



UNIVERSITÀ DEGLI STUDI DI BARI "ALDO MORO"

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA IN INFORMATICA

TESI DI LAUREA
IN
SISTEMI AD AGENTI

**TECNICHE DI FINE-TUNING DI LLM PER IL
QUESTION-ANSWERING SUI BANDI DELLA REGIONE
PUGLIA**

Relatore:

Prof. Berardina Nadja DE CAROLIS

Laureando:

Giacomo SIGNORILE
matr. 704897

ANNO ACCADEMICO 2022/2023

«Artificial intelligence is not just another big invention, but the mother of all inventions, capable of spawning a cascade of new technologies that redefine what is possible.»

Kevin Kelly

Indice

Elenco delle figure	VII
Elenco delle tabelle	IX
Introduzione	1
1 Introduzione ai Large Language Models (LLMs)	3
1.1 Breve Storia e Evoluzione degli LLM	3
1.2 Caratteristiche principali degli LLMs	4
1.2.1 Tokenizzazione	5
1.3 Architettura	5
1.3.1 Meccanismi di Attention	6
1.4 Implicazioni e Applicazioni	7
1.5 LLMs vs Chat Model vs Embedding Models	8
1.5.1 LLMs	8
1.5.2 Chat Models	8
1.5.3 Embeddings Models	8
1.6 Sfide e Considerazioni Future	9
2 Full Fine-Tuning	11
2.1 Full Fine-tuning in 6 step	11
2.1.1 Cosa rende il full fine-tuning vantaggioso?	12
2.1.2 Gli svantaggi del full fine-tuning	13
2.2 Nuove tecniche e metodologie	13
2.2.1 Tecniche di Data Augmentation per il miglioramento del fine-tuning	13
2.2.2 Paraphrasing Automatico	14
2.2.3 Generazione Sintetica di Testi	14
2.3 Fine-Tuning di ChatGPT: Ottimizzazione dei Modelli OpenAI	14
2.3.1 Fasi del Fine-Tuning	15
2.3.2 Modelli Abilitati per il Fine-Tuning	15
2.3.3 Quando Utilizzare il Fine-Tuning	15
2.3.4 Casi Comuni di Utilizzo del Fine-Tuning	16
2.3.5 Preparazione del Dataset per il Fine-Tuning	16
2.3.6 Gestione dei Limiti di Token	16
2.3.7 Verifica della Formattazione dei Dati	16

3	Parameter-efficient fine-tuning (PEFT)	17
3.1	LoRA	17
3.2	Vantaggi di LoRA	18
3.3	Metodo VeRA	18
3.3.1	Avanzamenti rispetto a LoRA	18
3.4	Dove PEFT batte il Full Fine-Tuning	19
4	Prompt-Engineering	21
4.1	Prompting Zero-shot	21
4.2	Prompting Few-shot	22
4.3	Prompting Chain-of-thought	22
4.4	Limitazioni e Sfide	23
4.5	Applicazioni nel progetto	23
5	Retrieval Augmented Generation(RAG)	25
5.1	Importanza nel Contesto Attuale dei Modelli di Linguaggio	25
5.2	Come funziona la RAG?	25
5.3	Casi d'uso RAG	26
5.4	Vantaggi della RAG	27
5.5	Limitazioni della RAG	28
6	Scelta del giusto approccio per l'applicazione	29
6.1	Costo	29
6.1.1	Prompt-Engineering	29
6.1.2	RAG	29
6.1.3	PEFT	29
6.1.4	Full fine-tuning	30
6.2	Complessità dell'implementazione	30
6.2.1	Prompt-Engineering	30
6.2.2	RAG	30
6.2.3	PEFT e Full fine-tuning	30
6.3	Accuratezza	30
6.3.1	Terminologia specifica del dominio	31
6.4	Approccio Implementato	31
7	Inizio del progetto	33
7.1	Raccolta dei Documenti	33
7.2	Estrazione del Testo dai PDF	33
7.2.1	Libreria PyPDF2	33
7.3	Organizzazione del Dataset	34
7.4	Gestione dei Documenti di Grandi Dimensioni	35
7.4.1	Libreria LangChain	36
7.4.2	Text Splitter	36

7.4.3	Generazione Automatica del Dataset con l'API di OpenAI	38
7.5	Fine-Tuning: Procedura e Risultati Metrici	40
7.6	Evaluations	43
7.6.1	Davinci-002 fine-tuned con 4 epoche	43
7.6.2	Davinci-002 fine-tuned con 6 epoche	45
7.6.3	Fine-Tuning di GPT-3.5 Turbo	47
7.6.4	Confronto RAG e Fine-Tuning nel chatbot sui Bandi	49
	Pinecone	49
7.6.5	Query Retrieval	50
8	Applicazione chatbot: BandiPugliaBot	51
8.1	Streamlit	51
8.2	Classe chatbt	54
8.3	Questionario Valutazione Usabilità del Chatbot	56
	Conclusioni	57
	Implicazioni e Riflessioni	57
	Sviluppi Futuri	57
	Bibliografia	61

Elenco delle figure

2.1	Fine-Tuning flow : Figura presa da altro autore [8]	11
7.1	Istogramma che raffigura la frequenza del numero della lunghezza di token nei chunk	35
8.1	Streamlit Interface per BandiPugliaBot	53

Elenco delle tabelle

7.1	Tabella relativa alla stima dei costi di Fine-Tuning	41
7.2	Tabella relativa alle metriche di addestramento del fine-tuning di OpenAI	42
7.3	Tabella che rappresenta il confronto tra le risposte date dal modello davinci-002 con 4 epoche	44
7.4	Tabella relativa a Domande e Risposte generate dal modello	45
7.5	Tabella contenente prompt e risposta del modello fine-tuned con 6 epoche con completion non migliorata	46
7.6	Tabella con risposte generate dalla Rag e dal modello fine-tuned con gpt-3.5-turbo e davinci-002	48

Introduzione

L'introduzione di questa tesi si colloca all'incrocio tra la crescente attrazione per i Large Language Models (LLMs) come ChatGPT e la necessità imperativa di adattarli per affrontare sfide specifiche e complesse nel contesto aziendale, nella vita quotidiana o in aiuto a utenti di fronte a una marea di informazione indecifrabile. Gli obiettivi di questa ricerca sono quelli di migliorare alcuni problemi degli LLMs come:

1. **Output Personalizzati:** Alcune applicazioni richiedono strutture o stili unici, come un tool di valutazione della scrittura che fornisce un feedback conciso a punti per gli elaborati;
2. **Contesto Mancante:** Gli LLMs preaddestrati potrebbero non essere a conoscenza di documenti specifici cruciali per un'applicazione, compromettendo l'accuratezza. Ad esempio, nel caso di un chatbot che risponda a domande sui bandi potrebbe avere difficoltà senza l'accesso ai nuovi documenti caricati ultimamente;
3. **Vocabolario Specializzato:** Alcuni settori, industrie e aziende utilizzano terminologie e concetti unici non ben rappresentati nei dati di preaddestramento generale. Questo può limitare la capacità di un LLM preaddestrato nel riassumere dati finanziari, interpretare documenti di ricerca medica o comprendere la terminologia presente nei bandi pubblici.

Nel primo capitolo della tesi verrà introdotto cos'è un Large Language e Model e nei quattro capitoli successivi verranno esplorati quattro metodi prominenti di fine-tuning o miglioramento di LLM:

1. **Full Fine-tuning:** Regola tutti i parametri degli LLMs utilizzando dati specifici del compito;
2. **Parameter-efficient Fine-tuning (PEFT):** Modifica la selezione di parametri per un'addestramento più efficiente;
3. **Prompt Engineering:** Perfeziona l'input del modello per guidare l'output;
4. **RAG (Retrieval Augmented Generation):** Combina prompt engineering con la consultazione(retrieval) del database per risposte generate dal contesto.

Questi metodi differiscono in termini di competenze richieste, costi e idoneità per scenari diversi. Il progetto di tesi si è incentrato sul modellare un sistema di risposta automatizzato, mettendo a frutto le potenzialità del modello linguistico GPT per fornire

risposte precise e contestualizzate alle domande poste dagli utenti sui bandi regionali. Gli obiettivi prefissati sono stati molteplici: personalizzare il modello GPT per adattarlo specificamente al corpus dei bandi regionali, garantire risposte immediate e pertinenti tramite un Chatbot dedicato, il tutto mantenendo una rigorosa ottimizzazione dei costi e l'efficienza del processo di fine-tuning.

Capitolo 1

Introduzione ai Large Language Models (LLMs)

I Large Language Models (LLMs) rappresentano una classe avanzata di modelli di intelligenza artificiale progettati per comprendere, generare e interagire con il linguaggio umano a livelli di complessità senza precedenti. Questi modelli, come illustrato in una panoramica comprensiva da Naveed et al. [18], sono stati all'avanguardia delle ricerche nel campo del Natural Language Processing (NLP) e sono stati progettati per comprendere, interpretare, generare e tradurre testi in linguaggio naturale. La loro "grandezza" (large) non deriva solo dalla dimensione del modello in termini di numero di parametri, ma anche dall'enorme quantità di dati su cui vengono addestrati. Questi modelli apprendono autonomamente le strutture linguistiche e le relazioni semantiche dai dati, permettendo loro di generare testi coerenti e pertinenti, rispondere a domande, riassumere documenti e molto altro.

1.1 Breve Storia e Evoluzione degli LLM

L'evoluzione degli LLM ha avuto inizio con modelli relativamente semplici come Word2Vec e GloVe, che rappresentavano le parole in spazi vettoriali. Questo permetteva operazioni matematiche semplici in grado di catturare le relazioni semantiche e sintattiche tra le parole. Tuttavia, questi primi modelli presentavano una limitazione notevole: faticavano a comprendere il contesto in cui venivano utilizzate le parole.

L'introduzione dell'architettura Transformer nel 2017, attraverso il paper "Attention is All You Need" di Vaswani et al. [25], ha rivoluzionato il campo dei Language Models, cambiando il modo in cui potevano elaborare il testo, consentendo così agli LLM di considerare l'intero contesto per generare risposte o completamenti. Questo ha portato allo sviluppo di modelli sempre più sofisticati come GPT (Generative Pre-trained Transformer) di OpenAI, BERT (Bidirectional Encoder Representations from Transformers) di Google, e molti altri, ognuno con i propri miglioramenti e specializzazioni.

Questo ha portato così allo sviluppo di modelli più sofisticati come:

- **GPT**(Generative Pre-trained Transformer) introdotto usando un articolo intitolato "Improving Language Understanding by Generative Pre-Training" da Radford e Narasimhan [23], esso ha segnato un importante progresso per la sua capacità di generare testo coerente al contesto dopo esser stato pre-addestrato su un vasto dataset di testo. OpenAI ha sviluppato e rilasciato versioni più avanzate, inclusi GPT-2 nel 2019 e GPT-3 nel 2020, ciascuna che andasse a migliorare la dimensione, la capacità e complessità di GPT.
- **BERT** (Bidirectional Encoder Representations from Transformers) è stato introdotto in un articolo intitolato "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", pubblicato da Devlin et al. [4], ha introdotto numerosi novità nel NLP per la sua capacità di comprendere il contesto delle parole in una frase in modo bidirezionale, ovvero osservando sia le parole che lo precedono sia le parole che seguono nella frase, ciò è ottenuto assegnando dei pesi a ciascuna parola nella frase.

1.2 Caratteristiche principali degli LLMs

Gli LLMs si distinguono per la loro enorme capacità di memorizzazione e la complessità dei loro meccanismi di apprendimento. Sono allenati su vasti dataset di testo che spaziano attraverso una vasta gamma di domini e stili, permettendo loro di catturare sottili sfumature linguistiche e varietà di conoscenza umana. Come riportato da Naveed et al. [18]:

- **Architettura:** Data l'enorme dimensione degli LLMs, piccole modifiche nell'architettura e nelle strategie di addestramento hanno un grande impatto sulle prestazioni e sulla stabilità. Si sottolinea l'importanza dei moduli architettonici chiave utilizzati in vari LLMs, che portano a migliori prestazioni, riduzione del tempo e della memoria di addestramento e maggiore stabilità.
- **Normalizzazione dei Livelli:** Ha un effetto significativo sulle prestazioni e sulla stabilità degli LLMs. La pre-norm (normalizzazione degli input piuttosto che degli output) è più comune tra gli LLMs per stabilizzare l'addestramento.
- **Attention:** è un meccanismo che permette al modello di focalizzarsi su parti specifiche dell'input mentre elabora o genera testo, migliorando la sua capacità di comprendere e produrre linguaggio naturale in modo contestualmente rilevante.
- **Miscela di Esperti (MoE):** Permette di scalare facilmente il modello a trilioni di parametri, attivando solo pochi esperti durante il calcolo, rendendoli efficienti dal punto di vista computazionale.
- **Strategie di Addestramento:** Addestrare modelli su vasta scala richiede accorgimenti per ridurre i costi di addestramento, evitare divergenze di perdita e ottenere migliori prestazioni.

