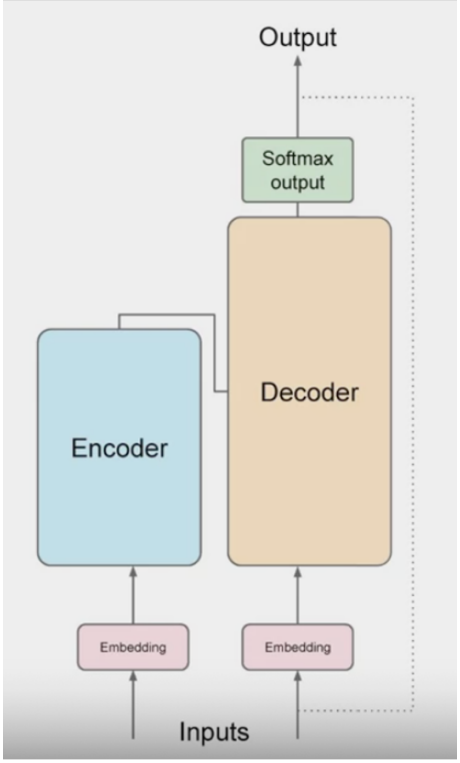


Figura 5: *Architettura del modello transformer*

head possa catturare aspetti diversi del linguaggio sottoforma di pesi chiamati self-attention weights. Il passo successivo è processare il tutto attraverso una fully-connected feed-forward network. L'output di questo layer è un vettore di interi non normalizzati che prende il nome di logit, proporzionale al valore di probabilità per ogni chunk di testo (token) nel dizionario dei token. Questi logits sono passati all'ultimo layer chiamato softmax dove sono normalizzati in valori di probabilità per ogni parola nel vocabolario. Per quanto riguarda i task di sentence to sentence (Fig. 8) le informazioni ricavate dall'encoder servono al decoder per produrre la frase di out. Più precisamente influenzano la fase di self attention del decoder. Successivamente viene posto un token di inizio frase nel decoder per fare in modo che questo inizi a predire i token successivi a quello dato sulla base delle informazioni che il decoder ha acquisito riguardo al contesto fornitogli dall'encoder e a ciò che ha già appreso in fase di addestramento. Ottenuti tutti i token questi vengono ritrasformati in parole ottenendo infine la frase finale. Riassumendo abbiamo che l'architettura completa è composta da encoder e decoder. L'encoder codifica sequenze di input in una rappresentazione della struttura e del

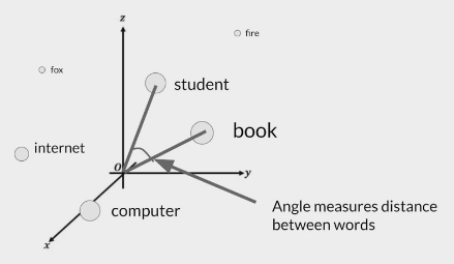


Figura 6: *Rappresentazione degli embeddings associati alle parole in uno spazio vettoriale*

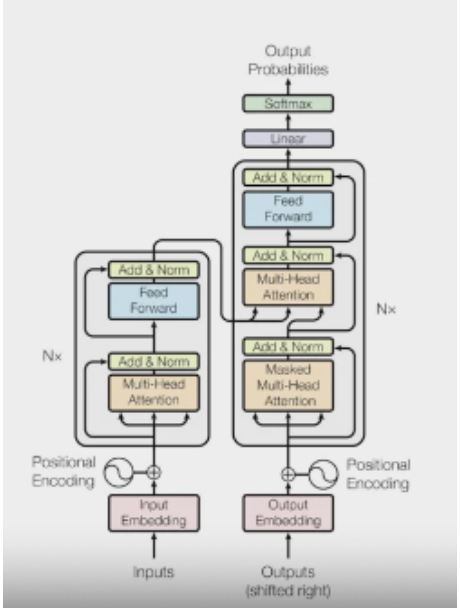


Figura 7: *Architettura transformer vista a basso livello*

significato dell'input. Il decoder, lavorando in base ai segnali dei token di input, utilizza la comprensione contestuale dell'encoder per generare nuovi token. Fa ciò fino a quando non si raggiunge una condizione di arresto. La componente finale dell'architettura, il blocco in verde in Fig. 5, prende il nome di layer softmax. Questo layer ha il compito di trasformare i logit, cioè i valori grezzi di output di un modello, in una distribuzione di probabilità. In contesti come la generazione di testo, il layer softmax determina quale parola il modello prevede che venga successivamente, assegnando una probabilità a ogni possibile parola nel vocabolario del modello. Esistono due modi con cui i modelli partono dalla distribuzione di probabilità delle parole ottenuta dall'ultimo layer per generale le parole successive da inserire nella frase. Il primo approccio è il greedy, cioè semplicemente si sceglie la parola da predire andando a

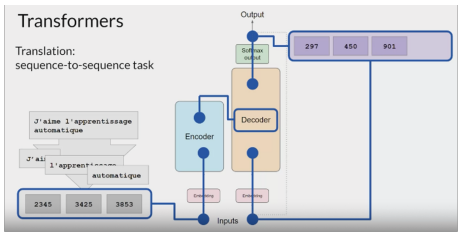


Figura 8: *Schema di funzionamento dei task sentence to sentence*

prendere quella con la probabilità più alta dalla lista ottenuta dal modello. Questo è semplice ma è soggetto al problema delle parole o delle frasi ripetute. Il secondo approccio, il random sampling, risolve questo problema selezionando le parole a caso prendendo la probabilità della parola stessa come peso per la selezione. Il comportamento del layer softmax (Fig. 9)

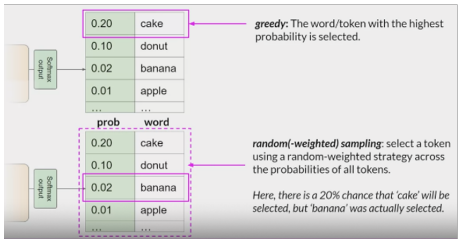


Figura 9: *Schema di funzionamento del layer softmax*

è influenzato da una serie di parametri:

- **Max Tokens**: Indica il numero massimo di token che possono essere generati in una sequenza. Questo parametro limita la lunghezza dell'output prodotto dal modello.
- **Sample Top K**: Durante la generazione di testo, il modello considera solo i primi 'K' token più probabili come possibili candidati per il prossimo token, dove 'K' è un numero definito dall'utente. Questo restringe la scelta ai token più rilevanti.
- **Top P**: Invece di limitarsi ai primi 'K' token, il modello seleziona un sottoinsieme di token che collettivamente superano una soglia di probabilità 'P'. Questo metodo consente una generazione di testo più variabile e meno prevedibile.
- **Temperature**: La temperatura (Fig. 10) regola la casualità nella selezione dei token. Una temperatura più alta porta a scelte più casuali e diverse, mentre una temperatura

più bassa rende il modello più conservativo e predittivo, tendendo a scegliere il token più probabile.

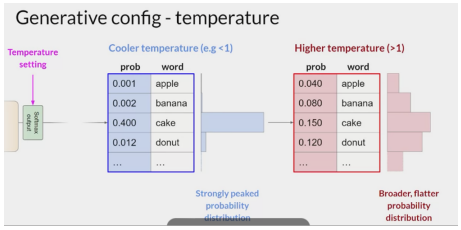


Figura 10: *Effetto della temperatura*

2.2.2 Utilizzi

A seconda di come viene utilizzata l'architettura transformer è possibile svolgere compiti diversi.

Encoder only

I modelli che utilizzano solo l'encoder (Fig. 11) nell'architettura Transformer sono tipicamente impiegati in compiti di elaborazione del linguaggio naturale che non richiedono una generazione sequenziale di output. Esempi notevoli includono BERT (Bidirectional Encoder Representations from Transformers) e modelli simili. Questi modelli sono particolarmente efficaci in compiti come la classificazione del testo, la sentiment analysis il riconoscimento di entità nominali, e la risposta a domande. L'uso esclusivo dell'encoder consente di elaborare l'intera sequenza di input contemporaneamente, catturando le relazioni complesse all'interno del testo.

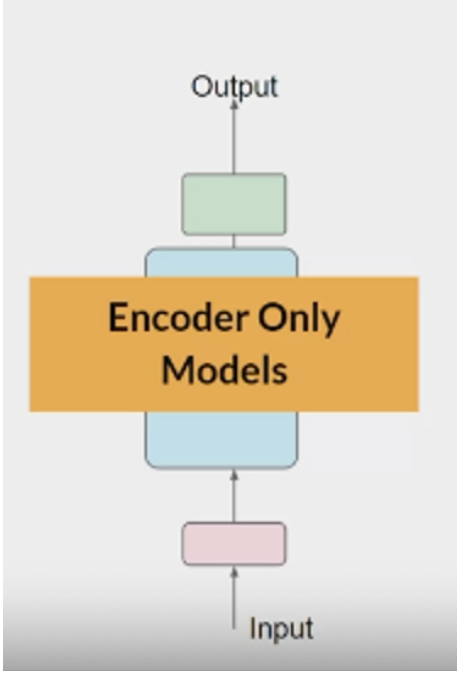


Figura 11: *Architettura solo encoder*

Encoder/Decoder

I modelli che utilizzano sia l'encoder che il decoder nell'architettura Transformer sono progettati per compiti di elaborazione del linguaggio che richiedono una trasformazione del testo di input in un nuovo testo di output. L'encoder processa il testo di input, mentre il decoder genera il testo di output (Fig. 12). Questi modelli sono particolarmente adatti per compiti come la traduzione automatica, il riassunto di testi e la generazione di risposte in chatbot. La combinazione di encoder e decoder permette di catturare il contesto del testo di input e di generare risposte coerenti e contestualmente appropriate.

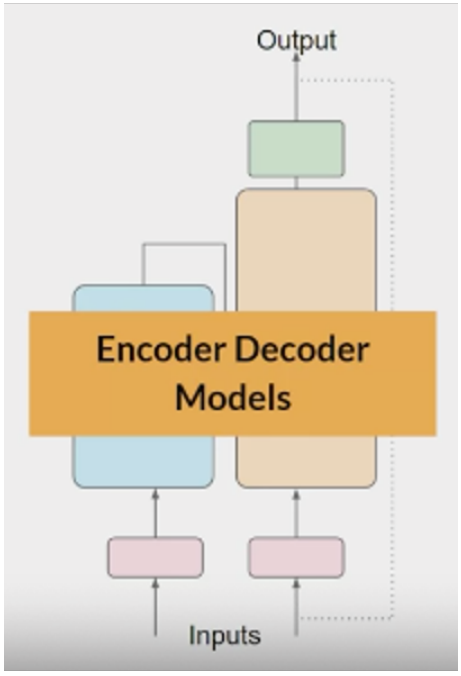


Figura 12: *Architettura encoder/decoder*

Decoder only

I modelli basati solamente sul decoder (Fig. 13), come GPT, BLOOM, Jurassic e LLaMA, sono tra i più comuni nel campo dell'intelligenza artificiale. Questi modelli sono particolarmente adatti per una vasta gamma di compiti, inclusa la generazione di testo, grazie alla loro capacità di prevedere il token successivo in una sequenza basandosi sul testo precedente. La loro architettura è ottimizzata per generare risposte coerenti e contestualmente appropriate, rendendoli strumenti versatili in diverse applicazioni di elaborazione del linguaggio naturale.

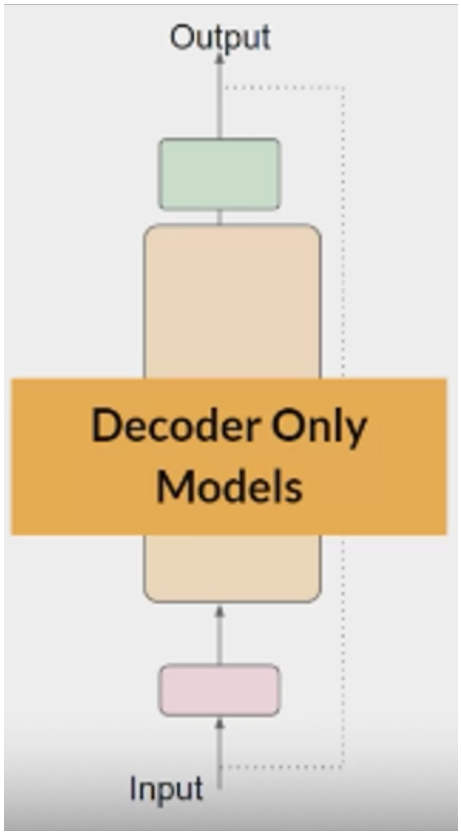


Figura 13: *Architettura decoder*

2.3 Prompting

Nei Large Language Models (LLM), i "prompt" sono input testuali forniti al modello per innescare una specifica risposta o output (Fig. 14). Questi prompt funzionano come indicazioni o domande che guidano il modello nel generare risposte coerenti e contestualmente pertinenti. La progettazione accurata di questi prompt è cruciale, poiché determina la qualità e la rilevanza del testo generato dal modello. In sostanza, i prompt sono gli strumenti attraverso i quali gli utenti comunicano con i modelli di linguaggio, definendo il contesto e la direzione della conversazione o dell'analisi linguistica. L'azione di generare il testo è chiamata inference. Può succedere che il modello non riesca a fornire sempre la risposta che vogliamo. Per aggiustare il tiro è possibile modificare la richiesta inserendo altre informazioni per ottenere dal modello l'out desiderato. Questo processo è noto come prompt engineering. Una tecnica abbastanza efficace è inserire nel prompt degli esempi che possano aiutare il modello a la

risposta desiderata. Questa tecnica di fornire esempi all'interno della finestra di richiesta è chiamata With in-context learning.

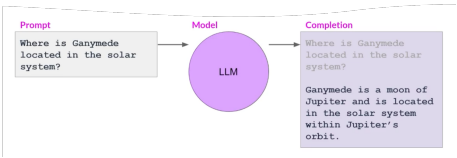


Figura 14: *Esempio di Inference*

Un solo esempio fornito prende il nome di single shot, più esempi prendono il nome di multi shot. Più i modelli sono grandi e più parametri hanno più è semplice raggiungere l'obbiettivo con dei prompt più semplici, più il modello è piccolo e più prompt articolate servono per raggiungere lo stesso obbiettivo. Come si può vedere in Fig. 15, un modello di dimensioni maggiori riesce a capire meglio dal testo che gli viene sottoposto cosa deve fare, in questo caso capire il "sentiment". Il modello di dimensione minore invece non riesce a cogliere affatto il senso e procede a generare banalmente una serie di token.

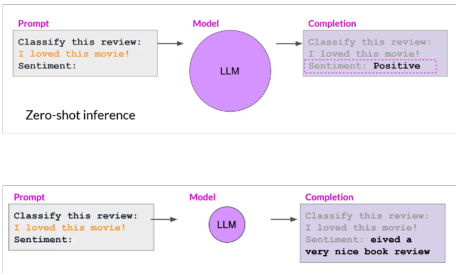


Figura 15: *Esempio di Inference al variare delle dimensioni del modello*

2.3.1 Descrizione formale del prompting

In un sistema tradizionale di apprendimento supervisionato prendiamo un input x, tipicamente testo, e cerchiamo di predire l'output y basato sul modello $P(y|x;\theta)$. y può essere un etichetta, del testo o altri tipi di input. Per ricavare i parametri θ del modello usiamo coppie di tipo input/output e addestriamo un modello per prevedere questa probabilità condizionata Il problema principale con i sistemi di apprendimento supervisionato è che è necessario avere dati etichettati per effettuare l'addestramento e questi sono difficili da reperire in grande quantità. I metodi di apprendimento basati su prompt cercano di evitare questo problema addestrando un modello di linguaggio che vada a modellare la probabilità $P(x:\theta)$ del testo

x andando ad usare questa probabilità per prevedere l'output y riducendo o addirittura ovviando completamente al bisogno di dataset etichettati. I metodi di prompting più utilizzati sono: Prompt addition, Answer search e Answer mapping.