- **Precisione Mista:** È un metodo famoso per ridurre l'uso della memoria e migliorare l'efficienza di addestramento degli LLMs.
- **Instabilità dell'Addestramento:** È un problema comune negli LLMs, dove si osservano divergenze o picchi di perdita durante l'addestramento.
- **Inizializzazione dei Pesi:** Gioca un ruolo cruciale nella convergenza del modello e nella stabilità dell'addestramento.
- **Modelli Supervisionati vs Modelli Generalizzati:** Sebbene i modelli generalizzati siano capaci di eseguire compiti diversi con buone prestazioni, non hanno ancora superato i modelli addestrati in contesti supervisionati.
- **Zero-Shot vs Few-Shot:** Gli LLMs si comportano bene sia in contesti zero-shot (un singolo esempio) che few-shot (un limitato numero di esempi labellati), ma la differenza di prestazione tra i due è significativa.
- **Encoder vs Decoder vs Encoder-Decoder:** Tradizionalmente, queste architetture si comportano bene per compiti diversi. Modelli solo-encoder sono noti per modelli più piccoli, mentre gli LLMs sono o solo-decoder o encoder-decoder. Varie variazioni nell'architettura e negli obiettivi di addestramento consentono a un modello di esibirsi bene in diversi contesti.

Questa sezione evidenzia l'importanza dell'architettura, delle strategie di addestramento e della configurazione dei modelli per ottimizzare le prestazioni e la stabilità degli LLMs. Mostra anche come piccole modifiche possono avere impatti significativi e come la scelta tra diversi approcci architettonici e di addestramento possa influenzare le capacità del modello in vari compiti di NLP.

1.2.1 Tokenizzazione

La tokenizzazione è essenziale per preparare i dati di input per gli LLMs, convertendo il testo in unità più piccole come parole o sub-parole. Tecniche come WordPiece, Byte Pair Encoding (BPE) e Unigram Language Model sono state ampiamente adottate [18].

1.3 Architettura

Nel documento di Zhao et al. [26] sono presentati diverse architecture design degli LLMs, grazie allo sviluppo dei Transformers che riescono a far sviluppare con miliardi di parametri, vengono descritte principalmente tre architetture:

- **Encoder-decoder Architecture:** usa due blocchi di Transformers, come encoder e decoder, rispettivamente, l'encoder adotta dei layers di multi-head self-attention per tradurre la sequenza di input e generare la sua rappresentazione, mentre il decoder processa una cross-attention su questa rappresentazione per generare la sequenza di target.

- **Casual Decoder Architecture:** Utilizza un layer di attention unidirezionale per assicurare che ogni token di input possa interfacciarsi con i token passati o l'input stesso. I modelli di OpenAI sono sviluppati basandosi su questa architettura.
- **Prefix Decode Architecture:** Modifica il meccanismo di masking dei Casual Decoder per permettere un'attention bidirezionale sui token del prefisso e un'attention unidirezionale solo sui token generati.
- **Mixture-of-Expert(MoE):** Questo approccio prende tutte e tre le architetture precedenti e le estende con un sottoinsieme di pesi della rete neurale che per ogni input è attivato in maniera sparsa, permettendo di aumentare la dimensione del modello mantenendo gli stessi costi.

Nuove architetture emergenti puntano a migliorare l'inferenza con input lunghi, incorporando meccanismi di aggiornamento ricorsivo, offrendo il vantaggio di generare token facendo fede allo stato precedente e alla codifica di frasi in parallelo come i Transformer, beneficiando del parallelismo delle GPU.

1.3.1 Meccanismi di Attention

Gli LLMs utilizzano l'Attention per ponderare l'importanza delle diverse parti dell'input. Questo approccio ha notevolmente migliorato la capacità dei modelli di comprendere e generare testi complessi come discusso da Zhao et al. [26].

- **Full Attention:** Nell'architettura Transformer originale, l'attention viene applicata considerando tutte le coppie di token nella sequenza mediante l'attention a prodotto scalare normalizzato. Utilizza inoltre l'attention multi-head, che proietta query, chiavi e valori in diverse "head" per catturare informazioni. L'output di ciascuna 'head' viene poi concatenato per formare l'output finale.
- **Sparse Attention:** Affronta la sfida della complessità quadratica $O(n^2)$ dell'attention completa, che diventa problematica con lunghe sequenze. Vari modelli più efficienti di Transformer sono stati proposti per ridurre questa complessità. Ad esempio, la sparse attention local banded è stata adottata in GPT-3, consentendo a ogni query di riferire solo a un sottoinsieme di token in base alle loro posizioni.
- **Multi-query/Grouped-query Attention:** In questa variante, diverse heads condividono le stesse matrici di trasformazione lineare per chiavi e valori, aumentando la velocità di inferenza con un minimo sacrificio nella qualità del modello. L'attention grouped/query assegna le heads a diversi gruppi che condividono le stesse matrici di trasformazione, esplorata nel modello LLaMA 2.
- **Flash Attention:** Ottimizza la velocità e il consumo di memoria dei moduli di attention su GPU, organizzando l'input in blocchi e introducendo la ricomputazione necessaria per utilizzare la memoria SRAM che è più veloce. La Flash Attention è stata implementata come un kernel fuso in CUDA e integrato in PyTorch.

- **Paged Attention:** Migliora l'efficienza della memoria e il throughput degli LLMs distribuiti, partizionando ogni sequenza in sottosequenze. Questa tecnica aumenta l'utilizzo della GPU e consente una condivisione efficiente della memoria nel campionamento parallelo.

1.4 Implicazioni e Applicazioni

L'impiego di LLMs ha trovato terreno fertile in numerosi ambiti applicativi, rivoluzionando il modo in cui interagiamo con la tecnologia. Dalla creazione di assistenti virtuali capaci di comprendere e rispondere a richieste complesse, alla generazione automatica di testi per la creazione di contenuti, gli LLMs stanno ridefinendo le potenzialità delle macchine di "comprendere" e utilizzare il linguaggio umano in modi sempre più sofisticati.

- **Salute e Medicina:** Gli LLMs stanno trasformando l'assistenza sanitaria, offrendo supporto decisionale in oncologia, migliorando la comprensione della letteratura medica e assistendo nelle decisioni cliniche. Sono stati esplorati per migliorare l'educazione medica e la collaborazione medico-paziente.
- **Educazione:** Gli LLMs offrono opportunità per migliorare l'istruzione, fornendo feedback personalizzato agli studenti e supportando l'apprendimento delle lingue. Tuttavia, esistono sfide legate alla qualità dell'insegnamento e considerazioni etiche riguardo il loro impiego nell'educazione.
- **Legge:** Gli LLMs sono utilizzati per analizzare testi legali, fornire spiegazioni e supportare la ricerca legale. Benché promettenti, ci sono preoccupazioni riguardanti l'accuratezza e la responsabilità nell'uso degli LLMs in contesti legali.
- **Finanza:** Gli LLMs trovano applicazione nell'analisi finanziaria e nella consulenza, offrendo potenziali benefici in termini di efficienza e precisione. Tuttavia, la loro implementazione richiede cautela per garantire decisioni finanziarie affidabili.
- **Interazione Uomo-Macchina:** Gli LLMs migliorano l'interazione uomo-macchina, promuovendo la fiducia e l'efficacia nella collaborazione. Sono esplorati per pianificazione di compiti e movimenti, dimostrando il potenziale nel migliorare l'assistenza personalizzata tramite robot.
- **Considerazioni Energetiche e Politiche:** L'impiego di LLMs solleva questioni relative al consumo energetico e alle politiche di regolamentazione, specialmente nel contesto del deep learning in NLP. È fondamentale affrontare queste preoccupazioni per garantire uno sviluppo sostenibile e responsabile degli LLMs.
- **Privacy e Etica:** Gli LLMs pongono sfide etiche significative, incluse preoccupazioni sulla privacy e il rischio di generare contenuti polarizzanti o inesatti. La trasparenza, la regolamentazione e le strategie di mitigazione sono cruciali per navigare queste questioni.

- **Ricerca e Sviluppo Futuro:** La ricerca continua a esplorare le capacità e le applicazioni degli LLMs, con un'enfasi crescente sulla sicurezza, l'affidabilità e l'etica. La collaborazione tra ricercatori, sviluppatori e regolatori sarà vitale per realizzare il pieno potenziale degli LLMs in modo responsabile.

1.5 LLMs vs Chat Model vs Embedding Models

Questi modelli, pur essendo interconnessi, servono a scopi diversi e presentano architetture e applicazioni uniche. La comprensione delle loro differenze è cruciale per applicare l'AI in maniera efficace e innovativa.

1.5.1 LLMs

Gli LLMs come GPT di OpenAI, e come discusso nel capitolo, sono capaci di generare testo e rispondere a domande contestualmente rilevanti.

1.5.2 Chat Models

I Chat Models, o modelli conversazionali, sono una sottocategoria degli LLMs specificamente addestrati o ottimizzati per la generazione di dialoghi. Questi modelli, come GPT-3.5-Turbo quando applicato a chatbot, mirano a produrre conversazioni fluide e naturali. Mentre gli LLMs generali possono eseguire una vasta gamma di compiti di NLP, i Chat Models sono particolarmente tunati per capire le sfumature del dialogo, per gestire gli scambi conversazionali e mantenerne la coerenza nel corso del dialogo.

1.5.3 Embeddings Models

Un embedding è un vettore (lista) di numeri in virgola mobile. La distanza tra due vettori misura il loro grado di correlazione. Distanze piccole suggeriscono un'alta correlazione mentre distanze grandi indicano una bassa correlazione.

Gli Embedding Models si concentrano sulla rappresentazione vettoriale delle parole o delle frasi in uno spazio continuo. Questi modelli catturano la semantica e le relazioni tra parole basandosi sull'occorrenza nel testo. A differenza degli LLMs e dei Chat Models, che possono generare testo o rispondere a domande, gli Embedding Models sono utilizzati principalmente per trasformare il testo in vettori che possono poi essere utilizzati per compiti come la classificazione del testo, il clustering o il recupero di informazioni. La loro forza risiede nella capacità di catturare il significato contestuale delle parole e di utilizzare queste rappresentazioni per migliorare le prestazioni dei modelli downstream in vari compiti di NLP.

Come spiegato da OpenAI [21] i text-embeddings di OpenAI misurano il grado di correlazione tra stringhe di testo. Essi sono comunemente utilizzati per:

- Ricerca (dove i risultati sono ordinati per rivelanza rispetto a una stringa di query)
- Clustering (dove le stringhe di testo sono raggruppate in base alla similarity)
- Referenze (dove vengono consigliati gruppi di testo con stringhe correlate)
- Rilevamento di anomalie(dove vengono analizzate le distribuzioni di somiglianza)
- Classificazione (dove le stringhe di testo sono classificate in base alla loro etichetta più simile)

1.6 Sfide e Considerazioni Future

Nonostante i notevoli successi, lo sviluppo e l'impiego di LLMs sollevano anche importanti questioni etiche e tecniche, inclusa la gestione della privacy dei dati, l'allineamento dei modelli con i valori umani e la necessità di risorse considerevoli per l'addestramento, il rischio di generazione di informazioni false o fuorvianti e la necessità di mitigare i bias nei modelli di apprendimento. La ricerca futura dovrà quindi concentrarsi non solo sull'espansione delle capacità degli LLMs ma anche sul garantire che il loro impiego sia responsabile e benefico per la società. [26, 10].

Capitolo 2

Full Fine-Tuning

2.1 Full Fine-tuning in 6 step

Il fine-tuning è un processo di apprendimento automatico che consente di migliorare le prestazioni di un modello di apprendimento automatico pre-addestrato su nuovi esempi. In particolare, il fine-tuning consente di adattare il modello alle caratteristiche specifiche del nuovo compito, migliorando la sua accuratezza e robustezza.