Prompt addition

Prompt addition è un passo dove una cosidetta funzione di prompting $f_{prompt}()$ è applicata per modificare il testo di input x in un prompt x' tale che $x' = f_{prompt}(x)$. È composto da due passi:

- **1**:Applica il template che è una stringa che ha due slot: input slot `[x]` per gli input x e uno slot di risposta `[z]` per un testo intermedio di risposta che viene poi mappato nell'output y
- **2**:Riempie lo slot `[x]` con il testo di input x.

Nel caso della sentiment analysis x = "amo questo film", il template è tipo ‘’`[x]` comunque era un `[z]` film". x' diventa ‘’ amo questo film. Comunque era un `[z]` film"

Answer search

Il metodo successivo è quello di *answer search*, ovvero cerchiamo il testo $\hat{z}$ con il punteggio più alto che massimizza il punteggio del modello di linguaggio (LM). Prima, definiamo un insieme $\mathcal{Z}$ che contiene tutti i possibili valori di z. $\mathcal{Z}$ può essere un piccolo sottoinsieme di parole o può essere un intero dizionario. Definiamo poi una funzione $f_{fill}(x', z)$ che riempie nella posizione `[Z]` nel prompt x' con la risposta scelta z. Chiamiamo quindi ogni prompt sottoposto a tale processo un *filled prompt*. In particolare, se il prompt è riempito con una risposta vera, lo chiamiamo *answered prompt*. Infine, cerchiamo nel set delle potenziali risposte z andando a calcolare la probabilità delle corrispondenti *filled prompt* usando un modello pre-addestrato $P(\cdot; \theta)$:

$$\hat{z} = \underset{z \in \mathcal{Z}}{\text{search}}\, P(f_{fill}(x', z); \theta).$$

Questa funzione di ricerca potrebbe essere una ricerca *argmax* che cerca l'output con il punteggio più alto, o una ricerca campionata che genera casualmente output seguendo la distribuzione di probabilità del LM.

Answer mapping

Infine l'ultimo metodo serve per andare dalla risposta $\hat{z}$ a punteggio più alto all'output $\hat{y}$ a punteggio più alto. Questo è banale in alcuni casi in quanto la risposta stessa è l'output come in casi di generazione di linguaggio come le traduzioni ma ci sono altri casi dove questo non lo è.

2.3.2 Prompt engineering

È il processo di creazione della così detta funzione $f_{prompt}()$ di cui parlavamo sopra. Tale processo può essere eseguito o da un essere umano o da un algoritmo e consiste nell'andare a cercare il miglior template per ogni task che il modello deve eseguire. Esistono due diversi tipi di prompt, i cloze prompt che riempiono gli spazi vuoti di una stringa e i prefix prompts che completano una stringa con prefisso. Quale scegliere tra le due dipende da cosa deve fare il modello. Per i task di generazione di testo o task risolti in genere con modelli di tipo auto-regressivi i prefix prompts sono i migliori. Per le task risolvibili con i masked language models (LM), ossia modelli di linguaggio addestrati per prevedere token mancanti o "mascherati" all'interno di una sequenza di testo, i cloze prompt risultano i più efficaci, poiché forniscono un output molto simile a quello ottenuto durante i task di pre-addestramento. La creazione manuale dei prompt consente di risolvere vari task con un buon grado di accuratezza, ma presenta alcuni svantaggi significativi, come il notevole dispendio di tempo. Inoltre, per task molto complessi e articolati, esiste il rischio di commettere errori durante la fase di progettazione dei prompt. Per risolvere questi problemi sono stati proposti diversi metodi per automatizzare tale processo. In particolare i prompt prodotte automaticamente possono essere separate in prompt discrete, dove il prompt è un testo sottoforma di stringa, e i prompt continui, dove il prompt è rappresentato nello spazio di embedding. Un'altra tecnica prevede che la funzione di prompt $f_{prompt}()$ sia statica, ovvero usa lo stesso template per ogni input oppure dinamica se genera template custom per ogni input.

Multi prompt learning

I metodi di prompt engineering in genere utilizzano un'unico prompt come input. È stato dimostrato però che l'uso di prompt multiple può migliorare significativamente l'efficacia dei

metodi di prompting. Il metodo di prompt ensembling usa più prompt senza risposta, discrete o continue, come input in fase di inferenza per fare le predizioni. La tecnica più intuitiva per ottenere una predizione a partire da più prompt è quella di prendere la media delle probabilità dei diversi prompt $P(z|x) := \frac{1}{K} \sum_i^K P(z|f_{\text{prompt},i}(x))$ dove $f_{\text{prompt},i}(\cdot)$ è il i-esimo prompt nell'ensemble di prompt e $P(z|x)$ indica la probabilità che una certa sequenza di testo z sia generata, dato un prompt o una sequenza di input x . I pro sono sicuramente la facilità di implementazione ma può dare problemi in caso vi siano alcuni prompt più performanti di altri. Per risolvere questo problema alcuni lavori di ricerca hanno approfondito l'uso di una media pesata al posto di una aritmetica in modo da pesare ogni singolo prompt. I pesi sono associati in base alle performance dei prompt o ottimizzati usando dati di addestramento. Per task di classificazione invece può essere usata la tecnica del majority voting.

2.3.3 Tecniche di addestramento

Le tecniche di prompting, applicate ai modelli di linguaggio, sono spesso impiegate senza necessità di un addestramento specifico su nuovi dati. Questi modelli, già istruiti per valutare la probabilità del testo fornito, vengono utilizzati direttamente per risolvere task come il completamento di testi incompleti o la risposta a prompt prefissati. Tale approccio è definito "zero-shot", poiché non richiede dati di addestramento aggiuntivi relativi al dominio specifico di impiego. Esistono approcci che integrano i dati di addestramento con le tecniche di prompting per affinare i modelli di linguaggio. Questi metodi prevedono l'uso di un ampio insieme di esempi per l'addestramento, oppure di un numero limitato di essi. In contesti dove gli esempi disponibili sono scarsi, le tecniche di prompting diventano strumenti preziosi per orientare efficacemente i modelli verso le direzioni desiderate, compensando la mancanza di dati con indicazioni mirate.

Capitolo 3

Obbiettivi del progetto

3.1 Contesto e rilevanza del progetto

L'obiettivo principale di questo progetto di tesi è esplorare l'efficacia delle tecniche di prompt engineering nella generazione automatica di descrizioni sintetiche (etichette) di topic, applicate specificatamente alla ricerca accademica nel campo del machine learning. Il topic modelling è una tecnica di elaborazione del linguaggio naturale che permette di identificare e raggruppare automaticamente temi ricorrenti all'interno di un insieme di documenti. Tuttavia, il focus di questa ricerca non è solo sull'identificazione dei topic, ma sulla creazione automatica di etichette descrittive che sintetizzano efficacemente i contenuti di questi topic. Questo studio mira a valutare se l'impiego di metodologie avanzate di prompt engineering, combinato con gli strumenti di Retrieval-Augmented Generation (RAG), in particolare tramite il framework Llama Index, può migliorare significativamente il processo di generazione delle etichette rispetto alla formulazione manuale dei prompt. Per chiarire meglio l'obiettivo del progetto, di seguito vengono riportati alcuni esempi di topic espressi come sequenza di termini e le loro descrizioni sintetiche:

- **Termini del Topic**: {dynamics, reinforcement, learning, model, robot, training, algorithms, optimal, agents, gradient}
  - **Descrizione**: "Reinforcement Learning Techniques"

- **Termini del Topic**: {encoder, voice, speech, models, language, model, synthesis, trained, neural, audio}
  - **Descrizione**: "Speech Processing and Synthesis"
- **Termini del Topic**: {federated, heterogeneity, distributed, algorithms, datasets, gradient, models, convergence, model, algorithm}
  - **Descrizione**: "Privacy in Federated Learning"

3.2 Fasi del progetto

- **Estrazione dei Topic:** La ricerca inizia con l'estrazione dei topic da un dataset composto da abstract di articoli di ricerca. Questa fase è cruciale poiché stabilisce la base su cui si applicano le tecniche successive di prompt engineering e generazione di etichette.
- **Sviluppo di Prompt mediante Prompt Engineering:** Utilizzando il modello di linguaggio Llama2, il progetto procede con lo sviluppo di prompt specifici che guidano il modello a generare etichette concise e rappresentative per ciascuno dei topic identificati. Questi prompt si costruiscono sfruttando le parole chiave distintive di ogni topic e documenti chiave che sintetizzano il contenuto essenziale del topic.
- **Applicazione della Metodologia RAG:** Parallelamente, il progetto esplora l'uso della metodologia RAG attraverso Llama Index per determinare se il recupero di informazioni contestuali pertinenti dai documenti arricchisce la capacità del modello di generare etichette più accurate e descrittive.
- **Sperimentazione e Confronto:** Il progetto prevede una serie di esperimenti che variano sia nelle tecniche di indicizzazione utilizzate per la preparazione dei dati sia nelle modalità di sintesi della risposta finale. Si esaminano sia l'utilizzo di prompt predefiniti di Llama Index sia la creazione di prompt personalizzati basati sulle tecniche di prompt engineering più avanzate.

3.3 Valutazione dei risultati

I risultati si confrontano con quelli ottenuti da metodologie precedenti per stabilire l'efficacia delle nuove tecniche implementate. L'analisi quantitativa e qualitativa degli output generati aiuta a discernere se e come le innovazioni in prompt engineering e RAG contribuiscono a un miglioramento sostanziale nella descrizione e nell'analisi dei topic nel campo del machine learning.

Capitolo 4

Strumenti utilizzati

Per raggiungere gli scopi delineati nel capitolo precedente, è stata impiegata una gamma ampia di strumenti. Questo capitolo li presenterà e ne fornirà una descrizione dei più significativi.

4.1 Python

Il linguaggio di programmazione che è stato impiegato nel progetto è il **Python**[7]. Esso è un linguaggio di programmazione di nuova concezione, orientato agli oggetti, adatto anche a sviluppare applicazioni distribuite, scripting e computazione numerica.
I suoi costrutti linguistici e l'approccio orientato agli oggetti mirano ad aiutare i programmatori a scrivere codice chiaro e logicamente corretto per progetti su piccola e grande scala. Python è tipizzato dinamicamente e sottoposto a garbage collection. Supporta più paradigmi di programmazione, inclusa la programmazione strutturata (in particolare procedurale), orientata agli oggetti e funzionale. Python è spesso descritto come un linguaggio a 360 gradi, a causa della sua completa standard library.
Python è un linguaggio interpretato (Fig. 16), ovvero il codice sorgente non viene compilato prima di essere eseguito, ma il sorgente stesso è il programma da eseguire, a patto di avere disponibile sulla propria macchina la versione corretta dell'interprete Python.

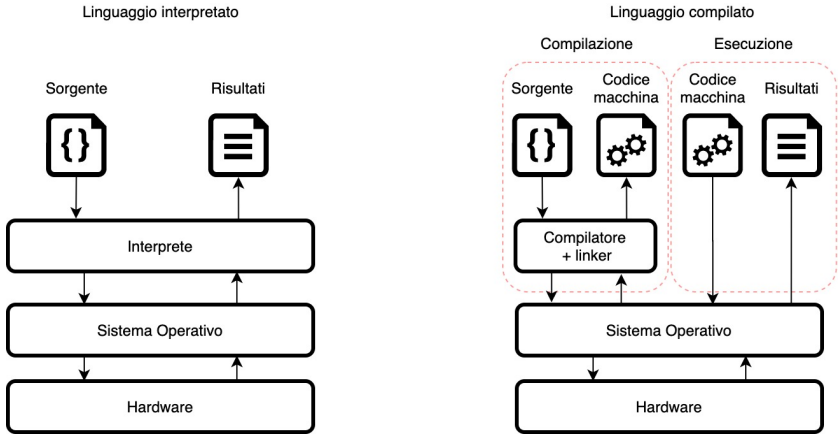


Figura 16: *Schema concettuale di un linguaggio interpretato e di uno compilato*

È possibile anche compilare un programma Python grazie all'ausilio del modulo `py_compile`. Esso mette a disposizione due funzioni, una permette di generare un file *byte code* da un file sorgente e l'altra viene utilizzata quando il sorgente del modulo viene richiamato come script. Python consente anche di aprire un terminale dove è possibile impartire comandi in tempo reale all'interprete. Un valore aggiunto del Python è dato dalla vasta gamma di librerie disponibili, del tutto open source messe a disposizione dalla community. Di seguito è riportata una descrizione dettagliata delle librerie utilizzate in questo progetto.

4.2 Google colab

Google Colaboratory, noto comunemente come Google Colab, è una piattaforma cloud che si pone l'obiettivo di facilitare la ricerca, l'istruzione e la collaborazione in ambito di machine learning e data science. Questo strumento si configura come un ambiente di sviluppo integrato (IDE) basato sul web che consente agli utenti di scrivere e eseguire codice Python direttamente dal proprio browser. La caratteristica distintiva di Google Colab risiede nella sua capacità di fornire accesso gratuito a risorse computazionali di notevole potenza, inclusi i processori grafici (GPU) e i tensor processing units (TPU), rendendolo attrattivo per compiti computazionalmente intensivi quali l'addestramento di modelli di machine learning. L'ambiente è costruito su Jupyter Notebook, un'applicazione web open-source che supporta la condivisione di documenti che contengono codice live, equazioni, visualizzazioni e testo narrativo. La piattaforma si distingue per la sua intuitiva interfaccia utente che facilita l'im-

portazione di dataset, la scrittura di codice, l'analisi dei risultati e la condivisione di progetti. Grazie all'integrazione diretta con Google Drive, gli utenti possono facilmente salvare i loro notebook e accedere ai dati senza la necessità di configurazioni complesse. Inoltre, Google Colab promuove la collaborazione in tempo reale, permettendo a più utenti di lavorare congiuntamente su un singolo notebook. Un altro aspetto rilevante di Google Colab è la sua natura pedagogica: la piattaforma si rivela uno strumento prezioso per l'apprendimento e l'insegnamento della programmazione e del machine learning, grazie alla possibilità di accedere a un vasto repertorio di notebook condivisi pubblicamente da ricercatori, docenti e appassionati. Questi notebook spaziano attraverso una vasta gamma di applicazioni, offrendo così agli utenti l'opportunità di esplorare e apprendere nuove tecniche e metodologie. Google Colab si afferma come uno strumento essenziale nel panorama dell'informatica moderna, rappresentando una risorsa per professionisti, accademici e appassionati che desiderano esplorare le frontiere dell'intelligenza artificiale e della data science con facilità, efficienza e senza onerosi investimenti in hardware.

4.3 BertTopic

Il topic modelling rappresenta una tecnica avanzata nell'ambito dell'elaborazione del linguaggio naturale (NLP), finalizzata a scoprire temi o "argomenti" nascosti all'interno di un insieme di documenti. BertTopic analizza le parole presenti nei testi per identificare modelli e connessioni che rivelano gli argomenti sottostanti. Ad esempio, un documento incentrato sull'apprendimento automatico utilizzerà probabilmente termini come "gradiente" e "embedding" più frequentemente rispetto a uno che tratta la panificazione. Ogni documento tende a coprire più argomenti in proporzioni variabili e, esaminando le statistiche delle parole, è possibile riconoscere cluster di parole correlate che rappresentano questi temi. Questo permette di analizzare un insieme di documenti determinando gli argomenti trattati e la distribuzione degli argomenti all'interno di ciascun documento. Con l'avvento di modelli basati su Transformer, il topic modelling ha compiuto un salto qualitativo, sfruttando rappresentazioni del testo più ricche rispetto alla semplice analisi lessicale. In questo contesto, BertTopic[2] emerge come una libreria Python all'avanguardia che semplifica il processo di topic modelling. BertTopic utilizza tecniche di embedding e c-TF-IDF per creare cluster densi che consento-

no di interpretare facilmente gli argomenti, mantenendo al contempo le parole importanti nelle descrizioni degli argomenti. Facile da avviare, BertTopic supporta una gamma di approcci avanzati al topic modelling, inclusi quelli guidati, supervisionati, semi-supervisionati e manuali. BertTopic si rivela uno strumento potente per chi desidera scoprire gli argomenti significativi all'interno di collezioni di testi. Che si tratti di analizzare recensioni dei clienti, esplorare articoli di ricerca o categorizzare articoli di giornale, BertTopic facilita l'estrazione di informazioni significative dai dati testuali, rendendolo uno strumento essenziale per chiunque voglia approfondire la conoscenza contenuta nei propri insiemi di dati. Per arrivare ad ottenere una rappresentazione dei topic BertTopic attiva 5 diverse fasi riportate in Fig. 17.

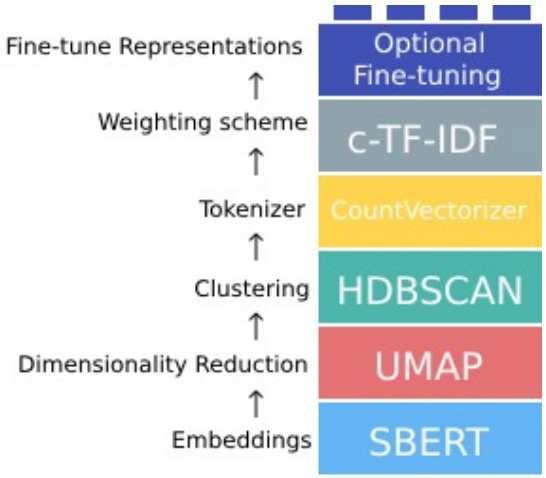


Figura 17: *Rappresentazione schematica del funzionamento di BertTopic*

Sebbene questi passaggi rappresentino il percorso standard, BertTopic offre una certa modularità rappresentata in Fig. 18. Ogni fase di questo processo è stata attentamente selezionata in modo che siano tutte relativamente indipendenti l'una dall'altra. Ad esempio, la fase di tokenizzazione non è direttamente influenzata dal modello di embedding utilizzato per convertire i documenti, ciò permette di poter scegliere il modo in cui eseguiamo la tokenizzazione.

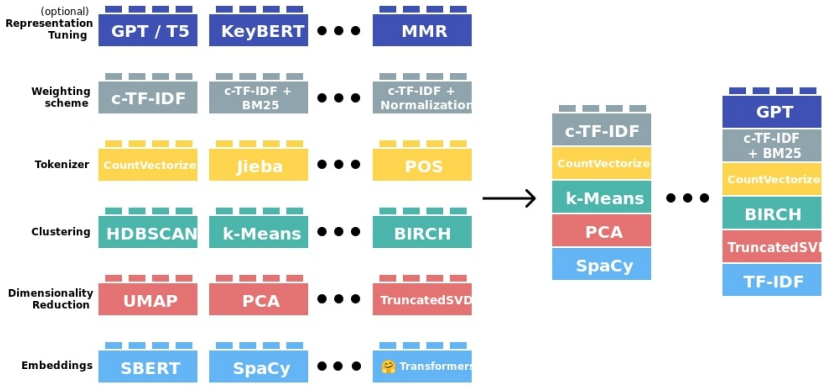


Figura 18: *Rappresentazione della modularità dei componenti di BertTopic*

Di seguito è riportata una panoramica di ogni componente della pipeline.

4.3.1 Embeddings

Il primo passo di BertTopic è trasformare i documenti di input in rappresentazioni numeriche, note come embeddings. Sebbene vi siano vari metodi per ottenere tale risultato, i sentence-transformers all-MiniLM-L6-v2[3] sono generalmente impiegati per la loro superiore abilità nell'identificare similitudini semantiche tra i documenti. Questi sentence trasformer impiegati da BertTopic Mappano frasi e paragrafi in uno spazio vettoriale denso di 384 dimensioni e possono essere utilizzati per compiti come il clustering o la ricerca semantica. Nonostante ciò, la natura modulare di BertTopic permette l'adozione di differenti strumenti per effettuare questo passaggio.

4.3.2 Dimensionality Reduction

Il secondo passo è la riduzione della dimensionalità degli embedding di input. Dato che gli embedding presentano frequentemente una elevata dimensionalità, il processo di clustering si rivela complesso. Una soluzione consiste nel ridurre la dimensionalità degli embedding a uno spazio vettoriale $\mathbb{R}^n$ ad uno spazio $\mathbb{R}^m$ con $m << n$ affinché gli algoritmi di clustering possano operare efficacemente. UMAP è utilizzato come predefinito in BertTopic poiché è capace di ridurre lo spazio ad alta dimensionalità sia locale che globale a dimensioni inferiori. Tuttavia, esistono altre soluzioni, come la Principal Component Analysis (PCA) ovvero una tecnica statistica utilizzata per ridurre la dimensionalità dei dati, preservando il più possibile

la variabilità presente nel dataset originale, che gli utenti potrebbero essere interessati a sperimentare. Poiché BertTopic permette di assumere una certa indipendenza tra le fasi, possiamo utilizzare qualsiasi altro algoritmo di riduzione della dimensionalità.

4.3.3 Clustering

Dopo la riduzione della dimensionalità degli embedding di input, è necessario raggrupparli in cluster di embedding simili per estrarre gli argomenti. Il processo di clustering è cruciale poiché l'efficacia della tecnica di clustering impiegata influisce direttamente sulla precisione delle rappresentazioni tematiche. In BertTopic, si tende comunemente ad utilizzare HDBSCAN, un algoritmo di clustering che estende DBSCAN trasformandolo in un algoritmo di clustering gerarchico, per la sua capacità di rilevare strutture con densità diverse. Tuttavia, non esiste un modello di clustering universale e si potrebbe preferire l'uso di un'alternativa completamente diversa a seconda del caso d'uso grazie alla modularità di BertTopic.

4.3.4 Tokenizer e Bag-of-word

Il tokenizer è un componente che gioca un ruolo cruciale nella creazione della rappresentazione bag of words. Il modello bag of words è una tecnica di rappresentazione del testo che trasforma un corpus di documenti in una matrice di frequenze di termini. Prima di poter contare le frequenze dei termini, è necessario quindi tokenizzare il testo. In questo modello, in cui ogni documento è rappresentato come un vettore di frequenze di termini utilizzando HDBSCAN come modello di clustering, possiamo presupporre che i nostri cluster presentino gradi di densità e forme diverse. Ciò implica che una tecnica di rappresentazione dei topic basata sul centroide potrebbe non essere il modello più adeguato. In altre parole, desideriamo una tecnica di rappresentazione dei topic che faccia poche o nessuna ipotesi sulla struttura attesa dei cluster. Per fare ciò, iniziamo combinando tutti i documenti di un cluster in un unico documento. Questo documento, molto esteso, rappresenta quindi il cluster. Successivamente, possiamo contare quante volte ogni parola appare in ciascun cluster. Questo processo genera ciò che viene chiamato una rappresentazione bag-of-words, nella quale è possibile trovare la frequenza di ciascuna parola in ogni cluster. Pertanto, questa rappresentazione bag-of-words opera a livello di cluster e non a livello di documento. Questa distinzione è fondamenta-

le poiché il nostro interesse si concentra sulle parole a livello di topic (ovvero, a livello di cluster). Utilizzando una rappresentazione bag-of-words, non viene formulata alcuna ipotesi riguardo alla struttura dei cluster. Inoltre, la rappresentazione bag-of-words viene normalizzata secondo la norma L1 per tenere conto dei cluster di dimensioni diverse. La norma L1 è la somma dei moduli dei vettori in uno spazio. Misura la distanza tra vettori come la somma delle differenze assolute tra le loro componenti, pesando ugualmente tutti i componenti del vettore.

4.3.5 Weighting Scheme

Nell'ambito dell'elaborazione linguistica con BertTopic, si adotta una strategia innovativa per analizzare e differenziare i cluster di documenti attraverso l'adattamento del metodo TF-IDF, trasformandolo in un approccio basato su classi piuttosto che su documenti singoli. La funzione di peso TF-IDF (term frequency–inverse document frequency) è una funzione utilizzata in information retrieval per misurare l'importanza di un termine rispetto ad un documento o ad una collezione di documenti. Questo processo inizia trasformando ogni cluster in un documento unificato, per poi analizzare la frequenza delle parole all'interno di tale aggregato. Attraverso questa metodologia, denominata TF-IDF basato su classi (c-TF-IDF), si ottiene una rappresentazione bag-of-words a livello di cluster, che viene poi normalizzata per tener conto delle varie dimensioni dei topic. L'obiettivo è di evidenziare le parole che caratterizzano distintamente ogni cluster, fornendo così una descrizione precisa dei topic rappresentati. Questo approccio modulare consente una rappresentazione più accurata e significativa dei topic, superando le limitazioni dei metodi tradizionali basati sui singoli documenti.

4.3.6 Fine-tune Topic representation

Le rappresentazioni c-TF-IDF forniscono un insieme di termini descrittivi per collezioni di documenti, offrendo un metodo efficace per la generazione rapida di rappresentazioni tematiche precise. Nel dinamico ambito del Natural Language Processing, l'introduzione continua di nuove metodologie apre la possibilità di affinare questi topic attraverso l'uso di tecnologie avanzate come GPT, T5, KeyBERT e Spacy, tutte accessibili tramite BertTopic. Più specifi-

camente, possiamo considerare i topic generati con c-TF-IDF come topic candidati. Ciascuno di essi contiene un insieme di parole chiave e un riferimento ai documenti rappresentativi che possiamo utilizzare per affinare ulteriormente le rappresentazioni dei topic. Disporre di un set di documenti rappresentativi per ciascun topic è un grande vantaggio poiché consente di effettuare il fine-tuning su un numero ridotto di documenti. Questo riduce il carico computazionale per i modelli di grandi dimensioni, che devono operare solo su quel piccolo insieme di documenti rappresentativi per ciascun topic. Di conseguenza, modelli di linguaggio di grandi dimensioni come GPT e T5 diventano più gestibili in ambienti di produzione e richiedono generalmente meno tempo rispetto ai passaggi di riduzione della dimensionalità e clustering.

4.4 Llama2

Llama2[5] è un'innovativa serie di modelli linguistici avanzati di Meta, resa disponibile tramite Hugging Face con una licenza aperta che ne consente l'uso commerciale. Questi modelli rappresentano un notevole passo avanti nella tecnologia AI, offrendo miglioramenti significativi rispetto alla generazione precedente, come l'addestramento su una quantità maggiore di dati, una lunghezza di contesto estesa e velocità di inferenza migliorate. Particolarmente degne di nota sono le versioni ottimizzate per il dialogo, che mostrano prestazioni eccellenti in termini di utilità e sicurezza. Llama2 si integra perfettamente nell'ecosistema Hugging Face, supportando una vasta gamma di applicazioni grazie alla sua flessibilità e alle diverse dimensioni dei modelli disponibili, dal 7B al 70B di parametri. La sua licenza open source apre nuove possibilità di utilizzo in vari settori, promuovendo l'innovazione e l'accesso aperto alle tecnologie AI di punta.

4.5 Llama Index

Il framework LlamaIndex[6] si posiziona come soluzione all'avanguardia nel panorama delle applicazioni basate sui Modelli di Linguaggio di Grandi Dimensioni (LLM), inaugurando una nuova fase nell'integrazione dei dati e nel potenziamento delle applicazioni. La sua essenza risiede nella capacità di affrontare una sfida fondamentale: nonostante l'ampia formazione su vasti dataset, gli LLM non sono intrinsecamente predisposti all'utilizzo dei dati specifici del-

l'utente. La metodologia RAG (Retrieval-Augmented Generation) è il fulcro di Llama Index, facilitando l'incorporazione dei dati utente nel vasto database di conoscenza degli LLM. Questo processo arricchisce le risposte del modello con informazioni pertinenti e personalizzate, migliorando significativamente l'accuratezza e la rilevanza degli output. Il processo di elaborazione realizzato da Llama Index è articolato in cinque fasi cruciali (Fig. 19): Caricamento, Indicizzazione, Memorizzazione, Interrogazione e Valutazione. Ogni fase è accuratamente progettata per facilitare una transizione fluida dei dati dalla loro origine al loro utilizzo finale nel generare risposte informate. Il framework fornisce un insieme robusto di strumenti per ogni passaggio, iniziando dalla fase di caricamento, dove i dati sono raccolti da varie fonti attraverso numerosi connettori. Segue la fase di indicizzazione, che implica la creazione di embedding vettoriali e metadati per organizzare efficientemente i dati. Una volta indicizzati, i dati vengono memorizzati per un facile recupero, fondamentale per la fase di interrogazione, in cui si ricercano informazioni specifiche. Infine, la fase di valutazione misura l'efficacia del risultato prodotto, garantendo il mantenimento di elevati standard di accuratezza e velocità. Elementi chiave del framework Llama Index sono le sue funzionalità avanzate, inclusi Nodi e Documenti per l'organizzazione dei dati, Connettori per l'ingestione dei dati e componenti specializzati per il recupero dei dati e la sintesi delle risposte. Nei paragrafi successivi vengono approfonditi i dettagli dei componenti del framework.

Figura 19: *Fasi di Lama Index*

4.5.1 Caricamento

La fase di Caricamento è il primo passo, dove i dati vengono raccolti da svariate fonti mediante l'utilizzo di numerosi connettori. Questo stadio prepara il terreno affinché i dati possano essere elaborati e resi utilizzabili. Le strutture che vengono utilizzate in questa fase per organizzare i dati sono i documenti e i nodi.

Un **Documento** nel contesto di Llama Index si configura come un contenitore poliedrico, destinato ad accogliere una molteplicità di fonti informative, che spaziano da file PDF, a

risposte di API, fino a dati ricavati da database. La progettazione dei Documenti mira a una notevole flessibilità; essi possono essere elaborati manualmente dall'utente oppure generati in maniera automatica attraverso i meccanismi di caricamento dati integrati in Llama Index. Al suo nucleo, il Documento è preminentemente un depositario di testo, tuttavia possiede la capacità di incorporare una serie di attributi supplementari che ne potenziano il valore informativo. Tra questi attributi si annoverano i metadati - un compendio di annotazioni volto a impreziosire il testo con un contesto aggiuntivo o con note specifiche - e le relazioni, ossia l'insieme delle interconnessioni tra un dato Documento e altri Documenti o Nodi all'interno dell'ecosistema.

Il **Nodo** è invece concepito come un segmento distinto di un Documento sorgente, che si manifesta come una porzione di testo, un'immagine o qualsivoglia altra forma di dato contenuto nel Documento. Riflettendo la struttura dei Documenti, i Nodi incorporano non soltanto il contenuto principale, ma anche uno strato ricco di metadati e informazioni relazionali, stabilendo collegamenti tra essi e altri Nodi all'interno del sistema. I Nodi sono considerati entità di estrema importanza in Llama Index, trattati con lo stesso livello di priorità e funzionalità dei Documenti dai quali derivano. Gli utenti hanno la libertà di definire con precisione le caratteristiche di ciascun Nodo, personalizzando i suoi attributi per soddisfare le esigenze specifiche della loro architettura di dati o degli obiettivi di analisi. In alternativa, è possibile generare dinamicamente i Nodi sezionando i Documenti sorgenti ed individuando i Nodi che li compongono un processo agevolato dalle classi NodeParser. Questo metodo di creazione dei Nodi assicura un'ereditarietà fluida dei metadati dal Documento genitore, mantenendo una coerenza informativa trasversale all'intero dataset. Ad esempio, un attributo come "file_name", presente nel Documento originale, diventa automaticamente una proprietà condivisa tra tutti i suoi Nodi derivati, preservando un legame chiaro e tracciabile con il materiale sorgente originario.

Per generare i nodi a partire dai documenti, Lama Index mette a disposizione un componente che prende il nome di `NodeParser`. Questo componente prende in input un elenco di documenti e li suddivide in oggetti Nodo, in modo tale che ogni nodo costituisca un frammento specifico del documento genitore. Quando un documento viene frammentato in nodi, tutti i suoi attributi vengono ereditati dai nodi figli (ad esempio, metadati, testo e modelli di meta-

dati, ecc.). Esistono diversi moduli del NodeParser progettati per elaborare e suddividere i documenti in oggetti Nodo basati su diversi tipi di contenuti e necessità di analisi. Di seguito è riassunta una panoramica dei principali parser di nodi disponibili e delle loro funzionalità distintive:

SimpleFileNodeParser

È un componente che suddivide un documento caricato da un file in nodi rilevando automaticamente il NodeParser da utilizzare in base al tipo di file. Questo modulo facilita l'automatizzazione nella scelta del parser di nodi più adatto per ogni tipo di contenuto (JSON, Markdown, ecc.).