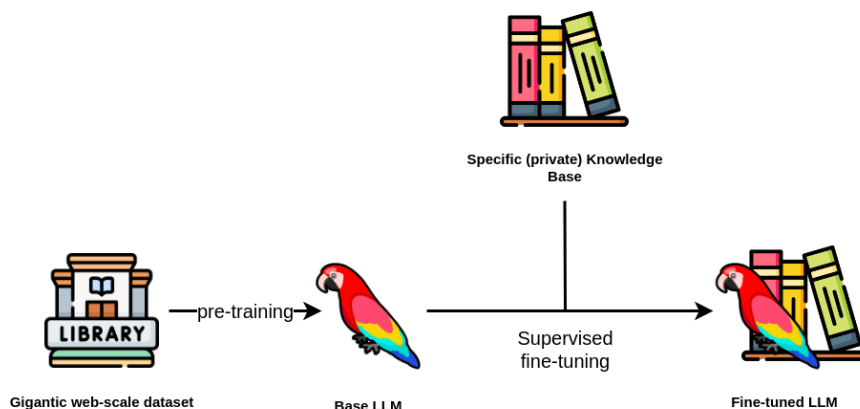


FIGURA 2.1: Fine-Tuning flow : Figura presa da altro autore [8]

Nel caso del Chatbot per i Bandi in corso della Regione Puglia, per eseguire il full fine-tuning, è necessario seguire i seguenti passi:

1. Creare il dataset:

- Raccogliere tutti i documenti pdf riguardanti i bandi attivi della Regione Puglia, assicurandosi che ciascun articolo sia corredato del suo abstract originale.
- Suddividere questa raccolta in chunk, ovvero pezzi di documenti più piccoli in modo da venir processati meglio.

2. Preprocessare i dati:

- Convertire ciascun documento di ricerca in un formato adatto al modello, ovvero seguire la struttura di addestramento, che può essere un file JSON con 'prompt' e 'completion'.
- Associare il contenuto di ciascun documento processato con il suo abstract corrispondente per formare coppie input-output per l'addestramento supervisionato.

3. Configurare il modello:

- Caricare il Large Language Model (LLM) preaddestrato (ad esempio, una versione preaddestrata di GPT-4).
- Decidere gli iperparametri per il fine-tuning, come il learning rate, la dimensione del batch e il numero di epoche, basandosi su test preliminari o conoscenze di dominio.

4. Addestrare il modello:

- Alimentare il contenuto processato al LLM come input e addestrarlo a generare l'abstract corrispondente come output.
- Monitorare le prestazioni del modello sul set di validazione per prevenire l'overfitting e decidere quando interrompere l'addestramento o apportare eventuali aggiustamenti.

5. Valutare le prestazioni:

- Una volta completato il fine-tuning, valutare le prestazioni del modello sul set di test, che non ha mai visto prima.
- Le metriche potrebbero includere il punteggio BLEU, il punteggio ROUGE o valutazioni umane per misurare la qualità e la pertinenza degli abstract generati rispetto a quelli originali.

6. Iterare fino a ottenere prestazioni soddisfacenti:

- Sulla base dei risultati della valutazione, iterare sui passaggi precedenti, eventualmente raccogliendo ulteriori dati, regolando gli iperparametri o provando diverse configurazioni del modello per migliorare le prestazioni.

2.1.1 Cosa rende il full fine-tuning vantaggioso?

- **Richiede meno dati rispetto all'addestramento da zero:** Il full fine-tuning può essere efficace anche con un dataset specifico per il compito relativamente piccolo. Il LLM preaddestrato già comprende costrutti linguistici generali, e il processo di fine-tuning si concentra principalmente sull'adattamento della conoscenza del modello alle specificità dei nuovi dati. Un LLM preaddestrato, inizialmente addestrato

su circa 1 trilione di token e con una solida performance generale, può essere efficientemente fine-tunato con solo alcuni centinaia di esempi, equivalenti a diverse centinaia di migliaia di token.

- **Miglioramento dell'accuratezza:** Attraverso il fine-tuning su un dataset specifico per il compito, il LLM può cogliere le sfumature di quel particolare dominio. Questo è particolarmente cruciale in settori con gergo, concetti o strutture specializzate, come documenti legali, testi medici o relazioni finanziarie. Di conseguenza, di fronte a esempi non visti dal dominio o dal compito specifico, il modello è probabile che effettui predizioni o generi output con maggiore accuratezza e rilevanza.
- **Aumento della robustezza:** Il fine-tuning consente di esporre il modello a più esempi, in particolare casi limite o scenari meno comuni nel dataset specifico del dominio. Ciò rende il modello più preparato per gestire una vasta gamma di input senza produrre output erronei.

2.1.2 Gli svantaggi del full fine-tuning

- **Alti costi computazionali:** Il full fine-tuning implica l'aggiornamento di tutti i parametri di un grande modello. Con LLM su larga scala che vantano decine o centinaia di miliardi di parametri, l'addestramento richiede enormi quantità di potenza di calcolo. Anche quando il dataset di fine-tuning è relativamente piccolo, il numero di token può essere elevato e costoso da calcolare.
- **Requisiti di memoria sostanziali:** Lavorare con modelli di grandi dimensioni può richiedere hardware specializzato, come GPU o TPU di fascia alta, con capacità di memoria significative. Questo è spesso impraticabile per molte aziende.
- **Tempo e competenze intensive:** Quando il modello è molto grande, spesso è necessario distribuire il calcolo su più GPU e nodi. Ciò richiede competenze adeguate. A seconda delle dimensioni del modello e del dataset, il fine-tuning può richiedere ore, giorni o addirittura settimane.

2.2 Nuove tecniche e metodologie

2.2.1 Tecniche di Data Augmentation per il miglioramento del fine-tuning

Ultimamente è cresciuta esponenzialmente la ricerca di tecniche innovative per la Data Augmentation, ovvero per arricchire i dati da dare in pasto per l'addestramento. Esse si sono dimostrate cruciali per superare una delle sfide più significative nel campo: la disponibilità limitata di dataset di alta qualità per l'addestramento dei modelli. Nel lavoro svolto da Feng et al. [5] viene sottolineato come l'adozione di strategie di Data Augmentation, abbia portato a miglioramenti notevoli nelle prestazioni dei modelli su compiti

specifici. Questo approccio non solo arricchisce i set di dati esistenti ma offre anche una soluzione pratica al problema della raccolta manuale di nuovi dati.

2.2.2 Paraphrasing Automatico

Il paraphrasing automatico si rivela una tecnica efficace nella Data Augmentation per la sua capacità di generare diverse varianti linguistiche di una stessa frase o paragrafo, mantenendo inalterato il significato originale. Questa tecnica migliora la robustezza dei modelli di apprendimento automatico permettendo loro di interpretare e comprendere meglio le variazioni linguistiche naturali. Implementando algoritmi di paraphrasing automatico, i ricercatori possono espandere in modo significativo il volume e la diversità dei dati di addestramento disponibili, migliorando così l'accuratezza e l'affidabilità dei modelli.

2.2.3 Generazione Sintetica di Testi

La generazione sintetica di testi, un'altra pietra miliare nella Data Augmentation, utilizza tecniche avanzate di intelligenza artificiale per produrre testi nuovi che rispecchiano la varietà e la complessità dei dati di addestramento originali. Questo metodo non solo amplia il dataset di addestramento ma introduce anche una ricchezza di scenari, esempi e contesti nuovi per i modelli, potenziando la loro capacità di generalizzazione e abilità di variare il testo generato. La generazione sintetica di testi si affida a modelli di linguaggio avanzati, che, attraverso l'apprendimento dalle immense collezioni di testi disponibili, sono capaci di produrre contenuti verosimili e coerenti con gli esempi di addestramento.

2.3 Fine-Tuning di ChatGPT: Ottimizzazione dei Modelli OpenAI

[19]

Il processo di Fine-Tuning rappresenta una fase essenziale nell'ottimizzazione delle prestazioni dei modelli API di OpenAI. Tale procedura offre diversi vantaggi rispetto ai tradizionali prompt, consentendo:

1. Risultati di qualità superiore
2. Capacità di addestrare su un maggior numero di esempi rispetto a quanto possa contenere un prompt
3. Risparmi di token grazie a prompt più brevi
4. Richieste con latenza inferiore

I modelli di generazione di testo di OpenAI sono preaddestrati su un vasto corpus di testo, richiedendo istruzioni e talvolta diversi esempi in un prompt per un'applicazione efficace. Questo approccio, noto come *few-shot learning*, dimostra come eseguire un'attività.

Il Fine-Tuning migliora il few-shot learning addestrando il modello su un numero maggiore di esempi, consentendo risultati migliori su un'ampia gamma di compiti. Una volta completato il Fine-Tuning, il numero di esempi richiesti nei prompt diminuisce, riducendo i costi e consentendo richieste con latenza inferiore.

2.3.1 Fasi del Fine-Tuning

A livello pratico, il processo di Fine-Tuning coinvolge le seguenti fasi:

1. Preparazione e upload dei dati di addestramento
2. Addestramento di un nuovo modello fine-tuned
3. Valutazione dei risultati e, se necessario, ritorno alla fase 1
4. Utilizzo del modello fine-tuned

2.3.2 Modelli Abilitati per il Fine-Tuning

Attualmente, il Fine-Tuning è disponibile per modelli specifici, tra cui:

- gpt-3.5-turbo-1106 (raccomandato)
- gpt-3.5-turbo-0613
- babbage-002
- davinci-002
- gpt-4-0613 (sperimentale)

GPT-3.5-turbo è generalmente consigliato per la maggior parte degli utenti in termini di risultati e facilità d'uso.

2.3.3 Quando Utilizzare il Fine-Tuning

Il Fine-Tuning può migliorare le prestazioni dei modelli di generazione di testo OpenAI per applicazioni specifiche, ma richiede un investimento attento di tempo ed energie. Prima di ricorrere al Fine-Tuning, si consiglia di esplorare strategie come ingegneria del prompt, concatenazione di prompt e chiamate di funzioni, poiché:

- Alcuni compiti possono sembrare inizialmente poco performanti, ma risultati migliori possono essere ottenuti con prompt adeguati, evitando la necessità di Fine-Tuning.
- La rapidità di feedback è maggiore iterando su prompt rispetto al Fine-Tuning, che richiede la creazione di dataset e l'esecuzione di lavori di addestramento.
- Nel caso in cui il Fine-Tuning sia ancora necessario, il lavoro iniziale sull'ingegneria del prompt non è vanificato e può contribuire ai risultati finali.

2.3.4 Casi Comuni di Utilizzo del Fine-Tuning

Alcuni scenari comuni in cui il Fine-Tuning può migliorare i risultati includono:

- Definire stile, tono, formato e altri aspetti qualitativi
- Migliorare l'affidabilità nella produzione di output desiderati
- Correggere errori nel seguire prompt complessi
- Gestire casi limite in modi specifici
- Apprendere nuove abilità o compiti difficili da esprimere in un prompt

2.3.5 Preparazione del Dataset per il Fine-Tuning

Una volta deciso che il Fine-Tuning è la soluzione appropriata, è necessario preparare il dataset di addestramento. Questo deve comprendere conversazioni di dimostrazione simili a quelle con cui si chiederà al modello di rispondere nella produzione. Il formato del dataset varia in base al modello selezionato, ad esempio:

- Per gpt-3.5-turbo, è richiesto un formato di chat conversazionale.
- Per babbage-002 e davinci-002, è richiesta una coppia di prompt e completamento.

2.3.6 Gestione dei Limiti di Token

I limiti di token dipendono dal modello selezionato. Ad esempio, gpt-3.5-turbo-1106 ha una lunghezza massima del contesto di 4096 token per esempio di addestramento. Per stimare i costi, è possibile utilizzare la formula fornita, considerando il costo base per 1.000 token, il numero di token nel file di input e il numero di epoche addestrate.

2.3.7 Verifica della Formattazione dei Dati

Prima di creare un lavoro di Fine-Tuning, è fondamentale verificare la formattazione dei dati. A tal fine, è stato creato uno script Python che facilita questa verifica, individuando potenziali errori, contando i token e stimando i costi del lavoro di Fine-Tuning.

Capitolo 3

Parameter-efficient fine-tuning (PEFT)

Negli ultimi anni, il campo del Natural Language Processing (NLP) ha assistito a una crescita nello sviluppo e nell'implementazione di language model pre-trained su larga scala. Questi modelli, come GPT-3 con i suoi impressionanti 175 miliardi di parametri addestrati, si sono dimostrati altamente efficaci in varie applicazioni NLP. Dunque l'utilizzo e l'adattamento di questi enormi modelli per i diversi compiti specifici rappresenta una enorme sfida, specialmente quando si effettua un fine-tuning.