HTMLNodeParser

Utilizza beautifulsoup (un pacchetto Python per l'analisi di documenti HTML e XML) per analizzare l'HTML grezzo, consentendo di selezionare un sottoinsieme di tag HTML per il parsing. È personalizzabile, consentendo di definire i tag specifici da considerare. Lo schema del funzionamento è riportato in Fig. 20

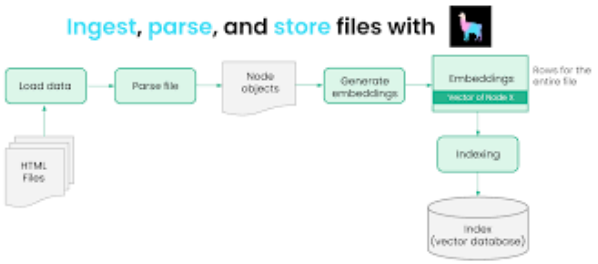


Figura 20: *HTMLNodeParser*

JSONNodeParser

Questo parser elabora JSON grezzo, trasformandolo in nodi che possono essere facilmente manipolati e analizzati all'interno di Llama Index.

MarkdownNodeParser

Analizza il testo markdown grezzo, suddividendolo in nodi basati sul contenuto del markdown, facilitando l'elaborazione e l'analisi di documenti in formato Markdown.

CodeSplitter

Questo splitter divide il testo del codice basandosi sul linguaggio di programmazione in cui è scritto, utile per l'elaborazione di documenti contenenti codice sorgente.

LangchainNodeParser

Permette di trasformare qualsiasi spiltter di testo definito in Langchain (un framework per lo sviluppo di applicazioni basate su LLM) in un parser di nodi, offrendo flessibilità nell'elaborazione del testo.

SentenceSplitter

Tenta di dividere il testo rispettando i confini delle frasi, utile per l'elaborazione di testi in cui la struttura delle singole frasi è cruciale.

SentenceWindowNodeParser

Simile ad altri parser di nodi, ma suddivide tutti i documenti in frasi singole, includendo nel metadato di ogni nodo una "finestra" di frasi circostanti. Questo è particolarmente utile per la generazione di embedding con uno scopo molto specifico.

SemanticSplitterNodeParser

Introduce il concetto di "chunking semantico", selezionando adattivamente i punti di interruzione tra le frasi usando la similarità degli embedding per assicurare che un "chunk" contenga frasi semanticamente correlate.

TokenTextSplitter

Tenta di dividere in chunk di dimensioni consistenti in base al conteggio dei token grezzi, adatto per l'elaborazione di testi in cui il conteggio dei token è un fattore determinante.

HierarchicalNodeParser

Questo parser suddivide i nodi in nodi gerarchici, cioè un singolo input viene diviso in diverse gerarchie di dimensioni di chunk, con ogni nodo che contiene un riferimento al suo nodo genitore. Questo è utile quando combinato con il AutoMergingRetriever che consente di recuperare informazioni anche dai nodi vicini per fornire al modello LLM un contesto più completo.

4.5.2 Indicizzazione

Un Indice in Llama Index rappresenta una struttura dati essenziale che consente il recupero rapido del contesto rilevante per una query utente, costituendo la base fondamentale per le applicazioni RAG. Questa tecnologia si rivela cruciale nell'implementazione di motori di query e di chat, facilitando l'interazione domanda e risposta nonché la conversazione diretta con i dati. Gli Indici sono costruiti a partire dai Documenti e svolgono un ruolo chiave nella creazione di sistemi capaci di navigare efficacemente su grandi moli di informazioni. Al loro interno, memorizzano dati in oggetti Nodo, che rappresentano segmenti dei documenti originali, e offrono un'interfaccia Retriever che supporta configurazioni e automazioni aggiuntive. Il tipo di indice più diffuso in Llama Index è il VectorStoreIndex, che rappresenta il punto di partenza ideale per chi si avvicina a queste tecnologie. Per esplorare altre tipologie di indici e comprendere quale possa meglio adattarsi alle proprie esigenze, Llama Index mette a disposizione una guida dettagliata sull'utilizzo di ciascun indice.

VectorStoreIndex

Il Vector Store Index, illustrato in Fig. 21 è un componente chiave nell'ecosistema di Llama Index, essenziale per le applicazioni RAG. Questo tipo di indice permette di archiviare e recuperare rapidamente il contesto pertinente per le query degli utenti, fungendo da fondamento per la costruzione di motori di query e di chat di Llama Index. Esistono diversi modi di usare il Vector Store Index in base a quanto controllo si vuole avere sulla costruzione dell'indice stesso. Il Vector Store Index trasforma l'intero corpus testuale in embedding mediante l'utilizzo di un'API fornita dal modello di linguaggio (LLM) scelto. La stessa cosa è fatta in fase di interrogazione, dove la query viene convertita a sua volta in embeddings in modo

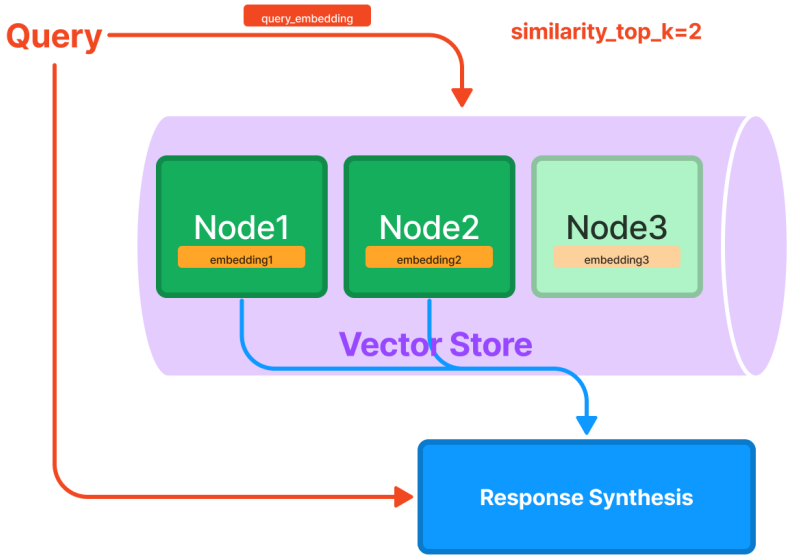


Figura 21: *Vector Store Index*

da classificare tutti gli elementi contenuti nell'indice in base alla loro somiglianza semantica con la query utente. Una volta che la classificazione è ultimata, il VectorStoreIndex restituisce gli embeddings più simili come chunk di testo. Il numero di embeddings che ritorna è indicato con `k` e di conseguenza il parametro che controlla quanti embeddings devono essere ritornati è indicato con `top_k`. questa tipologia di ricerca è indicata "top-k semantic retrival" ed è la forma più semplice di interrogare un vector index. Una volta che l'indice è stato costruito, questo va memorizzato. In maniera predefinita, il VectorStoreIndex conserva i dati in memoria volatile; tuttavia, è configurabile per adottare soluzioni di memorizzazione persistente.

4.5.3 Interrogazione

Una volta che i dati sono stati caricati e che l'indice è stato costruito e memorizzato è possibile procedere con la fase di interrogazione. L'oggetto che Llama Index mette a disposizione e che implementa l'interrogazione è il `QueryEngine` ma è anche possibile utilizzare i componenti che esso utilizza presi singolarmente se lo si desidera. Questo oggetto prende in ingresso una query scritta in linguaggio naturale e ritorna una risposta arricchita dai dati. Molto spesso il query engine è costruito su uno o più indici a cui si accede tramite dei retriever. Llama Index mette a disposizione sia API di alto che di basso livello per interagire con il query engine in base a quanto controllo si vuole avere sui suoi componenti. Per casi di utilizzo specifici è anche possibile andare a costruire un query engine completamente custom. Le tappe principali di questa fase sono riportate di seguito e ad ogniuna di esse corrisponde ad un componente.

Retrieval

Consiste nel recuperare i documenti più rilevanti per la query dall'indice. Il componente che la implementa è il Retriever e, come discusso in precedenza, la procedura più comune di recupero è quella basata su top-k. Il retriever può anche essere definito indipendentemente ma in genere è un componente che si posiziona sopra l'indice e proprio per questo motivo sono strettamente correlati l'uno con l'altro. Per quanto riguarda il retriever associato al vector store index, questo procede a recuperare i chunk dei documenti che soddisfano i requisiti di similarietà tra gli embeddings dei chunk e quelli della query.

Postprocessing

Il postprocessing dei nodi è una tappa opzionale che si colloca tra quella di retrieval e quella di response synthesis. L'obbiettivo è quello di filtrare i nodi che vengono recuperati secondo diversi criteri che vanno dallo scartare i nodi che hanno un valore di `similarity_score` sotto una certa soglia allo scartare quelli che presentano o non presentano determinate parole chiave. Il componente che lo implementa è il node post processor che è disponibile in tre diverse varianti:

- `KeywordNodePostprocessor`: filtra i nodi per parole chiavi richieste o escluse.

- `SimilarityPostprocessor`: filtra i nodi andando a settare una soglia sul similarity score (supportato solo da retriever basati sugli embedding).

- `PrevNextNodePostprocessor`: aumenta il contenuto informativo dei nodi recuperati con le informazioni contestuali dei nodi posizionati nell'intorno.

Response synthesis

Questo step è quello che genera la risposta dal modello di linguaggio a partire da una query e da una serie di chunk di testo. L'output del response synthesizer è un oggetto di tipo response e si può ottenere in diversi modi, più o meno complessi, caratterizzati da come la query e i chunk di testo recuperati dal retriever vengono passati al response synthesizer utilizzando uno o più prompt template. Quando un response sinthesizer viene utilizzato all'interno di un query engine, questo riceve i nodi subito dopo che sono stati recuperati, senza passare per la fase di post processing. Le diverse modalità di produzione di un response object da parte del response synthesizer prendono il nome di response mode.

- `refine`: Crea e affina una risposta effettuando una chiamata all'LLM per ogni chunk di testo recuperato dal retriever, utilizzando dei template prompt. I template prompt sono strutture predefinite che vengono popolati con le informazioni contenute nei nodi e con le richieste dell'utente prima di essere inviati all'LLM. In particolare, come illustrato nei listati 13 e 14, i template prompt `text_qa_template` e `refine_template` sono utilizzati nella modalità refine. In questa modalità, il primo chunk di testo e la query dell'utente vengono inseriti rispettivamente nei placeholder `{context_str}` e `{query_str}` all'interno del `text_qa_template`, e poi inviati all'LLM. Il risultato ottenuto viene quindi inserito nel `refine_template` al posto del placeholder `{existing_answer}`, insieme alla query dell'utente e al secondo chunk di testo, nei corrispondenti placeholder, e inviato nuovamente all'LLM. Questo processo continua per tutti i restanti k chunk di testo rimanenti.

- `compact`: simile alla modalità `refine` ma concatena i chunk fino a raggiungere la dimensione massima del prompt per eseguire meno chiamate all'LLM.

- `tree_sumarize`: Dato un insieme di oggetti di tipo nodo e la query, costruisce un albero in maniera ricorsiva e ritorna il nodo radice come risposta. Interroga l'LLM utilizzando il `DEFAULT_SUMMARY_PROMPT_TMPL`. Ottimo per eseguire riassunti.
  Il `DEFAULT_SUMMARY_PROMPT_TMPL` riportata nel listato 1 è sempre un template prompt come i precedenti ma presenta la particolarità di richiedere all'LLM il riassunto del chunk di testo fornito in input.

- `no_text`: Esegue solo il retriever per selezionare i nodi che poi vanno mandati all'LLM sensa effettivamente mandarli. Questi possono essere ispezionati accedendo alla proprietà `response.source_nodes` dell'oggetto `response`.

- `accumulate`: Dato un set di nodi e la query, applica la query ad ogni nodo e accumula le risposte in un array. Alla fine ritorna una stringa composta da tutte le risposte ottenute concatenate insieme. Buono quando si deve eseguire la stessa query su più chunk di testo.

```
1  DEFAULT_SUMMARY_PROMPT_TMPL = (
2      "Write a summary of the following. Try to use only the "
3      "information provided. "
4      "Try to include as many key details as possible.\n"
5      "\n"
6      "\n"
7      "{context_str}\n"
8      "\n"
9      "\n"
10     'SUMMARY:"""\n'
11 )
12
```

Listato 1: Default prompt per la tree summarize mode di LLama Index

Capitolo 5

Applicazione sviluppata

5.1 Dataset

La scelta del dataset da usare per sviluppare il progetto è ricaduta su **CShorten/ML-ArXiv-Papers**[4]. Come riportato nella pagina descrizione di HuggingFace, questo dataset è un sottoinsieme di una dataset più grande (ArXiv[1]) che contiene milioni di articoli scientifici. Nello specifico, il sottoinsieme scelto è composto da soli abstract e titoli di articoli che parlano di Machine Learning. Tale dataset ridotto contiene 117.592 articoli. Nel listato 2 è riportato un esempio di articolo.

```
The ability to combine known skills to create new ones may be crucial in the
solution of complex reinforcement learning problems that unfold over extended
periods. We argue that a robust way of combining skills is to define and
manipulate them in the space of pseudo-rewards (or "cumulants"). Based on this
premise, we propose a framework for combining skills using the formalism of
options. We show that every deterministic option can be unambiguously
represented as a cumulant defined in an extended domain. Building on this
insight and on previous results on transfer learning, we show how to
approximate options whose cumulants are linear combinations of the cumulants of
known options. This means that, once we have learned options associated with a
set of cumulants, we can instantaneously synthesise options induced by any
linear combination of them, without any learning involved. We describe how this
framework provides a hierarchical interface to the environment whose abstract
actions correspond to combinations of basic skills. We demonstrate the
practical benefits of our approach in a resource management problem and a
navigation task involving a quadrupedal simulated robot.
```

Listato 2: Esempio di abstract

Per abbassare i tempi di computazione, si è reso necessario eseguire un sampling randomico di ML-ArXiv dello 0,5 passando a 58.796 elementi. Il notebook che realizza questo è `dataset/generazione_dataset_da_ML-ArXiv-Papers.ipynb`. Il risultato di questo script è il dataset pronto per essere utilizzato dalle fasi seguenti ed è `ML-ArXiv/ML-ArXiv_58796.json`. La fase successiva è suddividere tutti gli abstract contenuti all'interno del dataset per topic. Per poterlo fare è necessario eseguire prima l'estrazione dei topic con BertTopic, descritta nella sezione seguente, e poi processare il risultato con lo script

`dataset/generazione_dataset_da_ML-ArXiv-Papers.ipynb`

per creare una struttura di directory (Fig. 22) che organizzi ogni abstract all'interno della cartella relativa al topic a cui appartiene.