La Parameter-efficient fine-tuning (PEFT) utilizza tecniche per affinare ulteriormente un modello pre-addestrato aggiornando solo un piccolo numero dei suoi parametri totali [16]. Un grande modello di linguaggio pre-addestrato su enormi quantità di dati ha già appreso un ampio spettro di costrutti e conoscenze linguistiche. In breve, possiede già molte delle informazioni necessarie per molti compiti. Data la portata più piccola, spesso è inutile e inefficiente regolare l'intero modello. Il processo di sintonizzazione fine viene condotto su un piccolo insieme di parametri.

I metodi PEFT variano nel loro approccio per determinare quali componenti del modello sono addestrabili. Alcune tecniche danno priorità all'addestramento di porzioni selezionate dei parametri del modello originale. Altre integrano e addestrano componenti aggiuntive più piccole, come gli strati adattatori, senza modificare la struttura originale.

3.1 LoRA

LoRA, acronimo di Low-Rank Adaptation of Large Language Models, è stato introdotto all'inizio del 2023 [9]. Da allora è diventato il metodo PEFT più comunemente utilizzato, aiutando aziende e ricercatori a ridurre i loro costi di sintonizzazione fine. Utilizzando la riperammetrizzazione, questa tecnica riduce l'insieme di parametri addestrabili eseguendo un'approssimazione di basso rango [9].

Ad esempio, se abbiamo una matrice di pesi 100.000×100.000 , allora per la sintonizzazione fine completa abbiamo bisogno di aggiornare 10.000.000.000 parametri. Utilizzando LoRA, possiamo catturare tutte o la maggior parte delle informazioni cruciali utilizzando una matrice di basso rango che contiene i parametri selezionati aggiornati durante la sintonizzazione fine [9].

Per ottenere questa matrice di basso rango, possiamo rireparametrizzare la matrice di pesi originale in due matrici, A e B, ciascuna di basso rango r . La nostra nuova matrice di basso rango viene quindi presa come il prodotto di A e B. Se $r = 2$, finiamo per aggiornare $(100.000 \times 2) + (100.000 \times 2) = 400.000$ parametri invece di 10.000.000. Aggiornando un numero molto più piccolo di parametri, riduciamo i requisiti computazionali e di memoria necessari per la sintonizzazione fine [9].

3.2 Vantaggi di LoRA

- **Efficienza nel cambio di compito** - Creare diverse versioni del modello per compiti specifici diventa più facile. È possibile semplicemente memorizzare una singola copia dei pesi pre-addestrati e costruire molti piccoli moduli LoRA. Quando si passa da un compito all'altro si sostituiscono solo le matrici, A e B, e si mantiene il LLM. Questo riduce significativamente i requisiti di archiviazione.
- **Meno GPU richieste** - LoRA riduce i requisiti di memoria della GPU fino a 3 volte poiché non calcoliamo/riaddestriamo i gradienti per la maggior parte dei parametri.
- **Alta precisione** - Su una varietà di benchmark di valutazione, le prestazioni di LoRA si dimostrano quasi equivalenti alla sintonizzazione fine completa, a una frazione del costo.

3.3 Metodo VeRA

VeRA, che sta per Vector-based Random Matrix Adaptation, si distingue per il suo approccio unico nel ridurre il numero di parametri addestrabili, utilizzando una coppia di matrici di basso rango condivise attraverso tutti i layer del modello e apprendendo piccoli vettori di scaling. Questa strategia consente a VeRA di ridurre il numero di parametri addestrabili di un ordine di grandezza rispetto a LoRA, mantenendo allo stesso tempo prestazioni comparabili nei compiti di benchmark.[12]

3.3.1 Avanzamenti rispetto a LoRA

- **Riduzione del numero dei parametri significativa:** VeRA utilizza matrici low-rank condivise e vettori di scaling, per ridurre in modo significativo il numero di parametri di training, facilitando così il fine-tuning efficiente su hardware con risorse limitate.

- **Stesse prestazioni** - Nonostante la significativa riduzione dei parametri, Vera mantiene prestazioni e risultati paragonabili a metodi tradizionali di full-fine-tuning e rispetto anche alla tecnica LoRA.[12]
- **Versatilità** - Il metodo testato su una varietà di benchmark di valutazione, ha avuto successo nell'istruzione-following utilizzando il modello Llama2 7B, utilizzando invece di 7 miliardi, solamente 1.4 milioni di parametri. [12]

3.4 Dove PEFT batte il Full Fine-Tuning

- **Addestramento più efficiente e veloce** - Meno utilizzo di parametri significa meno calcoli, che si traducono direttamente in sessioni di training più veloci che richiedono meno potenza di calcolo e meno risorse di memoria. Questo rende il fine-tuning più pratico in scenari con risorse limitate.[6].
- **Preservazione della conoscenza dalla pre-addestramento** - Un pre-training su ampi dataset inculca nei modelli preziose conoscenze e capacità. Con il PEFT, viene assicurato che questo bagaglio di conoscenza non vada perso quando si adatta o specializza il modello a nuovi compiti perché viene mantenuta la maggior parte o tutti i pesi del modello originale [6].

Capitolo 4

Prompt-Engineering

Il Prompt-Engineering è emersa come una tecnica cruciale per massimizzare l'efficacia dei modelli di linguaggio di grandi dimensioni (LLM), come GPT-3.5-Turbo. Attraverso l'arte di formulare prompt, gli sviluppatori possono guidare i modelli a produrre risposte desiderate, sfruttando la loro capacità di generalizzare da pochi esempi e di eseguire ragionamenti di base. Questo capitolo esplora le metodologie di prompting zero-shot, few-shot, e chain-of-thought, evidenziando come possono essere applicate per migliorare le prestazioni dei modelli in vari contesti.

4.1 Prompting Zero-shot

Nel prompting zero-shot, il modello riceve una nuova task senza esempi precedenti. Questa tecnica sfrutta la capacità del modello di generalizzare basandosi sul suo addestramento pregresso. Un'applicazione tipica è lo sviluppo di chatbot di assistenza tecnica, dove un prompt chiaro e ben formulato può guidare il modello a fornire soluzioni tecniche pertinenti, anche senza esperienza diretta nella risoluzione di quel particolare problema, come ad esempio:

- Prompt: Fornisci una soluzione di assistenza tecnica basata sulla seguente preoccupazione dell'utente.
- Preoccupazione dell'utente: Il mio computer non si accende.
- Soluzione:

Proponendo un'istruzione alla query dell'utente ("Il mio computer non si accende"), viene dato al modello il contesto per il tipo di risposta desiderata. Questo è un modo per adattare l'output per l'assistenza tecnica anche senza esempi espliciti di soluzioni tecniche.

Nello studio di al. [3] viene dimostrata l'efficacia del prompting zero-shot nel guidare i modelli di linguaggio di grandi dimensioni come GPT-3, offrendo una panoramica delle potenzialità di questa tecnica nell'elaborazione del linguaggio naturale.

4.2 Prompting Few-shot

Nel prompting few-shot, il modello viene informato con alcuni esempi prima di affrontare un nuovo task. Questi esempi sono essenzialmente coppie di input di esempio e output del modello atteso.

Nel caso si crei un'app di salute che classifica i piatti in 'Basso contenuto di grassi' o 'Alto contenuto di grassi' utilizzando un modello di linguaggio. Per orientare il modello, un paio di esempi sono preposti alla query dell'utente:

- Classifica il seguente piatto in base al suo contenuto di grassi: Pollo alla griglia, limone, erbe.
- Risposta: Basso contenuto di grassi
- Classifica il seguente piatto in base al suo contenuto di grassi: Mac e cheese con panna pesante e burro.
- Risposta: Alto contenuto di grassi
- Classifica il seguente piatto in base al suo contenuto di grassi: Toast all'avocado con olio d'oliva
- Risposta:

Informato dagli esempi nel prompt, un LLM abbastanza grande e ben addestrato risponderà in modo affidabile: "Alto contenuto di grassi."

Nell'articolo proposto da [1] viene discussa l'importanza del contesto e degli esempi nell'ottimizzare le prestazioni dei language models, evidenziando come queste tecniche possano essere utilizzate per adattare i modelli a compiti specifici.

Il prompting few-shot è un buon modo per indurre il modello a un certo formato di risposta. Ad esempio la risposta del modello si potrebbe conformare a una certa struttura, indicando anche restrizioni di lunghezza.

4.3 Prompting Chain-of-thought

Il prompting chain-of-thought incoraggia il modello a scomporre un problema complesso in passaggi intermedi più gestibili. Insieme al prompting few-shot può migliorare le prestazioni su compiti che necessitano di un'analisi riflessiva.

Ad esempio:

- Sottraendo il numero più piccolo dal più grande in questo gruppo si ottiene un numero pari: 5, 8, 9. A: Sottraendo 5 da 9 si ottiene 4. La risposta è Vero.

- Sottraendo il numero più piccolo dal più grande in questo gruppo si ottiene un numero pari: 10, 15, 20. A: Sottraendo 10 da 20 si ottiene 10. La risposta è Vero.

Infatti, il prompting chain of thought può anche essere accoppiato con il prompting zero shot per migliorare le prestazioni su compiti che richiedono un'analisi passo-passo. Se si vuole migliorare le prestazioni del modello, si potrebbe chiedergli di scomporre la soluzione passo dopo passo.

Seguendo questi prompt:

- Scomponi la soluzione di assistenza tecnica passo dopo passo in base alla seguente preoccupazione dell'utente. Preoccupazione dell'utente: Il mio computer non si accende.
- Soluzione: Per una varietà di applicazioni, l'ingegneria del prompt di base di un LLM molto grande può fornire un'accuratezza 'sufficientemente buona'.

Fornisce un metodo di adattamento economico perché è veloce e non coinvolge grandi quantità di potenza di calcolo. Lo svantaggio è che semplicemente non è abbastanza accurato o robusto per i casi d'uso in cui è richiesta una conoscenza di background aggiuntiva.

Nell'articolo al. [2] è illustrato come il prompting chain-of-thought, dopo aver sperimentato tale tecnica su tre grandi language models, ha migliorato le prestazioni in un'ampia gamma di operazioni di aritmetica, di prompting e di ragionamenti simbolici. I guadagni ottenuti, sono stati notevoli, ad esempio il prompting di un PaLM 540B con solo otto esempi di chain-of-thought raggiunge un accuracy all'avanguardia su problemi di matematica testuale, superando perfino un modello GPT-3 fine-tuned.

4.4 Limitazioni e Sfide

Nonostante il potenziale del Prompt-Engineering, esistono limitazioni significative. Queste includono la difficoltà di formulare prompt efficaci per compiti specifici e la sfida di garantire che il modello generalizzi correttamente da un limitato numero di esempi. La ricerca futura è fondamentale per affrontare queste sfide e per esplorare nuovi metodi che possano rendere i modelli ancora più adattabili e precisi.

4.5 Applicazioni nel progetto

Questa metodologia sarà utilizzata per realizzare l'augmentation dei dati, ovvero per arricchire e generare dati sintetici. Impiegando un modello GPT-3 per l'analisi testuale, sono stati ricercati e implementati i prompt più efficaci, al fine di fornire risposte precise e complete. Questo approccio mira a migliorare il testo esistente o a creare nuovi esempi pertinenti al contenuto trattato.

Capitolo 5

Retrieval Augmented Generation(RAG)

Questo capitolo introduce la Retrieval Augmented Generation(RAG), esplorando le sue origini, la sua importanza nel panorama attuale dell'AI e la sua evoluzione nel tempo.

La RAG è una tecnica che combina il Prompt-Engineering con la Retrieval delle informazioni da fonti esterne per guidare la generazione di testo di un LLM. Introdotta da ricercatori di Meta (precedentemente Facebook), la RAG offre una struttura in grado di migliorare significativamente la precisione, la rilevanza e la ricchezza delle risposte generate dai modelli. Un lavoro che ha introdotto il concetto di RAG può essere trovato da quanto scritto da Lewis et al. [15]

5.1 Importanza nel Contesto Attuale dei Modelli di Linguaggio

In un'era dominata da un'enorme quantità di informazioni digitali, la capacità di generare risposte precise e contestualmente rilevanti è più critica che mai. I Large Language Models, pur essendo sempre più capaci, spesso lottano con la generazione di contenuti che richiedono conoscenze aggiornate o altamente specifiche. La RAG affronta questa sfida integrando direttamente le informazioni esterne nel processo di generazione, permettendo ai modelli di attingere da un vasto insieme di dati forniti in tempo reale.

5.2 Come funziona la RAG?

Essenzialmente, la RAG accoppia meccanismi di recupero delle informazioni con modelli di generazione di testo. Il componente di recupero delle informazioni aiuta ad estrarre informazioni contestuali rilevanti da un database, e il modello di generazione di testo utilizza questo contesto aggiunto per produrre una risposta più precisa e "informata". Ecco come funziona:

1. **Vector Database:** Implementare la RAG comporta il caricamento di un'insieme di dati, la creazione di vettori da esso e la memorizzazione in un Vector Database. Ciò consente di cercare e recuperare informazioni in modo efficiente, basandosi sulla similarità semantica tra la query dell'utente e i contenuti del database.

2. **Query dell'utente:** La RAG inizia con la ricezione di una query dell'utente nel prompt di un chatbot o un box, questa query rappresenta la domanda o l'affermazione che l'utente desidera esplorare o a cui desidera ricevere una risposta.
3. **Retrieval:** Una volta ricevuta la query dell'utente, la chain di recupero o retrieval esamina il database vettoriale per identificare frammenti di informazioni che presentano somiglianze semantiche(usando ad esempio la similarità del coseno) con la query. Questi pezzi rilevanti vengono quindi utilizzati per fornire ulteriore contesto al LLM, consentendogli di generare una risposta più precisa e consapevole del contesto.
4. **Concatenazione:** I documenti o dati recuperati vengono concatenati con la query originale in un prompt che fornisce ulteriore contesto per la generazione delle risposte.
5. **Generazione di testo:** Infine, il prompt arricchito viene mandato al LLM, che genera una risposta finale. Questo testo non solo risponde alla query iniziale ma lo fa con un livello di precisione, dettaglio e rilevanza notevolmente migliorato grazie al contesto fornito dai dati recuperati.

L'articolo "Retrieval-Augmented Generation for Large Language Models: A Survey" [7] presenta una visione completa dei paradigmi di Retrieval-Augmented Generation (RAG), che utilizza conoscenze da database esterni per migliorare l'accuratezza e la credibilità dei modelli di linguaggio, specialmente per compiti specifici. Esamina l'evoluzione della RAG, dalle versioni Naive alle più avanzate e modulari, analizzando le tecnologie all'avanguardia per il recupero, la generazione e l'arricchimento delle informazioni. L'articolo discute anche metriche e benchmark per valutare i modelli RAG e suggerisce future direzioni di ricerca, come l'espansione delle multimodalità e il miglioramento dell'infrastruttura RAG, inoltre cerca di ridurre il fenomeno dell'hallucination.

5.3 Casi d'uso RAG

La RAG trova applicazione in una vasta gamma di scenari, ove la generazione di risposte accurate e contestualizzate è cruciale. Ecco alcuni esempi significativi:

- **Chatbots e Assistenti Virtuali:** Integrando la RAG, questi sistemi possono fornire risposte più precise e informative, attingendo da un'ampia base di conoscenze esterne.
- **Ricerca e Sintesi di Informazioni:** La RAG può essere impiegata per riassumere articoli, rapporti di ricerca o documentazione tecnica, fornendo sintesi contestualizzate e dettagliate basate su fonti affidabili.
- **Applicazioni in Medicina e Scienze:** Nel campo medico, la RAG può aiutare a fornire diagnosi assistite o riassunti di trattamenti basati sull'ultimo stato della ricerca, migliorando l'accuratezza delle informazioni fornite ai professionisti e ai pazienti.

La RAG è inoltre utile quando l'applicazione necessita di informazioni e documenti aggiornati che non facevano parte del set di addestramento del LLM. Alcuni esempi potrebbero includere database di notizie o applicazioni che cercano ricerche mediche associate a nuovi trattamenti.

Il prompt-engineering o un LLM semplice non possono gestire questi casi a causa della finestra di contesto limitata del LLM. Attualmente, per la maggior parte dei casi d'uso, non è possibile inserire l'intero set di documenti nel prompt del LLM.

L'articolo 'Development and Testing of Retrieval Augmented Generation in Large Language Models - A Case Study Report' [11] esplora l'uso dei Large Language Models (LLMs) insieme alla Retrieval Augmented Generation (RAG) per migliorare le applicazioni mediche, concentrandosi sulla medicina preoperatoria. Viene valutata la precisione e la sicurezza delle risposte generate da LLM-RAG rispetto a quelle umane e LLM standard, usando 35 linee guida preoperatorie per la valutazione.

Il metodo RAG migliora significativamente l'accuratezza dei modelli LLM, rendendoli più veloci e affidabili rispetto alle risposte umane per fornire istruzioni preoperatorie, dimostrando non inferiorità. Questo approccio offre vantaggi come l'aggiornabilità, la scalabilità e una base di conoscenza solida, cruciali per l'implementazione sanitaria degli LLM.

5.4 Vantaggi della RAG

La Retrieval Augmented Generation (RAG) offre numerosi vantaggi che la rendono una tecnologia preziosa per l'evoluzione dei modelli di linguaggio e l'ampliamento delle loro applicazioni. Questi benefici sono particolarmente evidenti in confronto alle tecniche di generazione basate esclusivamente sull'ingegneria delle istruzioni o sui modelli preaddestrati senza accesso a dati esterni.

- **Minimizza le Allucinazioni** I modelli di linguaggio, specialmente quelli di grandi dimensioni, sono talvolta propensi a generare risposte non basate sul contesto della risposta e su informazioni reali, fenomeno noto come "allucinazioni". La RAG riduce significativamente questo rischio integrando informazioni provenienti da fonti esterne verificate o inserite dall'utente, assicurando che le risposte siano ancorate a dati reali e pertinenti.
- **Si Adatta Facilmente a Nuovi Dati** In un mondo in rapida evoluzione, la capacità di aggiornarsi con nuove informazioni è cruciale. La RAG permette ai modelli di linguaggio di rimanere aggiornati integrando le ultime pubblicazioni e dati, senza la necessità di un costoso e continuo processo di re-addestramento.
- **Interpretabile** La tracciabilità delle informazioni è un altro vantaggio significativo della RAG. Essendo in grado di identificare le fonti delle informazioni utilizzate per

generare una risposta, gli utenti e i sviluppatori possono verificare l'accuratezza e l'affidabilità delle risposte fornite dal sistema.

- **Economico** Dal punto di vista dell'efficienza delle risorse, la RAG offre un modo per migliorare le prestazioni dei modelli di linguaggio senza la necessità di vasti set di dati etichettati o di complesse operazioni di addestramento. Questo può tradursi in significativi risparmi di tempo e costi.

[17] All'interno di questo articolo viene comparata la RAG con la DPR(Dense Passage Retrieval) evidenziando l'ottenimento di risultati ancora migliori, su question-answering.

5.5 Limitazioni della RAG

La RAG è progettata per potenziare le capacità di recupero delle informazioni di un LLM estraendo il contesto da documenti esterni. Tuttavia, in determinati casi d'uso, il contesto aggiuntivo potrebbe non essere sufficiente. Se un LLM preaddestrato fatica a riassumere dati finanziari o a interpretare tabelle, grafi o immagini e a trarre spunti dalla documentazione medica di un paziente, potrebbe un singolo documento non bastare, ma al contrario produrre ulteriore rumore. In tali casi, è più probabile che il Fine-Tuning possa produrre la risposta desiderata.

Capitolo 6

Scelta del giusto approccio per l'applicazione

Dopo aver esaminato i quattro approcci all'adattamento dei Large Language Model (LLM), confrontiamoli su tre metriche importanti: complessità, costo e accuratezza.

6.1 Costo

Nel valutare il costo di un approccio, ha senso prendere in considerazione il costo della sua implementazione iniziale, insieme al costo di mantenimento della soluzione. Dato ciò, confrontiamo i costi dei quattro approcci.

6.1.1 Prompt-Engineering

L'ingegneria delle istruzioni ha il costo più basso tra i quattro approcci. Si riduce alla scrittura e al test di prompt per trovare quelli che danno buoni risultati quando alimentati al LLM preaddestrato. Potrebbe anche implicare l'aggiornamento dei prompt se il modello preaddestrato viene periodicamente aggiornato o sostituito, ad esempio con un modello commerciale come il GPT-4 di OpenAI.

6.1.2 RAG

Il costo di implementazione di RAG potrebbe essere superiore a quello del prompt-engineering. Ciò è dovuto alla necessità di diversi componenti: modello di estrazioni dati, vector database, retriever e LLM pre-trained.

6.1.3 PEFT

Il costo di PEFT tende ad essere superiore a quello della RAG. Ciò è dovuto al fatto che il fine-tuning, anche efficiente, richiede una considerevole potenza di calcolo, tempo ed esperienza in machine learning. Inoltre, per mantenere questo approccio, sarà necessario eseguire il fine-tuning periodicamente per incorporare nuovi dati rilevanti nel modello.

6.1.4 Full fine-tuning

Questo metodo è significativamente più costoso rispetto a PEFT, dato che richiede ancora più potenza di calcolo e tempo.

6.2 Complessità dell'implementazione

Dall'approccio relativamente semplice del prompt-engineering alle configurazioni più intricate della RAG e metodi di fine-tuning, la complessità può variare notevolmente. Ecco una rapida panoramica di cosa comporta ciascun metodo.

6.2.1 Prompt-Engineering

Questo metodo ha una complessità di implementazione relativamente bassa. Richiede poca o nessuna programmazione. Per redigere un buon prompt e condurre esperimenti, un ingegnere delle istruzioni ha bisogno di buone competenze linguistiche, competenze di dominio e familiarità con gli approcci di apprendimento pochi tiri.

6.2.2 RAG

Questo approccio ha una complessità di implementazione più elevata rispetto all'ingegneria delle istruzioni. Per implementare questa soluzione, sono necessarie competenze di codifica e di architettura. A seconda dei componenti RAG scelti, la complessità potrebbe essere molto elevata.

6.2.3 PEFT e Full fine-tuning

Questi approcci sono i più complessi da implementare. Richiedono una profonda comprensione di deep learning e NLP, nonché competenze in scienze dei dati per modificare i pesi del modello tramite script di sintonizzazione. È necessario considerare anche fattori come i dati di addestramento, il tasso di apprendimento, la funzione di perdita, ecc.

6.3 Accuratezza

Valutare l'accuratezza dei diversi approcci per l'adattamento dei LLM può essere complesso, specialmente perché l'accuratezza spesso dipende da una combinazione di metriche distinte. La significatività di queste metriche può variare in base al caso d'uso specifico. Alcune applicazioni potrebbero dare priorità al gergo specifico del dominio, mentre altre potrebbero privilegiare la capacità di risalire alla fonte della risposta del modello. Per trovare l'approccio più accurato per le tue esigenze, è imperativo identificare le metriche di accuratezza pertinenti per la tua applicazione e confrontare le metodologie rispetto a tali criteri specifici.

6.3.1 Terminologia specifica del dominio

Il fine-tuning può essere utilizzato per impartire in modo efficace ai LLM la terminologia specifica del dominio. Mentre RAG è competente nel recupero dei dati, potrebbe non catturare modelli, vocabolario e sfumature specifiche del dominio tanto bene quanto un modello sintonizzato. Per compiti che cercano una forte affinità di dominio, il fine-tuning è la strada da percorrere.

6.4 Approccio Implementato

L'approccio scelto per lo sviluppo finale del chatbot BandiPugliaBot ha integrato l'impiego del modello GPT-3.5 Turbo, opportunamente fine-tuned con un insieme selezionato di bandi della Regione Puglia, abbinato all'impiego della tecnologia Retrieval Augmented Generation (RAG). Questa combinazione ha significativamente migliorato la precisione delle risposte fornite dal sistema. La decisione di adottare questa strategia è emersa dalla valutazione comparativa delle varie tecniche durante lo sviluppo del progetto. Tale scelta ha permesso non solo di ottimizzare le performance del sistema in termini di accuratezza e rilevanza delle informazioni fornite, ma anche di mantenere l'implementazione all'interno dei limiti di budget prefissati, dopo aver confrontato i risultati delle diverse tecniche, dimostrando l'efficacia dell'integrazione di queste avanzate tecnologie di intelligenza artificiale nel rispondere alle esigenze informative specifiche degli utenti interessati ai bandi della Regione Puglia.