5.2 Legacy

Successivamente alla definizione del dataset, il primo passo da eseguire per arrivare ad ottenere una serie di etichette per ogni topic risulta l'estrazione dei topic stessi dal dataset. Il notebook che implementa quanto descritto è stato recuperato in rete e prende il nome di "Topic Modeling with Llama2"[8]. Questo notebook è strutturato in due parti nonostante i

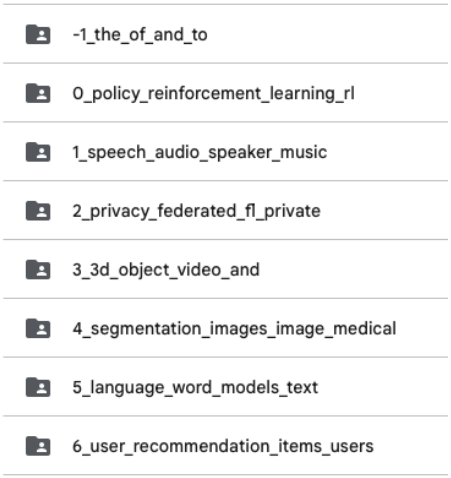


Figura 22: *Struttura a directory prodotta*

due processi vengano attivati dalla stessa istruzione. La prima parte sfrutta BertTopic per estrarre i topic da un dataset specificato, mentre la seconda sfrutta il modello di linguaggio Llama2 per estrarre delle etichette per descrivere ogni topic. Il notebook originale è stato leggermente modificato per adattarlo alle necessità del progetto. Una prima modifica ha riguardato la fase di caricamento del dataset (listato 3) in quanto nel progetto si fa uso di una versione ridotta di ML-ArXiv rispetto alla versione originale del notebook.

```
dataset_path = "/content/drive/MyDrive/Tirocinio_Magistrale/dataset/ML-ArXiv/ML-
    ArXiv_58796.json"

import pandas as pd
df = pd.read_json(dataset_path, lines=True)
abstracts = df.Abstracts.tolist()
```

Listato 3: Caricamento dataset

Un'ulteriore modifica riguarda il modo in cui è presentato l'output. Nello script originale vi erano una serie di blocchi per presentare i risultati tramite dei diagrammi e delle mappe. Questi blocchi sono stati rimossi e sostituiti con due chiamate metodi che riversano i risultati in file CSV in modo da poterli utilizzare nelle fasi successive. Le istruzioni per eseguire il salvataggio sono riportate nel listato 4.

```
# Save data
topic_model.get_topic_info().to_csv('/content/drive/MyDrive/Tirocinio_Magistrale/
    TopicModellingLLama2/RISULTATI/topic_labels_EX0.csv')
topic_model.get_document_info(abstracts).to_csv('/content/drive/MyDrive/
    Tirocinio_Magistrale/TopicModellingLLama2/RISULTATI/docs_info.csv')
```

Listato 4: Salvataggio dati

Lo script modificato è `TopicModellingLLama2/Topic_Modeling_with_Llama2_custom.ipynb`.

5.2.1 Llama2

Come precedentemente accennato, nonostante l'estrazione dei topic e la generazione delle etichette siano due fasi ben distinte, vengono di fatto attivate entrambe con una singola chiamata. Per arrivare quindi a questa chiamata è necessario definire prima tutti i componenti utilizzati. Il primo componente che viene definito è il modello di linguaggio da utilizzare. In questo caso viene utilizzato Llama2 con una appropriata configurazione di quantizzazione (Listato 5).

```
bnb_config = transformers.BitsAndBytesConfig(
    load_in_4bit=True,  # 4-bit quantization
    bnb_4bit_quant_type='nf4',  # Normalized float 4
    bnb_4bit_use_double_quant=True,  # Second quantization after the first
    bnb_4bit_compute_dtype=bfloat16  # Computation type
)
```

Listato 5: Configurazione quantizzazione

Definito il modello di linguaggio si passa alla fase di definizione del prompt. In questo caso l'approccio scelto prevede la composizione di un prompt in diverse componenti (Listato 6):

- `system_prompt`
- `user_prompt` ulteriormente suddiviso in
  - `example_prompt`
  - `main_prompt`

```
system_prompt = """
<s>[INST] <<SYS>>
You are a helpful, respectful and honest assistant for labeling topics.
<</SYS>>
"""

example_prompt = """
I have a topic that contains the following documents:
- Traditional diets in most cultures were primarily plant-based with a little meat on top, but with the rise of industrial style meat production and factory farming, meat has become a staple food.
- Meat, but especially beef, is the word food in terms of emissions.
- Eating meat doesn't make you a bad person, not eating meat doesn't make you a good one.

The topic is described by the following keywords: 'meat, beef, eat, eating, emissions, steak, food, health, processed, chicken'.

Based on the information about the topic above, please create a short label of this topic. Make sure you to only return the label and nothing more.

[/INST] Environmental impacts of eating meat
"""

main_prompt = """
[INST]
I have a topic that contains the following documents:
[DOCUMENTS]

The topic is described by the following keywords: '[KEYWORDS]'.

Based on the information about the topic above, please create a short label of this topic. Make sure you to only return the label and nothing more.
[/INST]
"""
```

Listato 6: Definizione dei prompt

La metodologia di prompt engineering adottata dagli autori del notebook implica l'utilizzo della `system_prompt` per comunicare chiaramente al modello di linguaggio l'obiettivo specifico da raggiungere, ovvero la generazione di etichette. Questo approccio guida fin dall'inizio il modello verso il risultato desiderato.

La scelta di specificare l'utilizzo di una `example_prompt` per il prompt engineering evidenzia un'accurata strategia di progettazione, dimostrando un'intenzionale direzione nell'interazione con il modello di linguaggio. Questa decisione sottolinea l'importanza di fornire al modello indicazioni precise sul compito da svolgere, attraverso esempi che delineano chiaramente l'aspettativa di output. L'adozione di una `example_prompt` non solo facilita la comprensione del modello riguardo l'attività richiesta ma garantisce anche che l'elaborazione delle informazioni sia allineata con gli obiettivi prefissati. Il prompt finale sarà quindi composto come segue:

```
prompt = system_prompt + example_prompt + main_prompt
```

5.2.2 BertTopic

Definito il modello di linguaggio bisogna definire i componenti di BertTopic (Listato 7). Il primo passo riguarda l'estrazione degli embeddings. Viene selezionato il modello `BAAI/bge-small-en` e successivamente si procede a definire i sottomodelli per eseguire il clustering nello spazio degli embeddings.

```
# Pre-calculate embeddings
embedding_model = SentenceTransformer("BAAI/bge-small-en")
embeddings = embedding_model.encode(abstracts, show_progress_bar=True)

#define all sub-models in BERTopic and do some small tweaks to the number of clusters to be created, setting random states, etc.
umap_model = UMAP(n_neighbors=15, n_components=5, min_dist=0.0, metric='cosine', random_state=42)
hdbscan_model = HDBSCAN(min_cluster_size=150, metric='euclidean', cluster_selection_method='eom', prediction_data=True)

# Pre-reduce embeddings for visualization purposes
reduced_embeddings = UMAP(n_neighbors=15, n_components=2, min_dist=0.0, metric='cosine', random_state=42).fit_transform(embeddings)
```

Listato 7: Parametri BertTopic

Questo consente di individuare gruppi di parole semanticamente vicine idealmente associate allo stesso topic. Come ultimo passo prima di lanciare il fit del modello si definisce l'oggetto `topic_model` come riportato nel listato 8. Successivamente alla fase di fit si procede

quindi ad estrarre il risultato del processo di etichettatura per ogni topic all'interno del file `topic_labels_EX0.csv` e la lista di tutti i documenti contenuti nel dataset con associato il topic assegnato da BertTopic all'interno del file `docs_info.csv`. Le label per i topic estratte da questo procedimento vengono identificate con il nome EX0 per indicare che si tratta dell'esperimento preliminare per estrarre le etichette.

```

from bertopic import BERTopic

topic_model = BERTopic(

  # Sub-models
  embedding_model=embedding_model,
  umap_model=umap_model,
  hdbscan_model=hdbscan_model,
  representation_model=representation_model,

  # Hyperparameters
  top_n_words=10,
  verbose=True
)

# Train model
topics, probs = topic_model.fit_transform(abstracts, embeddings)
```

Listato 8: Oggetto topic_model

5.3 Llama Index su dataset intero

Ultimato l'esperimento preliminare si procede a mettere in piedi un nuovo insieme di esperimenti che sfruttano uno strumento più raffinato come Llama Index per estrarre le etichette dai topic. L'idea è quella di sfruttare la capacità di Llama Index di recuperare documenti rilevanti per il contesto della query per fare in modo che il modello di linguaggio produca un'etichetta migliore rispetto a quella che produce nelle condizioni dello script legacy avendo a disposizione solo i tre documenti più rilevanti secondo BertTopic per ogni topic. Nell'ambito di questa ricerca, sono state delineate due distinte serie di esperimenti con l'obiettivo di approfondire la comprensione e l'efficacia dei componenti di Llama Index. La prima serie di esperimenti

mira primariamente all'acquisizione di confidenza e alla comprensione approfondita delle funzionalità di Llama Index, avvalendosi dell'intero corpus documentale senza limitarsi a singoli topic. Questo approccio è stato intenzionalmente scelto per evitare l'introduzione di ulteriori livelli di complessità, facilitando così un'apprendimento progressivo dei meccanismi di base della piattaforma. Contrariamente, la seconda serie di esperimenti, descritta più dettagliatamente nelle sezioni successive, si propone di affinare la precisione dell'analisi mediante la selezione di documenti esclusivamente pertinenti ai 20 topic specifici. In questo stadio, si ripetono sostanzialmente le operazioni condotte nella fase iniziale, ma con un'enfasi particolare sulla discriminazione tematica, allo scopo di valutare la capacità di Llama Index di generare etichette esaustive ed accurate per ciascun argomento trattato. Tale strutturazione bipartita degli esperimenti permette non solo di costruire una solida base di conoscenza sui componenti e sul funzionamento di Llama Index ma anche di sperimentare e verificare la sua adattabilità e precisione nell'elaborazione di dati tematicamente ristretti, fornendo così un confronto metodico tra capacità generali e specifiche della piattaforma. Nel contesto di un approccio evoluto rispetto alle metodologie tradizionali, Llama Index necessita di una sequenza ampliata di operazioni per l'elaborazione delle etichette. Prima ancora di iniziare con la costruzione dell'indice, il passo iniziale è definire il modello di linguaggio con i rispettivi parametri da utilizzare per tutte le fasi successive, dalla costruzione dell'indice alla generazione delle etichette (Listato 9). In questo progetto si è scelto di utilizzare il modello di linguaggio **meta-llama/Llama-2-7b-chat-hf** con 7 miliardi di parametri con il corrispettivo tokenizer, una quantizzazione `nf4` su 4 bit, modello di embeddings `local:BAAI/bge-small-en-v1.5` e come query wrapper `"<s> [INST] {query_str} [/INST] "`. La configurazione dell'LLM è praticamente la stessa del caso precedente per poter comparare i risultati ottenuti.

```
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
)

llm = HuggingFaceLLM(
    model_name="meta-llama/Llama-2-7b-chat-hf",
    tokenizer_name="meta-llama/Llama-2-7b-chat-hf",
    query_wrapper_prompt=PromptTemplate("<s> [INST] {query_str} [/INST] "),
    context_window=3900,
    model_kwargs={"token": hf_token, "quantization_config": quantization_config},
    tokenizer_kwargs={"token": hf_token},
    device_map="auto",
)

service_context = ServiceContext.from_defaults(llm=llm, embed_model="local:BAAI/bge-
    small-en-v1.5")
```

Listato 9: Configurazione LLama2

Il query wrapper è il testo che viene inviato al modello di linguaggio che fa da corredo alla query string prodotta da Llama Index. All'interno della stringa del query wrapper troviamo il tag `<s>` che rappresenta l'inizio dell'interrogazione al modello di linguaggio, ha un corrispettivo tag di chiusura `</s>` che viene prodotto dal modello al termine dopo la response, e i tag `[INST]` e `[/INST]` che delimitano le istruzioni che vengono impartite al modello. Per scaricare ed utilizzare il modello è stato richiesto l'accesso a Meta tramite la piattaforma **HuggingFace**, la quale, ad autorizzazione concessa, fornisce un token da utilizzare nel codice (`hf_token`). Tale configurazione di Llama2 viene ripetuta in tutti gli esperimenti.

5.3.1 Costruzione indice

La costruzione dell'indice è un procedimento proprio di Llama Index che consente di produrre una struttura dati su cui eseguire la ricerca e la selezione dei documenti da inserire all'interno della query da fornire al modello di linguaggio. Questo procedimento va svolto in ogni caso, indipendentemente dai componenti che si utililzzano nelle fasi succes-

sive. Per entrambe le famiglie di esperimenti è stato scelto di utilizzare il Vector Store Index. Il notebook che implementa la costruzione dell'indice su tutta la base documentale è `LLAMA_INDEX/indice/costruzione_indice.ipynb`. Questo notebook prende in ingresso `ML-ArXiv/ML-ArXiv_58796.json` e lo carica all'interno di un dataframe pandas. Successivamente vengono estratti solo gli abstract all'interno di una lista python che viene poi sfruttata per creare sia l'indice basato sui soli testi che quello basato sulle frasi (Listato 10).

```
from IPython.display import clear_output
import numpy as np

index = VectorStoreIndex([])
doc_chunks = []
for i, text in enumerate(text_chunks):
    doc = Document(text=text, id_=f"doc_id_{i}")
    doc_chunks.append(doc)

for r in range(0,len(doc_chunks)):
  clear_output(wait=True)
  index.insert(doc_chunks[r])
  print("Current progress:", np.round(r/len(doc_chunks) * 100,2),"%")

index.storage_context.persist(persist_dir=index_text_path)
```

Listato 10: Generazione indice basato su testi

Per la creazione dell'indice basato su testi interi il procedimento è relativamente semplice. Si costruisce un oggetto di tipo `Document` per ogni abstract, costituito dal testo e da un indice univoco progressivo e si inserisce ogni documento all'interno di una lista. Successivamente si provvede a sfruttare il metodo `insert()` della classe `VectorStoreIndex` per inserirlo all'interno dell'indice. Questo notebook implementa parallelamente anche la costruzione dell'indice basato sulle frasi. La descrizione del procedimento in dettaglio è riportato nelle sezioni successive.

5.3.2 Estrazione etichette

I diversi processi relativi all'estrazione delle etichette per ogni topic, a seconda delle diverse modalità, prendono il nome di esperimenti. Ogni esperimento è identificato da un codice. La tabella 1 descrive le modalità di ogni esperimento.

CODICE	PROCESSO	DATASET	K	SYNTHESIZER
EX0	Legacy	ML-ArXiv_58796	3	/
EX1	Llama Index	ML-ArXiv_58796	3	refine
EX2	Llama Index	ML-ArXiv_58796	3	tree_summarize
EX3	Llama Index	ML-ArXiv_58796_frasi	21	refine
EX4	Llama Index	ML-ArXiv_58796_frasi	21	tree_summarize

Tabella 1: *Tabella degli esperimenti*

Il primo esperimento denominato `EX0` riguarda l'estrazione delle etichette eseguita tramite il procedimento legacy. I rimanenti espereimenti riguardano Llama Index e nello specifico le modalità operative `refine` e `tree_summarize` sui dataset costituiti dai testi interi e dalle frasi di questi ultimi. Come descritto nel paragrafo 4.5.3, la fase di interrogazione eseguita tramite query engine è composta dalla fase di retrieval e dalla fase di response synthesis. La fase di retrieval riguarda il recupero dei nodi dall'indice da parte del retriever (Listato 11).

```
response_ex1 = RetrieverQueryEngine(
  retriever=VectorIndexRetriever(
      index=index_text,
      similarity_top_k=3,
  ),
```

Listato 11: Retreiver per l'esperimento EX1

Il parametro che definisce quanti nodi elementi recuperare dall'indice prende il nome di `top_k`. Nella tabella 1 la colonna `K` indica appunto il numero di elementi che vengono inseriti all'interno del prompt. Come si può osservare, per gli esperimenti basati su Llama Index riguardanti il dataset composto da testi interi, è stato scelto lo stesso numero di testi utilizzati nel procedimento legacy. Questa decisione è stata adottata allo scopo di garantire che i risultati emergessero dalla medesima quantità di informazione, rendendoli così direttamente confrontabili.

Response synthesizer in refine mode

Caricato correttamente l'indice e selezionati i topic e le rispettive parole chiave di cui si desidera generare le etichette, rimangono da configurare i parametri degli esperimenti. Per quanto

riguarda l'esperimento `EX1` il response synthesizer viene impiegato sfruttando la response mode `REFINE`. Come visto in precedenza questa modalità operativa sfrutta due template prompt diversi (Listato 12), la `DEFAULT_TEXT_QA_PROMPT_TMPL` per il primo nodo recuperato e la `DEFAULT_REFINE_PROMPT_TMPL` per i successivi $n-1$ nodi.

```
DEFAULT_TEXT_QA_PROMPT_TMPL = (
    "Context information is below.\n"
    "---------------------\n"
    "{context_str}\n"
    "---------------------\n"
    "Given the context information and not prior knowledge, "
    "answer the query.\n"
    "Query: {query_str}\n"
    "Answer: "
)

DEFAULT_REFINE_PROMPT_TMPL = (
    "The original query is as follows: {query_str}\n"
    "We have provided an existing answer: {existing_answer}\n"
    "We have the opportunity to refine the existing answer "
    "(only if needed) with some more context below.\n"
    "------------\n"
    "{context_msg}\n"
    "------------\n"
    "Given the new context, refine the original answer to better "
    "answer the query. "
    "If the context isn't useful, return the original answer.\n"
    "Refined Answer: "
)
```

Listato 12: Default prompt per la refine mode di Llama Index

Nel listato 13 e nel listato 14 si possono osservare dei place holder delimitati da parentesi graffe:

- `{query_str}` : Rappresenta la user query digitata dall'utente che verrà poi aumentata dalle informazioni contestuali recuperate dall'indice.
- `{context_str} & {context_msg}` : Rappresentano le informazioni contestuali recuperate dall'indice.