Capitolo 7

Inizio del progetto

Il progetto si è diviso in notebook per la parte di machine learning e codice chatbot per la UX. Nel notebook "01_Dataset_Building", l'attenzione è rivolta alla preparazione del dataset destinato al fine-tuning del modello davinci-002.

7.1 Raccolta dei Documenti

Inizialmente, si effettua la raccolta di tutti i documenti in formato PDF relativi ai bandi attualmente in corso, reperibili sul sito sistema.puglia.it. Questi documenti vengono successivamente scaricati e archiviati nella cartella "Documenti" del repository sul drive.

7.2 Estrazione del Testo dai PDF

Per gestire l'estrazione del testo dai PDF, viene impiegata la libreria PyPDF2, consentendo così di creare una lista di 26 documenti. Successivamente, si procede alla creazione di un dizionario, utilizzando il titolo di ciascun documento come chiave e associandovi il testo estratto come valore.

7.2.1 Libreria PyPDF2

PyPDF2 è una libreria Python open-source che semplifica il processo di lavoro con i file PDF. Fornisce una vasta gamma di funzionalità, tra cui la lettura e la scrittura di file PDF, l'estrazione di testo e metadati, la divisione e l'unione di documenti, l'aggiunta di filigrane, la crittografia e la decrittografia di file e altro ancora.

Per gestire l'estrazione del testo dai PDF nel progetto, è stato utilizzato PyPDF2, che ha permesso di creare una lista di 26 documenti. Successivamente, si è proceduto alla creazione di un dizionario, utilizzando il titolo di ciascun documento come chiave e associandovi il testo estratto come valore.

```
1 import os
2 import PyPDF2
3
4 def carica_pdf(cartella):
```

```
5
6 pdf_list = []
7 pdf_readers = []
8 for file in os.listdir(cartella):
9     if file.endswith(".pdf"):
10         pdf_list.append(file)
11         pdf_reader = PyPDF2.PdfReader(os.path.join(cartella, file))
12         pdf_readers.append(pdf_reader.pages)
13
14 return pdf_readers, pdf_list
15
16
17 if __name__ == "__main__":
18     cartella = "./Documenti"
19
20     pdf_readers, pdf_list = carica_pdf(cartella)
21
22     documenti_dict = {}
23
24     for pdf, reader in zip(pdf_list, pdf_readers):
25         testo = ""
26         for i, page in enumerate(reader):
27             testo += page.extract_text()
28         documenti_dict[pdf] = testo
```

SNIPPET 7.1: Caricamento PDF Bandi Sistema Puglia

7.3 Organizzazione del Dataset

L'enumerazione dei documenti viene eseguita per garantire un'organizzazione chiara e ordinata del dataset. Nel corso di questa fase, si rileva che alcuni documenti superano il limite massimo di 4096 caratteri. Ciò è stato rilevato usando una funzione di tiktoken per contare tutti i numeri di token all'interno di un documento. Il numero di token nei diversi modelli non è direttamente proporzionale al numero di caratteri in quanto la codifica dei token è diversa per ogni modello utilizzato. [20]

```
1 import tiktoken
2 def num_tokens_from_string(string: str, encoding_name: str) -> int:
3     """Returns the number of tokens in a text string."""
4     encoding = tiktoken.get_encoding(encoding_name)
5     num_tokens = len(encoding.encode(string))
6     return num_tokens
```

SNIPPET 7.2: Funzione per contare i token decodificati

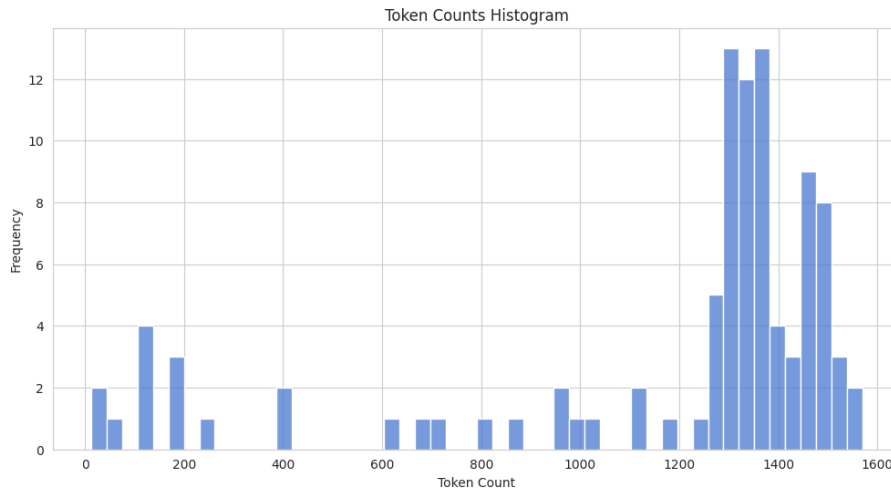


FIGURA 7.1: Istogramma che raffigura la frequenza del numero della lunghezza di token nei chunk

7.4 Gestione dei Documenti di Grandi Dimensioni

Per affrontare questa sfida, si adotta la libreria LangChain, impiegando la funzione di suddivisione del testo denominata 'RecursiveCharacterTextSplitter', impostando un parametro di 'chunk_size' pari a 3796 caratteri, in modo da mantenere complessivamente tra prompt e completion un totale di circa 4000 caratteri. Questa scelta è cruciale, poiché il numero di caratteri influisce direttamente sulle risposte del modello fine-tunato. Se gli esempi presentano una lunghezza uniforme, il modello, una volta addestrato, risponderà coerentemente con risposte brevi o lunghe a seconda del pattern di lunghezza presente negli esempi di addestramento.

Quindi si è cercato di mantenere una lunghezza di caratteri pari a un massimo di 4000, incluso il prompt (la domanda) e la completion (l'esempio di risposta), in quanto dovrebbero equivalere circa a 1024 token, sapendo che un prompt di Chat-GPT riesce a produrre fino ad un massimo di 4096 token (incluso il prompt), possiamo includere un 'storico' di 3 esempi di risposte generate come prompt, in modo da migliorare la qualità della risposta in un massimo di 1024 token.

Ciò viene fatto anche perché:

- **Gestione della memoria:** I modelli di linguaggio, come GPT-3 di OpenAI, hanno un limite massimo alla lunghezza del testo che possono gestire in una singola iterazione, noto come "lunghezza massima della sequenza". Per GPT-3, questo limite è di 4096 token. Dividendo i documenti in segmenti più piccoli, siamo in grado di adattarci a questo vincolo.

- **Attenzione del modello:** I modelli Transformer, come GPT-3, utilizzano un meccanismo di attenzione che permette al modello di “guardare indietro” ai token precedenti nella sequenza quando genera un nuovo token. Tuttavia, l’attenzione diminuisce con la distanza, quindi i token all’inizio di un documento lungo potrebbero avere meno impatto sui token successivi. Dividendo un documento in segmenti più piccoli, ogni token ha più probabilità di influenzare la generazione di token successivi.
- **Efficienza computazionale:** Il fine-tuning e la RAG richiedono una grande quantità di calcolo. Dividendo i documenti in segmenti più piccoli, possiamo distribuire il calcolo su più unità di elaborazione, accelerando il processo di addestramento.
- **Precisione del modello:** Infine, dividendo i documenti in segmenti più piccoli, possiamo ottenere una rappresentazione più precisa del documento. Questo perché i modelli di linguaggio sono addestrati su segmenti di testo di dimensioni simili durante la fase di pre-addestramento. Pertanto, lavorare con segmenti di dimensioni simili durante il fine-tuning o la RAG può portare a migliori prestazioni del modello.

7.4.1 Libreria LangChain

LangChain [13] è un framework progettato per semplificare la creazione di applicazioni che utilizzano modelli di linguaggio di grandi dimensioni. È molto più di un semplice framework; è un ecosistema completo composto da diversi moduli.

Le librerie LangChain, disponibili sia in Python che in JavaScript, offrono interfacce e integrazioni per vari componenti. Forniscono una runtime di base per combinare questi componenti in chain e agenti, insieme a implementazioni pronte per l’uso.

LangChain è stato progettato per supportare la messa in produzione dei prototipi, senza modifiche al codice, dalla chain più semplice “prompt + LLM” alle chain più complesse.

7.4.2 Text Splitter

Dunque il testo è stato suddiviso come si vede nel seguente codice:

```
1 r_splitter = RecursiveCharacterTextSplitter(  
2     chunk_size=3796,  
3     chunk_overlap=30,  
4     separators=["\n\n", "\n", "(?<=\. )", " ", ""]  
5 )  
6 for doc in documenti_dict.keys():  
7     documenti_dict[doc] = r_splitter.split_text(documenti_dict[doc])
```

SNIPPET 7.3: Funzione per splittare i documenti

[14] Dove `chunk_overlap` che si riferisce a una sovrapposizione o sovrapposibilità tra i "chunk" o blocchi di dati. Mentre Il parametro `separators` riveste un ruolo chiave nella funzione `RecursiveCharacterTextSplitter` della libreria `LangChain`. Questo parametro consente di specificare un insieme di separatori che la funzione utilizzerà per suddividere il testo in porzioni più piccole, comunemente chiamate "chunk".

Quando si utilizzano gli separatori, la procedura avviene seguendo un ordine, partendo dal primo elemento dell'insieme. La funzione cerca di suddividere il testo utilizzando il primo separatore. Se la dimensione del blocco risultante è ancora eccessiva, la funzione passa al separatore successivo nella lista.

Per esempio, considera un insieme di separatori come

```
{["\n\n", "\n", " ", ""]}
```

. In questo caso, la funzione cercherà di suddividere il testo usando

```
{\n\n}
```

come primo separatore. Se il blocco risultante è ancora troppo grande, passerà a

```
{\n}
```

e così via.

Questa strategia di divisione del testo è progettata per mantenere insieme il più a lungo possibile le porzioni di testo semanticamente correlate, come paragrafi e frasi.

È rilevante notare che la dimensione del blocco è misurata in termini di numero di caratteri. Inoltre, è possibile personalizzare la funzione `RecursiveCharacterTextSplitter` utilizzando separatori arbitrari, forniti come parametro separato.

La struttura originaria dei documenti prima della suddivisione era la seguente:

```
1 Il documento 'Scheda MicroPrestito della Regione Puglia - edizione
  2021.pdf' ha una lista di 6289 caratteri. che corrispondono ad un
  numero di token 2644
2 Il documento 'Scheda NIDI - Nuove iniziative d'impresa_ Strumento di
  ingegneria finanziaria.pdf' ha una lista di 16305 caratteri. che
  corrispondono ad un numero di token 6771
3 Il documento 'Aiuti agli Investimenti delle Piccole e Medie Imprese (Titolo
  II Capo 3 Reg_ Regionale 17_2014)_br ___br __.pdf' ha una lista di
  25127 caratteri. che corrispondono ad un numero di token 10847
4 Il documento 'Aiuti Agli Investimenti delle PMI nel Settore Turistico -
  Alberghiero (Titolo II capo 6 Reg_ Reg_ 17_2014) _br ___br __.pdf' ha
  una lista di 26391 caratteri. che corrispondono ad un numero di token
  11486
```

Mentre dopo esser stati splittati ogni chunk ad esempio è formato così :

```

1   Il documento 'Scheda MicroPrestito della Regione Puglia - edizione
    2021.pdf' ha una lista di 3674 caratteri. che corrispondono ad un
    numero di token 1450
2   Il documento 'Scheda NIDI - Nuove iniziative d'impresa_ Strumento di
    ingegneria finanziaria.pdf' ha una lista di 3723 caratteri. che
    corrispondono ad un numero di token 1431
3   Il documento 'Aiuti agli Investimenti delle Piccole e Medie Imprese (Titolo
    II Capo 3 Reg_ Regionale 17_2014)_br ___br __.pdf' ha una lista di
    3689 caratteri. che corrispondono ad un numero di token 1465
4   Il documento 'Aiuti Agli Investimenti delle PMI nel Settore Turistico -
    Alberghiero (Titolo II capo 6 Reg_ Reg_ 17_2014) _br ___br __.pdf' ha
    una lista di 3715 caratteri. che corrispondono ad un numero di token
    1498

```

7.4.3 Generazione Automatica del Dataset con l'API di OpenAI

Il problema associato al fine-tuning risiede nella necessità di costruire manualmente un dataset composto da esempi, ciascuno contenente una domanda e la relativa risposta. Questa procedura può risultare dispendiosa in termini di tempo, specialmente quando si gestiscono numerosi esempi, come nel nostro caso con 100 di essi.