- `{existing_answer}` : Rappresenta la risposta ottenuta allo step $i-1$ da inserire nello step i-esimo (solo per il refine prompt).

Llama Index, attraverso il QueryEngine, offre la possibilità di eseguire l'override dei template prompt utilizzati dal response synthetizer. Questa possibilità viene offerta per consentire all'utilizzatore di sintetizzare, tramite opportune tecniche di prompt engineering, dei template prompt più specifici per l'ambito applicativo in cui il framework viene utilizzato. Detto ciò, sono stati sintetizzati i seguenti template prompt per rimpiazzare quelli di default proprie del framework.

```
qea_template = (
    " <<SYS>> \n"
    "You are a helpful, respectful and honest assistant for labeling topics.\n"
    "<</SYS>> \n"
    "Context information is below.\n"
    "---------------------\n"
    "- Traditional diets in most cultures were primarily plant-based with a little
    meat on top, but with the rise of industrial style meat production and factory
    farming, meat has become a staple food.\n"
    "- Meat, but especially beef, is the word food in terms of emissions.\n"
    "- Eating meat doesn't make you a bad person, not eating meat doesn't make you a
    good one.\n"
    "---------------------\n"
    "Given the context information and not prior knowledge, "
    "answer the query.\n"
    "Query: The topic is described by the following keywords: [\"meat\", \"beef\", \"
    eat\", \"eating\", \"emissions\", \"steak\", \"food\", \"health\", \"processed\",
    \"chicken\"]. Based on this information, please create a short label of this topic
    . Make sure you to only return the label and nothing more.\n"
    "Answer: [/INST] Environmental impacts of eating meat \n"
    "[INST] \n"
    "Context information is below.\n"
    "---------------------\n"
    "{context_str}\n"
    "---------------------\n"
    "Given the context information and not prior knowledge, "
    "answer the query.\n"
    "Query: {query_str}\n"
    "Answer: "
)
```

Listato 13: Default prompt per la refine mode di Llama Index

```
r_template = (
    " <<SYS>> \n"
    "You are a helpful, respectful and honest assistant for labeling topics.\n"
    "<</SYS>> \n"
    "The original query is as follows: The topic is described by the following
    keywords: [\"meat\", \"beef\", \"eat\", \"eating\", \"emissions\", \"steak\", \"
    food\", \"health\", \"processed\", \"chicken\"]. Based on this information, please
     create a short label of this topic. Make sure you to only return the label and
    nothing more.\n"
    "We have provided an existing answer: Environmental impacts of eating meat \n"
    "We have the opportunity to refine the existing answer "
    "(only if needed) with some more context below.\n"
    "------------\n"
    "- Traditional diets in most cultures were primarily plant-based with a little
    meat on top, but with the rise of industrial style meat production and factory
    farming, meat has become a staple food.\n"
    "- Meat, but especially beef, is the word food in terms of emissions.\n"
    "- Eating meat doesn't make you a bad person, not eating meat doesn't make you a
    good one.\n"
    "------------\n"
    "Given the new context, refine the original answer to better "
    "answer the query. "
    "If the context isn't useful, return the original answer.\n"
    "Refined Answer: [/INST] Environmental impacts of eating meat \n"
    "[INST] \n"
    "The original query is as follows: {query_str}\n"
    "We have provided an existing answer: {existing_answer}\n"
    "We have the opportunity to refine the existing answer "
    "(only if needed) with some more context below.\n"
    "------------\n"
    "{context_msg}\n"
    "------------\n"
    "Given the new context, refine the original answer to better "
    "answer the query. "
    "If the context isn't useful, return the original answer.\n"
    "Refined Answer: "
)
```

Listato 14: Default prompt per la refine mode di Llama Index

L'approccio che è stato scelto per sintetizzare i prompt per Llama Index ricalca quello seguito per il legacy, ovvero costruirle mettendo insieme system prompt, example prompt e main

prompt. Per ottenere i prompt template e la user query, il processo di sintesi ha attraversato diverse fasi di messa appunto, ogniuna delle quali identificata da un codice (`TMPL` seguito da un numero per identificare i prompt template e `Q` seguita da un numero per identificare le user query). Per brevità nel presente elaborato sono riportate solamente le versioni utilizzate negli esperimenti (`TMPL4` per i prompt template relativi alla modalità refine, `TMPL3` per il prompt template relativo alla modalità tree summarize e `Q1` per la user query usata in tutte le modalità operative). Come system prompt, racchiuso tra i tag `<<SYS>>` `<</SYS>>` sia per la qa che per il refine prompt, è stato scelto lo stesso del procedimento legacy, ovvero **You are a helpful, respectful and honest assistant for labeling topics.** Questo system prompt, come nel procedimento legacy, serve per fare in modo che il modello di linguaggio si focalizzi sull'obbiettivo da raggiungere, ovvero la generazione di etichette. Successivamente si è deciso di inserire una componente di esempio che però varia tra QA e refine prompt in quanto la struttura deve ricalcare quella del main prompt. Si è scelto quindi di inserire dei dati di esempio, una user query che comprende una lista di parole chiavi relative ad un determinato topic, una serie di documenti relativi a quel topic e infine la risposta attesa ovvero l'etichetta. Come si può notare l'etichetta di esemptio è esclusa dai tag `[INST]` `[/INST]` in quanto non è un'istruzione per il modello ma ciò che vogliamo ottenere. Infine, in un nuovo tag `[INST]` `[/INST]` si è provveduto ad inserire il main prompt finale, con i placeholder nelle posizioni corrette dove LlamaIndex va a popolare con le informazioni appropriate. Si fa presente che non sono specificati i tag di apertura `[INST]` all'inizio dei prompt template e `[/INST]` alla fine in quanto, prima di passare tutto al modello di linguaggio, Llama Index inserisce il prompt template popolato di tutti gli elementi all'interno del query wrapper che riporta già i tag di apertura e chiusura delle istruzioni al modello.

Response synthesizer in tree summarize mode

Per quanto riguarda la modalità operativa tree summarize la procedura è fondamentalmente la stessa. La differenza è che questa modalità prevede che le informazioni contestuali vengano inserite tutte all'interno di un unico prompt eseguita al modello di linguaggio (Listato 15). In questo caso il template prompt è unico ed è stato riformulato rispetto a quello originale del framework.

```
1
2  ts_template = (
3      " <<SYS>> \n"
4      "You are a helpful, respectful and honest assistant for labeling topics.\n"
5      "<</SYS>> \n"
6      "Context information from multiple sources is below.\n"
7      "---------------------\n"
8      "- Traditional diets in most cultures were primarily plant-based with a little
       meat on top, but with the rise of industrial style meat production and factory
       farming, meat has become a staple food.\n"
9      "- Meat, but especially beef, is the word food in terms of emissions.\n"
10     "- Eating meat doesn't make you a bad person, not eating meat doesn't make you a
       good one.\n"
11     "---------------------\n"
12     "Given the information from multiple sources and not prior knowledge, "
13     "answer the query.\n"
14     "Query: The topic is described by the following keywords: [\"meat\", \"beef\", \"
       eat\", \"eating\", \"emissions\", \"steak\", \"food\", \"health\", \"processed\",
       \"chicken\"]. Based on this information, please create a short label of this topic
       . Make sure you to only return the label and nothing more.\n"
15     "Answer: [/INST] Environmental impacts of eating meat \n"
16     "[INST] \n"
17     "Context information from multiple sources is below.\n"
18     "---------------------\n"
19     "{context_str}\n"
20     "---------------------\n"
21     "Given the information from multiple sources and not prior knowledge, "
22     "answer the query.\n"
23     "Query: {query_str}\n"
24     "Answer: "
25 )
```

Listato 15: Default prompt per la tree summarize mode di Llama Index

Come si può vedere, anche in questo caso, la struttura ricalca quella degli altri template. Abbiamo una prima parte racchiusa tra i tag `<<SYS>>` `<</SYS>>` che indica la porzione relativa al system prompt, una parte successiva che riporta l'example prompt e infine abbiamo il main prompt completo di placeholder che verranno valorizzati dal framework esattamente come nella modalità refine. Per ovvi motivi non è presente il placeholder `{existing_answer}`.

5.4 Llama Index estrazione su topic specifici di abstract

Per sperimentare un approccio leggermente diverso alla definizione dei topic si è deciso di seguire un approccio diverso per la creazione degli indici e il recupero dei documenti. L'approccio scelto prevede la suddivisione dei documenti in topic e la costruzione di un indice per ogni topic in modo da poter migliorare la capacità di Llama Index di scegliere documenti esclusivamente del topic in oggetto.

5.4.1 Costruzione indice

Il primo passo riguarda l'estrazione dei documenti con la rispettiva organizzazione degli stessi in una cartella per ogni topic. Lo script che si occupa di questo è

`dataset/estrazione_dataset_da_ML-ArXivPapers_per_topic.ipynb`

Questo script prende in ingresso il file `TopicModellingLLama2/RISULTATI/docs_info.csv` (Fig. 23) prodotto da BertTopic nelle fasi precedenti e costituito da tante righe quanti sono gli abstract contenuti nel dataset con accanto, tra altre informazioni, anche il nome del topic a cui appartiene secondo BertTopic. Specificata la directory di output lo script genera

Document	Topic	Name	Representation	KeyBERT	Llama2
Tree-structured data usually contain both topological and g…	-1	-1_the_of_and_to	['the', 'of', 'and', 'to', 'in', 'we', 'is', 'for', 'that', 'on']	['models', 'model', 'learning', 'algorithms', 'neural', 'datasets'…	['Machine
Given side information that an Ising tree-structured graphic…	-1	-1_the_of_and_to	['the', 'of', 'and', 'to', 'in', 'we', 'is', 'for', 'that', 'on']	['models', 'model', 'learning', 'algorithms', 'neural', 'datasets'…	['Machine
Annotating seismic data is expensive, laborious and subjec…	31	31_climate_and_the_of	['climate', 'and', 'the', 'of', 'weather', 'data', 'seismic', 'to', 'in', …	['forecasting', 'modeling', 'models', 'forecasts', 'prediction', '…	['Climate a
Estimating scene flow in RGB-D videos is attracting much i…	3	3_3d_object_video_and	['3d', 'object', 'video', 'and', 'to', 'the', 'we', 'image', 'of', 'segm…	['3d', 'models', 'representations', 'model', 'segmentation', 'sh…	['3D Objec
Traffic congestion has large economic and social costs. Th…	23	23_driving_autonomous_vehicle_vehicles	['driving', 'autonomous', 'vehicle', 'vehicles', 'driver', 'lane', 'tr…	['driving', 'autonomous', 'simulation', 'trajectories', 'traffic', 'm…	['Autonom
Machine learning (ML) is quickly emerging as a powerful to…	-1	-1_the_of_and_to	['the', 'of', 'and', 'to', 'in', 'we', 'is', 'for', 'that', 'on']	['models', 'model', 'learning', 'algorithms', 'neural', 'datasets'…	['Machine

Figura 23: *Estratto di docs_info.csv*

tante cartelle quanti sono i topic e inserisce all'interno di ogni cartella tutti gli abstract che hanno sulla stessa riga il topic relativo alla cartella in oggetto. Al termine si ottiene quindi una struttura a directory che raccoglie tutti gli abstract per topic. Il passo successivo consiste nell'estrazione di un file contenente tutte le keyword di ogni topic. Questo è ricavabile dal file `TopicModellingLLama2/RISULTATI/topic_labels_EX0.csv` (Fig. 24). Nello stesso notebook è presente una seconda cella che si occupa di eseguire questo compito.

Topic	Count	Name	Representation	KeyBERT
-1	18512	-1_the_of_and_to	['the', 'of', 'and', 'to', 'in', 'we', 'is', 'for', 'that', 'on']	['models', 'model', 'learning', 'algorithms', 'neural', 'datasets', 'networks', 'training', 'algorithm', 'dataset']
0	4994	0_policy_reinforcement_learning_rl	['policy', 'reinforcement', 'learning', 'rl', 'agent', 'to', 'the', 'in', 'we', 'of']	['dynamics', 'learning', 'reinforcement', 'robot', 'model', 'training', 'algorithms', 'agents', 'learned', 'markov']
1	1804	1_speech_audio_speaker_music	['speech', 'audio', 'speaker', 'music', 'the', 'to', 'and', 'of', 'asr', 'acoustic']	['encoder', 'voice', 'neural', 'speech', 'trained', 'language', 'audio', 'learning', 'models', 'acoustic']
2	1756	2_privacy_federated_fl_private	['privacy', 'federated', 'fl', 'private', 'data', 'clients', 'learning', 'the', 'model', 'to']	['federated', 'heterogeneity', 'distributed', 'datasets', 'algorithms', 'models', 'gradient', 'private', 'model', 'cc
3	1361	3_3d_object_video_and	['3d', 'object', 'video', 'and', 'to', 'the', 'we', 'image', 'of', 'segmentation']	['3d', 'models', 'representations', 'model', 'segmentation', 'shape', 'shapes', 'depth', 'representation', '2d']
4	1291	4_segmentation_images_image_medical	['segmentation', 'images', 'image', 'medical', 'and', 'the', 'of', 'to', 'imaging', 'cancer']	['segmentation', 'convolutional', 'imaging', 'mri', 'supervised', 'models', 'networks', 'model', 'scans', 'neura
5	1264	5_language_word_models_text	['language', 'word', 'models', 'text', 'bert', 'translation', 'model', 'we', 'to', 'on']	['nlp', 'embeddings', 'embedding', 'multilingual', 'language', 'corpus', 'syntactic', 'semantic', 'languages', '
6	1155	6_user_recommendation_items_users	['user', 'recommendation', 'items', 'users', 'item', 'recommender', 'the', 'to', 'ranking', 'and']	['recommender', 'embeddings', 'embedding', 'recommendation', 'prediction', 'collaborative', 'models', 'mo
7	1153	7_adversarial_attacks_robustness_attack	['adversarial', 'attacks', 'robustness', 'attack', 'examples', 'perturbations', 'to', 'against', 'robust', 'the']	['adversarial', 'adversarially', 'dnns', 'dnn', 'attacks', 'imagenet', 'models', 'robustness', 'trained', 'model']
8	1133	8_networks_neural_the_network	['networks', 'neural', 'the', 'network', 'deep', 'of', 'relu', 'layer', 'that', 'we']	['dnns', 'regularization', 'neural', 'gradient', 'optimization', 'generalization', 'layers', 'networks', 'trained', 'ap
9	1077	9_graph_node_graphs_gnns	['graph', 'node', 'graphs', 'gnns', 'nodes', 'gnn', 'networks', 'embedding', 'network', 'the']	['graphs', 'graph', 'subgraph', 'networks', 'embeddings', 'embedding', 'nodes', 'gnns', 'convolutional', 'gnn']
10	989	10_equations_the_neural_differential	['equations', 'the', 'neural', 'differential', 'of', 'physics', 'dynamics', 'systems', 'pdes', 'dynamical']	['pdes', 'dynamics', 'modeling', 'pde', 'models', 'simulations', 'equations', 'model', 'computational', 'dynam
11	974	11_molecular_protein_drug_molecules	['molecular', 'protein', 'drug', 'molecules', 'chemical', 'of', 'and', 'the', 'to', 'for']	['molecular', 'molecules', 'compounds', 'ligand', 'computational', 'molecule', 'prediction', 'predicting', 'drug
12	944	12_gradient_stochastic_convex_convergence	['gradient', 'stochastic', 'convex', 'convergence', 'sgd', 'optimization', 'descent', 'problems', 'rate', 'the']	['minimization', 'optimization', 'gradients', 'stochastic', 'gradient', 'hessian', 'convergence', 'optimal', 'algo
13	936	13_label_classification_labels_the	['label', 'classification', 'labels', 'the', 'tree', 'feature', 'trees', 'of', 'selection', 'to']	['classifiers', 'prediction', 'regression', 'supervised', 'classifier', 'datasets', 'classification', 'algorithms', 'm
14	859	14_channel_wireless_the_to	['channel', 'wireless', 'the', 'to', 'network', 'and', 'in', 'of', 'based', 'is']	['networks', 'mimo', 'modulation', 'sensing', 'wireless', 'network', 'signals', 'signal', 'optimization', '5g']

Figura 24: *Estratto di topic_labels_EX0.csv*

All'interno di `topic_labels_EX0.csv` sono presenti tutti i topic estratti da BertTopic comprese le keywords estratte. Il codice non fa altro che estrarle dal csv e inserirle all'interno di un file json chiamato `/dataset/ML-ArXiv-keywords.json` (Fig. 25) insieme al topic id e al nome del topic. Ultimata la fase di recupero delle informazioni si passa al-


```
[
    {
        "topic_id":-1,
        "name":"-1_the_of_and_to",
        "keywords":[
            "models",
            "model",
            "learning",
            "algorithms",
            "neural",
            "datasets",
            "networks",
            "training",
            "algorithm",
            "dataset"
        ]
    },
    {
        "topic_id":0,
        "name":"0_policy_reinforcement_learning_rl",
        "keywords":[
            "dynamics",
            "learning",
            "reinforcement",
            "robot",
            "model",
            "training",
            "algorithms",
            "agents",
            "learned",
            "markov"
        ]
    },
```


Figura 25: *Estratto di ML-ArXiv-keywords.json*

la fase successiva dove si procede a generare l'indice. Il notebook che se ne occupa è

`LLAMA_INDEX/indici/costruzione_indice_topic.ipynb`. Diversamente da quanto fatto in precedenza, in questa fase è necessario generare k indici con $k < n$ (n numero totale dei topic) pari al numero di topic che si vogliono analizzare. Dopo aver installato le librerie e montato il drive, la prima cella è quella che si occupa di generare i path di tutte le cartelle relative ai topic su cui si vuole creare l'indice. Nel listato 16 sono riportati i parametri che vanno valorizzati.

```
1
2  #Lista con i topic code su cui generare l'indice
3  topic_code = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
4
5  #Path contenente i dati su cui creare l'indice organizzati in topic (ogni topic in una
       directory con nome: topicCode_topicName)
6  dataset_dir="/content/drive/MyDrive/Tirocinio_Magistrale/dataset/ML-ArXiv-bytopic"
7  #Path dove andare a salvare gli indici costruiti sugli abstract interi. NOTA gli
       indici saranno organizzati secondo la stessa struttura a directory dell'input
8  index_dir="/content/drive/MyDrive/Tirocinio_Magistrale/LLAMA_INDEX/indici/
       VECTOR_STORE_INDEX/ML-ArXiv_58796_topic"
9  #Path dove andare a salvare gli indici costruiti sulle frasi estratte dagli abstract.
       NOTA gli indici saranno organizzati secondo la stessa struttura a directory dell'
       input
10 index_sentences_dir="/content/drive/MyDrive/Tirocinio_Magistrale/LLAMA_INDEX/indici/
       VECTOR_STORE_INDEX/ML-ArXiv_58796_topic_frasi"
```

Listato 16: Parametri da valorizzare per la generazione dei path relativi ai topic su cui generare gli indici

- `topic_code` contiene la lista dei topic id su cui si vuole costruire l'indice.
- `dataset_dir` è la cartella radice all'interno della quale è contenuto il dataset suddiviso in topic.
- `index_dir` è la cartella all'interno della quale vanno a finire gli indici che vengono generati.
- `index_sentences_dir` è la cartella all'interno della quale vanno a finire gli indici generati sui dataset suddivisi in frasi. Questa modalità è riportata in dettaglo nelle sezioni successive.

Il codice di questo blocco utilizza le informazioni contenute nei parametri per generare la lista dei path delle cartelle dei topic selezionati da cui andare a leggere i documenti, generare la lista dei path dove andare a salvare gli indici generati per ogni topic sia per quelli costruiti su i dataset interi che per quelli generati sui dataset suddivisi in frasi. Generati i path, il codice presente nel listato 17 si occupa di caricare il modello Llama2, i blocchi sono gli stessi presenti in tutti gli altri script. Infine il blocco finale esegue le stesse operazioni per la creazione dell'indice come nel caso dell'indice unico, l'unica differenza è che l'inserimento di documenti non è unico ma è ripetuto tante volte quanti sono i topic selezionati. Come anticipato in precedenza è presente un altro blocco che si occupa di costruire anche gli indici relativi alla divisione in frasi, che viene approfondito nelle sezioni successive.

```
for topic_path, index_path in zip(dataset_paths, index_sentences_paths):
  doc_chunks = []
  i = 0
  for filename in os.listdir(topic_path):
      if filename.endswith('.txt'):
          clear_output(wait=True)
          print("\nGenerazione indice " + topic_path.split("/")[-1] + "\n")
          print("Loading files: ", np.round(i/len(os.listdir(topic_path)) * 100,2),"%"
    )
          file_path = os.path.join(topic_path, filename)
          # Open and read the file
          with open(file_path, 'r', encoding='utf-8') as file:
            doc = Document(text=file.read(), id_=f"doc_id_{i}")
            doc_chunks.append(doc)
            i = i + 1
```

Listato 17: Generazione indice per ogni topic.

5.4.2 Esecuzione degli esperimenti

Terminata la fase di costruzione dell'indice il passo successivo prevede l'esecuzione degli esperimenti. Il notebook che implementa ciò è

`LLAMA_INDEX/2_LLAMA_INDEX_Esperimenti_topic.ipynb`

Questo notebook è una versione modificata rispetto a quello visto nelle sezioni precedenti proprio per tenere conto del fatto che stiamo lavorando con k indici diversi invece che uno

solo. Dopo le celle che implementano il caricamento delle librerie e il montaggio del drive troviamo una sezione che definisce alcuni path (Listato 18).

```
keywords_path = "/content/drive/MyDrive/Tirocinio_Magistrale/dataset/ML-ArXiv-keywords.json"
index_text_directory = '/content/drive/MyDrive/Tirocinio_Magistrale/LLAMA_INDEX/indici/VECTOR_STORE_INDEX/ML-ArXiv_58796_topic'
index_sentences_directory = '/content/drive/MyDrive/Tirocinio_Magistrale/LLAMA_INDEX/indici/VECTOR_STORE_INDEX/ML-ArXiv_58796_topic_frasi'
output_path = "/content/drive/MyDrive/Tirocinio_Magistrale/LLAMA_INDEX/output/TMPL3&4_Q1_topic/"
report_file = "/content/drive/MyDrive/Tirocinio_Magistrale/LLAMA_INDEX/report/TMPL3&4_Q1_topic/"
topic_codes = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Listato 18: Definizione dei path.

- `keywords_path` definisce il path del file contentente le keywords da utilizzare all'interno delle query
- `index_text_directory` definisce il path della directory che contiene tutti gli indici di tutti i topic costituiti a partire da testi interi.
- `index_text_directory` definisce il path della directory che contiene tutti gli indici di tutti i topic costituiti a partire da testi suddivisi per frasi.
- `output_path` definisce il path della directory radice dove andranno salvati i risultati degli esperimenti suddivisi in sotto cartelle per nome del topic
- `report_path` definisce il path di un file riassuntivo che contiene per ogni topic le etichette generate da ogni esperimento.
- `topic_codes` è una lista dei topic-id selezionati per il run che si sta eseguendo.

Successivamente è presente una con i i path per il salvataggi dei risultati degli esperimenti basati sia sui testi interi che sui testi suddivisi in frasi. Caricato Llama2, in seguito è presente una parte dove avviene il caricamento delle keywords dal file json che le contiene (Listato 19). Questa cella utilizza `topic_codes` per selezionare quali keyword di quali topic andare a caricare.

```python
import json

# Function to load specific topic_ids from a JSON file
def load_specific_topic_ids(path, ids):
    """
    Load specific topic_ids from a JSON file into a dictionary.

    Parameters:
    - path: Path to the JSON file.
    - ids: A list of integers representing the topic_ids to be loaded.

    Returns:
    A dictionary with the loaded topic_ids and their corresponding keywords.
    """
    with open(path, 'r') as file:
        data = json.load(file)

    filtered_data = {}
    for item in data:
        if int(item["topic_id"]) in ids:
            filtered_data[item["topic_id"]] = item["keywords"]

    return filtered_data

# Load the specified topic_ids
keywords = load_specific_topic_ids(keywords_path, topic_codes)

for key, value in keywords.items():
    print(f"{key}: {value}")
```

Listato 19: Caricamento keyword.

Sempre nella stessa sezione, come in precedenza, è presente una cella che definisce i prompt custom da utilizzare. Una volta completata la preparazione, si procede con la fase di esecuzione degli esperimenti, che si svolge in modo sostanzialmente analogo a quanto illustrato nelle sezioni precedenti. La differenza principale risiede nel fatto che l'esecuzione avviene per ciascuno dei topic da analizzare: per ogni ciclo, si carica un nuovo indice, si conducono gli esperimenti e i risultati vengono registrati nella cartella corrispondente. Un aspetto distintivo di questa fase è che, all'interno della stessa iterazione, si effettuano sia gli esperimenti basati sugli indici derivati dall'intero dataset (EX1, EX2) sia quelli relativi agli indici specifici

costruiti su dataset frazionati in frasi. Di conseguenza, in ogni iterazione avviene prima il caricamento degli indici per EX1 ed EX2 e successivamente quello per EX3 ed EX4.

5.5 Llama Index estrazione su topic specifici di frasi degli abstract

Come accennato più volte nelle sezioni precedenti, gli esperimenti contrassegnati con i codici `EX3` e `EX4` prevedono l'esecuzione sul dataset costituito dai testi suddivisi in frasi. L'ipotesi che c'è dietro a questa modalità è che costruendo gli indici sulle frasi Llama Index sia in grado di recuperare informazioni più specifiche rispetto a quello che sia in grado di fare con testi interi e quindi aumentare la qualità delle etichette prodotte.

5.5.1 Costruzione indice

Come detto in precedenza, la porzione di codice che si occupa della generazione degli indici basati su frasi si trova all'interno dello script

`LLAMA_INDEX/indici/costruzione_indice_topic.ipynb`

identificato con il nome **Sentences extraction**. Il codice, per ogni dataset path contenuto all-interno della lista `dataset_paths`, provvede a caricare i testi contenuti in ogni abstract file e, invece che inserirli direttamente nell'indice come in passato, utilizza il componente `SentenceWindowNodeParser` di Llama Index (Listato 20).

```
# parse nodes
splitter = SentenceWindowNodeParser.from_defaults(
    window_size=3,
    window_metadata_key="window",
    original_text_metadata_key="original_text",
)
nodes = splitter.get_nodes_from_documents(doc_chunks, show_progress = True)

# build index
index = VectorStoreIndex(nodes, show_progress=True)
```

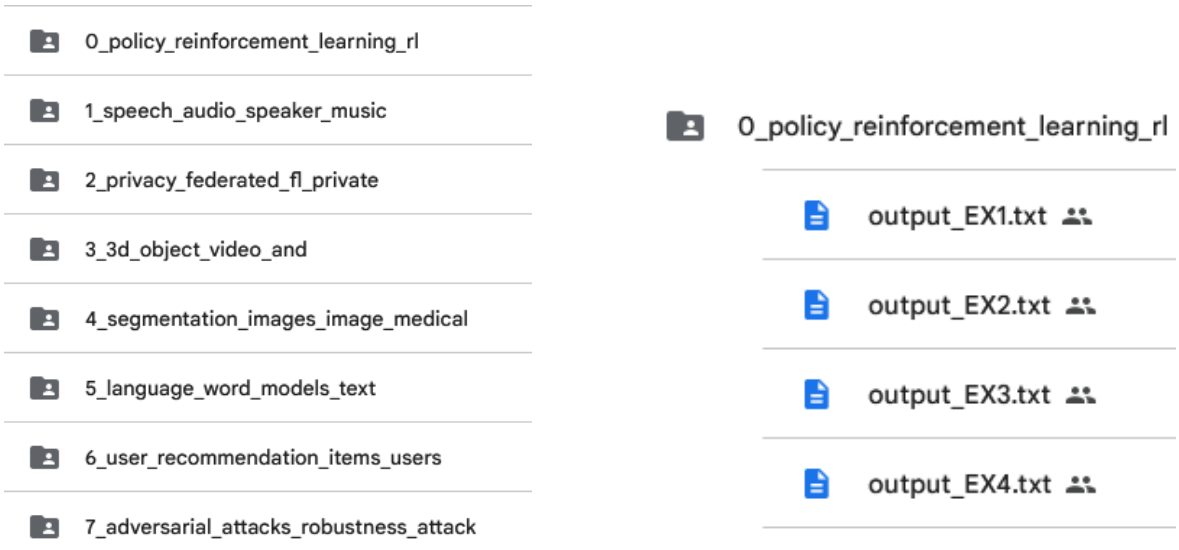
Listato 20: Estrazione delle frasi tramite il componente SentenceWindowNodeParser di LLamaIndex.

I parser di nodi, quali astrazioni semplici, elaborano un elenco di documenti suddividendoli in oggetti Nodo, in modo che ciascun nodo rappresenti un frammento specifico del documento genitore. Quando un documento viene frammentato in nodi, tutti i suoi attributi vengono ereditati dai nodi figli, inclusi metadati, testo e modelli di metadati. Nel contesto di Llama Index, il SentenceWindowNodeParser sfrutta questo processo per dividere i documenti in singole frasi, generando nodi che, oltre al testo specifico, incorporano nei loro metadati una "finestra" di frasi adiacenti. Quest'approccio non solo mantiene la ricchezza informativa del documento originale ma amplifica l'utilità del nodo preparandolo per un'analisi contestuale più profonda. Questo rende il SentenceWindowNodeParser estremamente efficace per la generazione di embedding focalizzati, potenziati ulteriormente dalla possibilità di sostituire le frasi con il loro contesto attraverso l'integrazione con il MetadataReplacementNodePostProcessor (un componente che consente di manipolare o sostituire metadati associati ai nodi di un indice), prima della loro elaborazione finale dall'LLM. Terminata la procedura di splitting dei nodi da parte del SentenceWindowNodeParser, i nodi vengono passati al VectorStoreIndex per essere inseriti all'interno dell'indice con la successiva memorizzazione all'interno del path corrispondente al rispettivo topic. Questa procedura viene ripetuta k volte per ogni topic selezionato. Ricavato l'indice i passi successivi sono gli stessi visti per le fasi precedenti e le operazioni vengono svolti negli stessi script di seguito alle altre.

5.6 Risultati

Per quanto riguarda i risultati degli esperimenti, all'interno della directory indicata dal `output_path` del listato 18 viene riprodotta la solita struttura a sottodirectory per ogni topic (Fig. 26a) con all'interno i 4 file che riassumono tutte le chiamate al modello di linguaggio (Fig.26b). All'interno di ogni file risultato sono presenti:

- **UserQuery** ovvero la query che viene inoltrata all'LLM
- **Prompt** ovvero il prompt template definito e valorizzato da LlamaIndex con le opportune informazioni
- **Completion** solo per la modalità refine, è la risposta parziale fornita dall'LLM che poi viene inclusa nella chiamata successiva (visibile chiaramente in questi file di output)



(a) *Lista topic* **(b)** *Risultati per ogni topic*

Figura 26: *Risultati*

- **Response** è la risposta che LLM fornisce al termine delle chiamate per la modalità refine oppure della chiamata per la modalità tree_summarize.

Per ottenere il logging di tutte le chiamate che Llama Index fa all'LLM è necessario abilitare il modulo di logging con la chiamata riportata nel listato 21