Per superare questa sfida in modo automatizzato, è stato sfruttato l'API di OpenAI utilizzando il modello "text-davinci-003", noto per la sua capacità di generare complementi del contesto. Al fine di creare i prompt, ossia le domande, è stato adottato il seguente approccio:

```

1   import openai
2   import json
3   import time
4
5   openai.api_key = 'sk-VByZ1sFayj9tFERV4a3uT3B1bkFJ9sQwMSJ1Arru3LvNH5N'
6
7   data_to_save = []
8
9   for key, value in documenti_dict.items():
10      if isinstance(value, list):
11         for text in value:
12            prompt_text = f"Genera una domanda basata sul seguente testo:\n
13            {text}\nDomanda:"
14
15            num_tokens_input = len(prompt_text) / 4
16            print(f"Numero approssimativo di token dell'input: {
17            num_tokens_input}")
18
19            response = None
20            while response is None:
21               try:
22                  response = openai.Completion.create(
23                     engine="text-davinci-003",

```



```

22         prompt=prompt_text,
23         max_tokens=150,
24         n=1,
25         stop=None,
26         temperature=0.7
27     )
28     except openai.error.RateLimitError as e:
29         print(f"Rate limit reached. Waiting for 20 seconds...")
30         time.sleep(20)
31
32     question = response.choices[0].text.strip()
33     print(question)
34
35     data = {
36         "prompt": question + f" in riferimento al documento: '{key
37     },",
38         "completion": text
39     }
40     data_to_save.append(data)
41
42     time.sleep(2)
43
44 with open('domande_e_testi.json', 'w') as json_file:
45     json.dump(data_to_save, json_file, indent=4)

```

La risposta generata è stata successivamente inclusa in un dizionario denominato “data”, contenente sia il prompt generato che la completion, ovvero il testo su cui è stata generata la domanda, con riferimento al nome del documento a cui appartiene. Il costo totale della generazione delle domande utilizzando il modello “text-davinci-003” è di \$3.14. Pur essendo un modello più costoso rispetto a “gpt-3.5-turbo”, offre una maggiore precisione nel testo generato, con un costo di \$0.0200 per ogni 1.000 token rispetto ai \$0.0015 / 1.000 token di “gpt-3.5-turbo-0613”. Di seguito sono presenti alcuni esempi di prompt generati:

- “Quali sono i codici ATECO ammissibili per le imprese confiscate? in riferimento al documento: ‘Scheda NIDI - Nuove iniziative d’impresa_ Strumento di ingegneria finanziaria.pdf’ ”
- “Qual è l’agevolazione prevista per gli investimenti compresi tra 50.000 e 100.000 euro per le iniziative in continuità con imprese pre-esistenti? in riferimento al documento: ‘Scheda NIDI - Nuove iniziative d’impresa_ Strumento di ingegneria finanziaria.pdf’ ”
- “Quali spese non sono ammissibili alle agevolazioni previste dal Fondo europeo di sviluppo regionale per il periodo di programmazione 2014/2020? in riferimento al documento: ‘Aiuti agli Investimenti delle Piccole e Medie Imprese (Titolo II Capo 3 Reg_ Regionale 17_2014)_br __br __.pdf’ ”

Successivamente, è stato seguito un procedimento analogo migliorando la completion per ottenere una struttura simile a una risposta anziché a una porzione di testo del PDF. Di seguito è riportato il codice per eseguire questo miglioramento:

```

1  import openai
2  import json
3  import time
4
5  for data in data_to_save:
6      prompt_text = f"Migliora il seguente testo:\n{data['completion']}\n"
7      rimanendo fedele alla domanda {data['prompt']}"
8
9      response = None
10     while response is None:
11         try:
12             response = openai.Completion.create(
13                 engine="text-davinci-003",
14                 prompt=prompt_text,
15                 max_tokens=1024,
16                 n=1,
17                 stop=None,
18                 temperature=0.7
19             )
20         except openai.error.RateLimitError as e:
21             print(openai.error.RateLimitError)
22             print(f"Rate limit reached. Waiting for 20 seconds...")
23             time.sleep(20)
24
25     improved_completion = response.choices[0].text.strip()
26     print(improved_completion)
27     # Estrai il numero di token utilizzati dalla risposta
28     number_of_tokens_used = response['usage']['total_tokens']
29     print(f"Numero di token utilizzati: {number_of_tokens_used}")
30     data['completion'] = improved_completion
31
32     time.sleep(2)

```

Per eseguire questo lavoro appunto è stata utilizzata la tecnica del Prompt-Engineering, discussa nei capitoli precedenti, applicata su text-davinci-003 per appunto ricercare il miglior prompt che eseguisse al meglio e più efficacemente il completamento del Dataset per fine-tunare il modello.

7.5 Fine-Tuning: Procedura e Risultati Metrici

Nei notebook dedicati all'avvio del Fine-Tuning, sono stati inizialmente stimati i costi associati ai modelli OpenAI, in particolare a davinci-002 e gpt-3.5-turbo, sui quali verrà successivamente eseguito il Fine-Tuning. I costi sono suddivisi tra il costo del Fine-Tuning in sé e il costo dell'utilizzo del modello Fine-Tuned.

Complessivamente, nel corso del progetto, il Fine-Tuning è stato eseguito per tre iterazioni. La prima iterazione ha coinvolto il Fine-Tuning del dataset comprendente i prompt generati e le completion migliorate, con un numero di epoche pari a 4 (solitamente utilizzato per impostazione predefinita). Nel secondo processo di Fine-Tuning, il dataset con i prompt generati è stato utilizzato, ma in questo caso la completion non è stata migliorata da text-davinci-003.

Il numero di epoche è stato aumentato a 6 al fine di ottenere una risposta più simile al modello addestrato su un dataset migliorato. Nel terzo processo, il dataset è stato trasformato nel formato JSONL e sottoposto a Fine-Tuning utilizzando la struttura conversazionale di "gpt-3.5-turbo". Questo modello è stato necessario per la fase di RetrievalQA di LangChain per la RAG. Il totale dei token nell'intero dataset si aggira intorno ai 120,000.

Questa è la tabella che include il costo di Fine-Tuning stimato per le 3 iterazioni dopo aver stimato il costo con il seguente codice:

```

1  def costo_utilizzo(texts):
2      import tiktoken
3      enc = tiktoken.encoding_for_model("davinci-002")
4      total_tokens=(len(enc.encode(texts)))
5      print(f'Totale token:{total_tokens}')
6      costo_input = (total_tokens/1000* 0.0120)
7      costo_output = (total_tokens/1000* 0.0120)
8      costo_totale = costo_input + costo_output
9      print(f'Costo utilizzo Davinci-002 in $ (OpenAI): {costo_totale:.6f}')
10     return costo_totale
11 costo_utilizzo_davinci = costo_utilizzo(json_str)

```

Nome Modello	Costo Training Stimato	Costo Utilizzo del Modello stimato	Costo Training Effettivo	Costo Totale
davinci-002 con 4 epoche	\$1.665	\$1.665	\$1.51	\$3.165
davinci-002 con 6 epoche ma completion non migliorata	\$5.412	\$1.665	\$3.6	\$5.265
gpt-3.5-turbo	\$2.220	\$0.665	\$1.85	\$2.515

TABELLA 7.1: Tabella relativa alla stima dei costi di Fine-Tuning

Una volta completata la fase di stima dei costi, l'effettivo processo di Fine-Tuning implica il caricamento del file contenente il dataset attraverso la funzione `openai.File.create`. Successivamente, l'ID del file viene utilizzato in un'altra funzione di OpenAI per avviare immediatamente il Fine-Tuning, come eseguito nel codice:

```

1 import os
2 import openai
3
4 openai.api_key = os.getenv("OPENAI_API_KEY")
5
6 c = openai.File.create(
7     file=open("./Tesi/fine_tune_improved_list.json", "rb"),
8     purpose='fine-tune'
9 )
10
11
12 r = openai.FineTuningJob.create(training_file="file-3
    KvLZc6kKFMdNrn8Juuh2LvU", model="davinci-002", suffix="test-Signorile",
    hyperparameters={"n_epochs":4})
13
14 jobname = "ftjob-XuAnm3MliLFiyjxoJFN0wyxC"
15
16 r = openai.FineTuningJob.retrieve(jobname)
17 e = openai.FineTuningJob.list_events(id=jobname, limit=10)

```

Dove "training_file" rappresenta l'ID del file, "model" indica il modello da sottoporre a Fine-Tuning, "suffix" consente di specificare un suffisso per il nome del modello generato e "hyperparameters" permette di definire gli iperparametri per il processo di addestramento. La funzione di Fine-Tuning restituirà un "job name" come, ad esempio, "ftjob-XuAnm3MliLFiyjxoJFN0wyxC". Attraverso la funzione `openai.FineTuningJob.retrieve(jobname)`, è possibile ottenere lo stato del Fine-Tuning, visualizzando tutti gli iperparametri utilizzati nell'addestramento. Inoltre, con `openai.FineTuningJob.list_events(id=jobname, limit=10)`, è possibile visualizzare gli eventi, inclusi gli step e le relative training loss del processo di addestramento

Queste sono le metriche di tutti le esecuzioni di Fine-Tuning per i modelli:

Modello	Epochs	Trained tokens	Training Loss
Davinci-002	4	251 120	1.1438
Davinci-002	6	613 428	0.7629
Gpt-3.5-turbo	4	251 120	1.0772

TABELLA 7.2: Tabella relativa alle metriche di addestramento del fine-tuning di OpenAI

Come possiamo notare dalla tabella eseguendo il Fine-Tuning di Davinci-002 con 4 epoche produce una Training loss molto maggiore rispetto ad addestrarlo con 6 epoche, il che si traduce in un potenziale miglioramento del modello o una maggiore efficienza nell'addestramento con l'aumentare delle epoche, aumentando tuttavia il costo del Fine-Tuning dovuto al numero maggiore di tokens allenati. Anche GPT-3.5-Turbo a 4 epoche evidenzia una leggera previsione più accurata sul modello sempre con 4 epoche.

In sintesi, la tabella fornisce una visione chiara su quanto bene ciascun modello ha appreso durante il Fine-Tuning.

7.6 Evaluations

Le tabelle che seguono presentano una serie di valutazioni effettuate da tre esperti distinti. Ogni tabella corrisponde a un'analisi dettagliata delle prestazioni dei modelli in esame, offrendo insight critici e osservazioni approfondite sulle risposte generate dai modelli ai vari prompt.

7.6.1 Davinci-002 fine-tuned con 4 epoche

si procede con la valutazione dei modelli basata sulle risposte fornite ai prompt su cui sono stati addestrati. Inizialmente, viene eseguita una prova con il modello fine-tuned utilizzando davinci-002 per un totale di 4 epoche. Questa valutazione viene effettuata attraverso l'impiego della funzione `openai.Completion.create()`.

Al momento dell'implementazione, non essendo ancora pienamente a conoscenza del funzionamento della funzione, sono stati impostati alcuni parametri, tra cui una `temperature` pari a 0.5.

La `temperature` è un parametro che influisce sulla creatività della risposta generata dal modello. Un valore di 0.5 suggerisce un livello moderato di creatività, in cui la risposta è più prevedibile. Valori più vicini a 0 tendono a produrre completamenti più deterministici.

A `temperature = 1`, dopo una valutazione degli esperti, si può osservare che il modello produce risposte estremamente creative, spesso prive di senso. Nel caso della temperatura 0.7, la prima risposta mostra frasi ripetitive in forma di loop. La risposta più equilibrata e coerente si ottiene a temperatura 0.5. È da notare che, occasionalmente, le risposte a questa temperatura possono presentare fenomeni di "hallucination". In tal caso, rigenerare le risposte può contribuire a ottenere una formulazione più coerente.

Successivamente, sono state generate da text-davinci-003 dieci domande simili a: "Quali sono le modalità per ottenere l'agevolazione prevista dall'Avviso? in riferimento al documento: 'Aiuti agli Investimenti delle Piccole e Medie Imprese (Titolo II Capo 3 Reg. Regionale 17_2014)_br __br __.pdf'". Una volta ottenute le 10 domande, sono state sottoposte al modello per l'analisi delle risposte.