```
#import logging
import llama_index
llama_index.set_global_handler("simple")
```

Listato 21: Abilitare il modulo di logging di LLamaIndex.

e successivamente in fase di chiamata va eseguito un redirect dell'stdout verso il file di testo desiderato tramite l'istruzione `with redirect_stdout(f):` Questo consente di ridirigere l'output dall'stdout (unica modalità del modulo di logging semplice di Llama Index) verso un file di testo persistente. Questo perché Llama Index include la possibilità di sfruttare dei framework di logging molto sofisticati ma la loro eccessiva complessità di utilizzo è stata ritenuta superflua ai fini del progetto sviluppato e quindi si è deciso di ripiegare sulla soluzione utilizzata. Contestualmente ai file di logging per ogni topic, viene prodotto anche un file chiamato `report.csv` (Fig. 27)che riassume, in una struttura tabellare, tutte le etichette generate per ogni topic per ogni esperimento (tranne EX0 per cui vanno aggiunte manualmente).

topic_id	KeyBERT	EX0	EX1	EX2	EX3	EX4
0	['dynamics', 'learning', 'reinforcen	Reinforcement Learning for Multi-	Reinforcement learning in R	Reinforcement learning and dynar	Reinforcement learning for robot c	Reinforcement Learning for Robotics
1	['encoder', 'voice', 'neural', 'spee	Audio Processing and Speech Rec	Audio transcription using deep lea	Audio representation and transcri	Speech encoding for singing voice	Neural Speech Learning
2	['federated', 'heterogeneity', 'distr	Privacy-Preserving Machine Learn	Federated Learning Heterogeneity	Privacy-preserving distributed ma	Federated Heterogeneous Learnir	Distributed Machine Learning with Heterogeneous Data
3	['3d', 'models', 'representations', '	3D Object Reconstruction and Ge	Shape representation and segmer	Shape representation and segmer	Fine-grained 3D shape segmentat	Computer Vision and 3D Shape Representation
4	['segmentation', 'convolutional', 'i	Medical Image Segmentation	Medical Image Segmentation	Medical Image Segmentation	Medical image segmentation with	Medical image segmentation with deep learning models
5	['nlp', 'embeddings', 'embedding',	Natural Language Processing	Multilingual NLP Embeddings	NLP Embeddings and Multilingual	Multilingual NLP modeling	Multilingual Embeddings for Natural Language Processing
6	['recommender', 'embeddings', 'e	Recommender Systems	Personalized Neural Embedding (F	Embedding-based Recommendat	Embedding-based recommendatio	Embedding-based Recommendation Models
7	['adversarial', 'adversarially', 'dnn	Adversarial Robustness in Machin	Robust adversarial training for pat	Adversarial Robustness Training	Adversarial attacks on deep neura	Adversarial attacks on deep neural networks (DNNs)
8	['dnns', 'regularization', 'neural', '	Training and Optimization of Deep	Deep Neural Networks Generaliza	Neural network regularization	Deep neural network regularizatio	Deep Neural Networks Regularization
9	['graphs', 'graph', 'subgraph', 'net	Graph Learning	Graph embedding and multi-layer	Graph embedding and analysis	Graph embedding	Graph Embeddings and Representation Learning
10	['pdes', 'dynamics', 'modeling', 'pi	Physics-Informed neural networks	NeuralPDE Modeling	Neural modeling of dynamical sys	Stochastic Partial Differential Equ	Mathematical modeling of dynamical systems
11	['molecular', 'molecules', 'compo	Machine Learning in Drug Discove	Computational drug discovery usi	Computational drug design	Molecular dynamics (MD) in drug d	Molecular Prediction and Drug Discovery
12	['minimization', 'optimization', 'gra	Optimization of Stochastic Object	Gradient Descent and Stochastic	Gradient Descent and Stochastic	Stochastic optimization of machir	Stochastic Optimization
13	['classifiers', 'prediction', 'regressi	Machine Learning Techniques for	Supervised learning classifier perf	Supervised learning	Multi-target image classification	Supervised learning and classification
14	['networks', 'mimo', 'modulation',	Wireless Network Design and Man	Edge-based Heterogeneous Wirel	Wireless Signal Recognition and C	Interference source identification	Wireless Network Optimization
15	['predictive', 'ehr', 'ehrs', 'predictii	Healthcare Informatics	Predictive EHR Representation Le	Predictive Healthcare Analytics	Predictive Healthcare Analytics	Predictive Analytics in Healthcare

Figura 27: *Etichette generate per ogni esperimento*

Capitolo 6

Conclusioni e sviluppi futuri

6.1 Riepilogo dei risultati

Gli esperimenti condotti hanno permesso la generazione di un insieme distinto di etichette per ciascuno dei venti topic selezionati. La tabella 1 offre una sintesi concisa delle peculiarità associate a ciascun esperimento, rappresentando una risorsa preziosa per il processo di valutazione. Nelle tabelle 2, 3, 4 sono presentate diverse tabelle, ciascuna dedicata a un differente topic. Per ogni topic, vengono elencate le etichette prodotte da ciascun esperimento, facilitando un'analisi dettagliata e comparativa delle performance.

Topic 4	
Experiment	**Description**
EX0	Medical Image Segmentation
EX1	Medical Image Segmentation
EX2	Medical Image Segmentation
EX3	Medical image segmentation with CNNs
EX4	Medical image segmentation with deep learning techniques

Tabella 2: *Risultati degli esperimenti relativi al Topic 4: Medical Image Segmentation*

Topic 9	
Experiment	**Description**
EX0	Graph Learning
EX1	Graph embedding and multi-layer analysis
EX2	Graph embedding and analysis
EX3	Graph embedding
EX4	Graph Embeddings and Representation Learning

Tabella 3: *Risultati degli esperimenti relativi al Topic 9: Graph Learning*

Topic 14	
Experiment	**Description**
EX0	Wireless Network Design and Management
EX1	Edge-based Heterogeneous Wireless Signal Recognition
EX2	Wireless Signal Recognition and Optimization for 5G
EX3	Interference source identification in 5G MIMO networks
EX4	Wireless Network Optimization

Tabella 4: *Risultati degli esperimenti relativi al Topic 14: Wireless Network Design and Management*

Come si evince da questo gruppo rappresentativo di risultati, emerge una varietà nelle performance degli esperimenti: in alcuni casi, le etichette generate mostrano un'allineamento sostanziale, mentre in altri le discrepanze sono marcate, con risultati che divergono completamente.

6.2 Processo di valutazione

Per quanto riguarda il processo di valutazione dei risultati, è stato necessario adottare un approccio qualitativo piuttosto che quantitativo, data la mancanza di metriche standardizzate applicabili ad un dataset non etichettato come quello utilizzato. Questa scelta riflette una pratica comune anche nella letteratura accademica[9], dove numerosi studi che valutano i risultati prodotti dagli LLM si affidano a metodologie qualitative per apprezzare la coerenza,

la pertinenza e la completezza delle risposte generate, in assenza di parametri quantitativi consolidati.

6.2.1 Valutazione dei risultati

Il processo di valutazione dei risultati degli esperimenti di etichettatura ha coinvolto tre revisori esperti nel settore del machine learning. Ciascuno di loro ha espresso un giudizio indipendente su quale fosse, secondo la loro esperienza, l'etichetta più accurata prodotta dai diversi esperimenti. Questa metodologia qualitativa ha permesso di ottenere una valutazione approfondita e basata sull'expertise nel campo.

Esperimenti	**Revisore 1**	**Revisore 2**	**Revisore 3**
0	30%	35%	30%
1	10%	0%	0%
2	20%	5%	15%
3	5%	10%	10%
4	35%	50%	45%

Tabella 5: *Percentuali di label corrette per esperimenti*

Dai risultati riportati nella tabella 5, possiamo commentare le percentuali di etichette corrette attribuite dai revisori agli esperimenti:

- **Esperimento 0**: Ha ottenuto una percentuale relativamente equilibrata di etichette corrette (30%, 35%, 30%), suggerendo una performance media.
- **Esperimento 1**: Mostra le percentuali più basse (10%, 0%, 0%), indicando che questo esperimento non ha prodotto etichette adeguatamente rappresentative o accurate.
- **Esperimento 2**: Presenta percentuali leggermente migliori rispetto all'esperimento 1 (20%, 5%, 15%), ma ancora insufficienti per considerarlo efficace.
- **Esperimento 3**: Anche in questo caso, le percentuali sono basse (5%, 10%, 10%), suggerendo una qualità moderata delle etichette generate.

- **Esperimento 4**: Ha ricevuto le valutazioni più alte (35%, 50%, 45%), indicando che le etichette prodotte sono state ritenute le più accurate e utili dai revisori.

Questi risultati suggeriscono che l'Esperimento 4 è stato il più efficace nel produrre etichette di qualità, mentre l'Esperimento 1 ha mostrato difficoltà significative. Tale confronto mette in evidenza l'importanza di raffinare ulteriormente le tecniche di prompt engineering e di utilizzo di RAG per migliorare la qualità delle etichette generate. Per valutare la concordanza tra i revisori nella selezione delle etichette, è stata calcolata la kappa di Fleiss, un indice statistico che misura l'accordo inter-rater per valutazioni categoriali. Questo indice è stato calcolato in due modi distinti:

- Nel primo, è stato considerato quale delle etichette dei cinque esperimenti per ciascuno dei 20 topic fosse ritenuta la migliore da ciascun revisore. In questo scenario, la kappa di Fleiss ha prodotto un valore di 0,05, indicando un livello di accordo molto basso.

- Nel secondo metodo, l'analisi si è concentrata sull'accordo tra i revisori riguardo alla correttezza di ciascuna delle 100 etichette prodotte dai vari esperimenti. Questo approccio ha prodotto una kappa di Fleiss di 0,19, mostrando un accordo leggermente superiore ma ancora limitato.

Questi risultati evidenziano differenze significative nel livello di consenso tra i revisori, a seconda del metodo di valutazione impiegato. L'agreement tra i revisori risulta essere basso, il che può essere attribuito alla natura soggettiva delle valutazioni effettuate. Ogni revisore ha adottato criteri personali ritenuti più opportuni per giudicare le etichette. Questi criteri includono vari aspetti come la presenza o l'assenza di parole chiave rilevanti per ciascun topic, la correttezza dell'etichetta in relazione ai soli documenti effettivamente recuperati e passati all'LLM, tra gli altri. Questa diversità nei metodi di valutazione ha inevitabilmente portato a un livello di concordanza relativamente basso, riflettendo la complessità e la sfida di stabilire un accordo unanime in contesti di valutazione fortemente soggettivi. Dai risultati si può evincere inoltre, considerando anche EX0, che i risultati migliori si ottengono nel momento in cui si passa all'LLM tutta l'informazione nello stesso istante, cosa che avviene sia per EX0 senza l'utilizzo di tecniche RAG che in EX2 ed EX4 con la modalità `tree sumarize` del response sytnethizer. Di contro, gli scarsi risultati ottenuti dalla modalità `refine` sug-

geriscono che tale modalità operativa risulta più adatta ad applicazioni come i processi di sintesi, che prevedono un vero e proprio processo di distillazione dell'informazione contenuta in una determinata base documentale, data proprio dalle diverse chiamate in serie al modello di linguaggio. L'altra considerazione da fare e che effettivamente la suddivisione del dataset in frasi consente al sistema RAG di recuperare solo le porzioni di testo rilevanti, il che si traduce quindi in sintesi di etichette migliori (EX2 EX4) rispetto a quelle prodotte a partire da testi interi mantenendo comunque costante la quantità di informazione che si da in pasto al modello di linguaggio.

6.3 Limitazioni delle Soluzioni proposte

Tra i limiti delle soluzioni di topic modelling tramite LLM emergono principalmente due aspetti critici. Da un lato, vi è la necessità di disporre di una potenza di calcolo adeguata alla vastità e complessità delle informazioni che si intendono processare. Questo requisito implica significativi investimenti in infrastrutture hardware o l'acquisto di servizi cloud capaci di gestire grandi volumi di dati, con conseguenti costi elevati che possono limitare l'accessibilità a tali tecnologie, specialmente per entità con risorse limitate. Dall'altro lato, si presentano questioni legate ai diritti d'autore e all'uso dei dati, in particolare quando si lavora con materiale che non è di proprietà esclusiva o liberamente utilizzabile. Gli articoli di ricerca, ad esempio, sono spesso protetti da diritti d'autore, il che pone limitazioni significative sull'uso dei loro contenuti per l'addestramento o l'applicazione di modelli LLM. Questo aspetto solleva problematiche non solo legali ma anche etiche, obbligando i ricercatori a navigare un terreno complesso tra l'accesso alle informazioni necessarie per il topic modelling e il rispetto delle normative sui diritti d'autore. Questi limiti sottolineano la necessità di considerare attentamente sia gli aspetti economici sia quelli legali nella pianificazione e implementazione di soluzioni avanzate di topic modelling con LLM, garantendo così che l'innovazione tecnologica proceda in maniera sostenibile e responsabile.

6.4 Implicazioni Pratiche e sviluppi futuri

I risultati ottenuti confermano che l'impiego di sistemi RAG produce un incremento apprezzabile delle prestazioni nei sistemi di topic modelling basati su LLM. Oltre all'ambito della catalogazione di articoli di ricerca per il settore accademico, un'ulteriore e immediata implicazione di queste tecniche potrebbe emergere nel campo della sicurezza informatica, dove potrebbero trovare potenziale applicazione. In particolare, l'applicazione dei sistemi RAG nel settore della sicurezza informatica potrebbe rivoluzionare il modo in cui vengono identificate e classificate le minacce, migliorando significativamente la capacità di rilevamento precoce e la risposta agli attacchi. L'analisi dettagliata dei dati e la generazione di insight pertinenti attraverso il topic modelling permetterebbero agli esperti di sicurezza di decifrare e anticipare schemi di attacco, contribuendo così a rafforzare le misure di difesa e a mitigare i rischi associati a violazioni e intrusioni. Questo approccio rappresenta un potenziale passo avanti verso un'intelligenza di sicurezza più agile e adattiva, capace di affrontare le sfide in continua evoluzione del panorama delle minacce informatiche.

6.5 Riflessioni Finali

La capacità di questi sistemi di affinare e precisare le informazioni estratte dai vasti depositi di dati rappresenta un avanzamento tecnologico significativo, che potrebbe trasformare il modo in cui le informazioni sono analizzate e utilizzate. La facilità con cui possono essere adattati a contesti diversi dimostra la loro flessibilità e il potenziale per ulteriori sviluppi e miglioramenti. Tuttavia, è essenziale riconoscere e affrontare le sfide, come le questioni di costo, potenza computazionale e questioni etiche legate ai diritti d'autore e alla privacy dei dati. Queste considerazioni sottolineano la necessità di un'attenta pianificazione e di strategie di mitigazione dei rischi, garantendo che l'innovazione tecnologica proceda in maniera responsabile e sostenibile. In conclusione, mentre esploriamo le possibilità offerte da queste tecnologie avanzate, dobbiamo rimanere consapevoli delle loro implicazioni e lavorare proattivamente per massimizzare i loro benefici e minimizzare i potenziali rischi. Con l'approccio giusto, il futuro del topic modelling e dell'intelligenza artificiale appare ricco di opportunità per miglioramenti significativi in molteplici settori della società.

Bibliografia

[1] arxiv dataset. https://www.kaggle.com/datasets/Cornell-University/arxiv.

[2] Bertopic. https://maartengr.github.io/BERTopic/index.html.

[3] Coursera, generative ai with large language models. https://www.coursera.org/learn/generative-ai-with-llms/.

[4] Cshorten/ml-arxiv-papers. https://huggingface.co/datasets/CShorten/ML-ArXiv-Papers.

[5] Llama2. https://llama.meta.com/llama2/.

[6] Llamaindex. https://www.llamaindex.ai/.

[7] Official documentations of the python language. https://docs.python.org/3/.

[8] Topic modeling with llama2. https://colab.research.google.com/drive/1QCERSMUjqGetGGujdrvv_6_EeoIcd_9M.

[9] E. Rijcken, F. Scheepers, K. Zervanou, M. Spruit, P. Mosteiro, and U. Kaymak. Towards interpreting topic models with chatgpt. 2023. The 20th World Congress of the International Fuzzy Systems Association, IFSA ; Conference date: 20-08-2023 Through 24-08-2023.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.