Prompt

"Qual è l'agevolazione prevista per gli investimenti compresi tra 50.000 e 100.000 euro per le iniziative in continuità con imprese pre-esistenti? in riferimento al documento: 'Scheda NIDI - Nuove iniziative d'impresa_ Strumento di ingegneria finanziaria.pdf'"

Completion Originale

"Per le iniziative in continuità con imprese pre-esistenti quali: passaggio generazionale, rilevamento di impresa in crisi... L'agevolazione per investimenti compresi tra 50.000 e 100.000 euro è pari al 90%: un contributo a fondo perduto pari al 50% del totale degli investimenti agevolati e un prestito rimborsabile pari al 50%..."

Risposta Generata con Temperatura 0.7

"Sono previste agevolazioni fino al 50% dell'investimento... Il finanziamento in conto capitale, a tasso agevolato, non supera il 25% dell'importo complessivo del progetto..."

Risposta Generata con Temperatura 0.5

"Per gli investimenti compresi tra 50.000 e 100.000 euro, l'agevolazione è pari al 25% del finanziamento concesso... La misura agevolativa massima è pari a 60.000 euro per gli investimenti fino a 250.000 euro..."

Risposta Generata con Temperatura 1

"Per riferimento alla domanda 'Qual è l'agevolazione prevista...' il Fondo europeo per lo Sviluppo Regionale (FESR), il Fondo europeo agricolo per lo Sviluppo Rurale (FESR) e il POR Puglia 2014-2020 attribuiscono un'agevolazione pari al 30% dell'investimento..."

TABELLA 7.3: Tabella che rappresenta il confronto tra le risposte date dal modello davinci-002 con 4 epoche

Domanda	Risposta
Quali sono i requisiti necessari per accedere all'agevolazione prevista nell'Avviso?	Per accedere all'agevolazione prevista nell'Avviso i requisiti necessari sono: età massima dell'impresa al momento della presentazione della domanda di agevolazione pari a 10 anni per le nuove imprese e/o quelle nate da ristrutturazioni o trasformazioni societarie...
Qual è la durata dell'agevolazione una volta ottenuta?	Le agevolazioni sono erogate in conto interessi con un tasso fisso pari al 0.25% annuo. Il rimborso delle stesse avviene in un'unica soluzione alla scadenza del finanziamento (anno 5)...
C'è una scadenza per presentare la domanda di ...	Sì, la scadenza è il 30 giugno 2016. Per prese...
Quali sono i criteri di valutazione utilizzati...	I criteri di valutazione utilizzati per selezi...
Qual è l'importo massimo dell'agevolazione che...	Aiuti agli Investimenti delle Piccole e Medie ...
Quali sono le tipologie di investimenti ammiss...	Si possono ottenere agevolazioni per investime...

TABELLA 7.4: Tabella relativa a Domande e Risposte generate dal modello

Al fine di evitare la presenza ricorrente di frasi ripetute in forma di loop nelle risposte, è stato introdotto un parametro denominato `frequency_penalty` nell'ambito della Completion. Tale parametro agisce come un metodo di controllo sulla ripetitività, assegnando un peso a ciascun token generato. Un intervallo ragionevole per il valore di `frequency_penalty` va da 0.1 a 1. Un incremento oltre 2 potrebbe compromettere significativamente la qualità della risposta.

L'analisi delle risposte evidenzia la presenza di completamenti molto estesi, superando frequentemente i 1000 token. Questa caratteristica è il risultato diretto del processo di addestramento, durante il quale il modello ha appreso a rispondere alle domande seguendo la struttura degli esempi forniti. Va sottolineato che tali esempi presentavano una lunghezza media di circa 1400 token.

7.6.2 Davinci-002 fine-tuned con 6 epoche

Nel progetto viene anche eseguito un confronto del comportamento del secondo modello addestrato utilizzando la completion che include il testo non migliorato. Questa completion presenta il testo suddiviso per ciascun prompt correlato.

Si procede quindi a sottoporre al nuovo modello le stesse domande già poste al modello precedente:

Prompt

"Quali sono i requisiti necessari per accedere all'agevolazione prevista nell'Avviso in riferimento al documento: 'Aiuti agli Investimenti delle Piccole e Medie Imprese (Titolo II Capo 3 Reg. Regionale 17/2014)

br__br__.pdf'"

Risposta

05 gennaio 2016 Pag 1 di 2 Centro Servizi Sistema Puglia www.sistema.puglia.it Avviso 12/2016 Aiuti agli Investimenti delle Piccole e Medie Imprese (Titolo II Capo 3 Reg. Regionale 17/2014) Le modalità di presentazione delle domande di agevolazione sono le seguenti: a) telematicamente, mediante la compilazione ed il successivo invio on line dei moduli messi a disposizione sul sito www.sistema.puglia.it, mediante l'attivazione della procedura "Avviso 12/2016" e della specifica voce "Presenta domanda"; b) telematicamente, mediante la compilazione ed il successivo invio on line dei moduli messi a disposizione sul sito www.sistema.puglia.it, mediante l'attivazione della procedura "Avviso 12/2016" e della specifica voce "Scarica moduli e allegati", di seguito siglati "BI" ed "AA" e di quelle presenti nell'Avviso, in carta semplice, in originale, pena l'inefficacia degli atti, redatte ai sensi degli artt. 1340 e 1341 del codice civile e degli artt. 1414 e 1423

TABELLA 7.5: Tabella contenente prompt e risposta del modello fine-tuned con 6 epoche con completion non migliorata

Dalla risposta generata è possibile fornire alcune ulteriori osservazioni:

Il modello dimostra una notevole fedeltà alla struttura del testo originale presente nei documenti PDF, mantenendo nella risposta esattamente l'intestazione e il numero di pagina. Tale accuratezza può essere un vantaggio nel caso si voglia generare un documento con la medesima struttura.

La risposta generata offre una quantità considerevole di dettagli e informazioni relative alle modalità di presentazione delle domande di agevolazione. Tuttavia, è opportuno valutare che la lunghezza eccessiva può influire sulla chiarezza e la concisione delle risposte, tenendo conto delle preferenze degli utenti per informazioni dirette e sintetiche. Nonostante la fedeltà al formato originale, è evidente la presenza di testo non strettamente correlato alla domanda, dunque non risulta una risposta adatta alle esigenze di un chatbot.

Dalla tabella è possibile notare che entrambe le risposte soddisfano la domanda posta; tuttavia, il modello con la completion migliorata si distingue per la sua notevole capacità di fornire una risposta più chiara e facilmente interpretabile, risultando quindi strutturato in modo più efficace.

In conclusione, tra i due modelli considerati, il primo risulta preferibile. Prima di sottoporre il modello al processo di Fine-Tuning, è consigliabile migliorare il testo di esempio, anche considerando i costi associati al perfezionamento con text-davinci-003.

I costi complessivi sono paragonabili; infatti, il miglioramento della completion ha comportato una spesa di 3.14\$, mentre l'aumento delle epoche da 4 a 6 ha generato un

costo aggiuntivo di 3\$. Tali costi sono stati effettuati con la speranza di ottenere risposte di miglior qualità.

7.6.3 Fine-Tuning di GPT-3.5 Turbo

Per eseguire il fine-tuning di GPT-3.5 Turbo, è stato necessario convertire il dataset precedente nel formato JSONL, strutturato con `{"prompt": " "}` e `{"completion": " "}` in un JSONL più complesso in modo da adattarsi alle specifiche di GPT-3.5 Turbo. Il formato richiesto è organizzato come una lista di *messages*, come mostrato nell'esempio seguente:

```
1 {  
2   "messages": [  
3     {"role": "system", "content": "You are a helpful assistant."},  
4     {"role": "user", "content": "Quale documentazione necessaria per la  
presentazione della domanda di agevolazione? in riferimento al  
documento: 'Scheda MicroPrestito della Regione Puglia - edizione 2021.  
pdf'"},  
5     {"role": "assistant", "content": "Per presentare la domanda di  
agevolazione, necessario produrre i seguenti documenti: - Domanda di  
agevolazione firmata digitalmente; - Dichiarazione di consenso al  
trattamento dei dati personali....."}  
6   ]  
7 }
```

SNIPPET 7.4: Esempio di formato JSONL per il fine-tuning di GPT-3.5 Turbo

Possiamo confrontare le risposte usando lo stesso prompt tra le diverse tecniche:

La risposta generata con la Retriever-Augmented Generation (RAG) fornisce informazioni dettagliate sulla procedura di partecipazione. Raccomanda l'iscrizione al portale Sistema Puglia all'indirizzo <https://microcredito2021.sistema.puglia.it> e fornisce dettagli su documenti necessari e ulteriori approfondimenti disponibili sul sito ufficiale <https://regione.puglia.it/web/guest/bandi-e-avvisi>.

La risposta generata da gpt-3.5-turbo fine-tuned offre una guida più sintetica, evidenziando i passaggi chiave per partecipare all'Avviso MicroPrestito della Regione Puglia. Descrive il processo di registrazione sul portale www.nuvola.puglia.it, la consegna dell'istanza, l'ottenimento di un codice identificativo, la compilazione della domanda online e l'attesa dell'esito della valutazione.

D'altra parte, la risposta generata da davinci-002 con 4 epoche è più esaustiva e si concentra sul contesto dell'Avviso MicroPrestito della Regione Puglia, delineando le finalità, i requisiti e le categorie di imprese eleggibili.

È evidente che ciascun modello ha un approccio unico nel generare risposte. La sinteticità di gpt-3.5-turbo lo rende adatto a un contesto di chatbot conversazionale, mentre la profondità e la specificità di davinci-002 potrebbero essere più adatte a esigenze di informazioni più dettagliate.

Prompt

"Quale è il modo per partecipare all'Avviso MicroPrestito della Regione Puglia? In riferimento al documento: 'Scheda MicroPrestito della Regione Puglia - edizione 2021.pdf'"

Risposta generata da GPT-3.5-Turbo fine-tuned

Il modo per partecipare all'Avviso MicroPrestito della Regione Puglia è il seguente: 1. Registrarsi sul portale www.nuvola.puglia.it e compilare l'istanza di accesso; 2. Consegnare l'istanza presso un Ufficio Postale o un Patronato abilitato; 3. Ottenere un codice identificativo di pratica e inserirlo nel portale NU.VO.LA.; 4. Compilare la domanda online, allegando tutta la documentazione richiesta e sottoscrivendo digitalmente; 5. Attendere l'esito della valutazione; 6. In caso di esito positivo, firmare il contratto di microcredito presso una delle Soggettività Convenzionate (banche o altri operatori finanziari).

Risposta generata con davinci-002 fine-tuned con 4 epoche

Il MicroPrestito della Regione Puglia, in riferimento al documento 'Scheda MicroPrestito della Regione Puglia - edizione 2021.pdf', è un avviso pubblico per la concessione di finanziamenti agevolati finalizzati all'attuazione di progetti imprenditoriali di microimprese e piccole imprese che si attesteranno tra i 25.000,00 euro e i 200.000,00 euro di spesa complessiva ed impatteranno in modo significativo nel breve periodo. L'avviso è rivolto alle microimprese e alle piccole imprese che intendono: 1) Reinvestire i propri utili, anche non distribuiti, o una percentuale del ricavo netto dell'esercizio precedente, destinata al reinvestimento in determinati beni mobili e/o immobili. continua...

Risposta generata con la RAG e GPT-3.5-Turbo non fine-tuned

Il modo per partecipare all'Avviso MicroPrestito della Regione Puglia è presentare la domanda di partecipazione direttamente dall'impresa che richiede l'agevolazione, previa iscrizione al portale Sistema Puglia all'indirizzo <https://microcredito2021.sistema.puglia.it>. La procedura di partecipazione è interamente telematica. continua...

TABELLA 7.6: Tabella con risposte generate dalla Rag e dal modello fine-tuned con gpt-3.5-turbo e davinci-002

7.6.4 Confronto RAG e Fine-Tuning nel chatbot sui Bandi

Dopodichè , si è effettuato un confronto tra la tecnica RAG e il Fine-Tuning. La Retriever-Augmented Generation (RAG) è una tecnica usata per la creazione di Chatbot ottenuta unendo i vantaggi dei modelli di retrieval, che cercano chunk significativi all'interno di un database chiamato Vector Store, insieme ai modelli linguaggio GPT, come gpt-3.5-turbo.

Per implementarla, si è utilizzata la libreria LangChain, che consente la suddivisione del testo e la traduzione in vettori attraverso gli embeddings di OpenAI, una funzione che traduce il testo contenuto nei chunk usando modelli come text-embeddings-002, con Pinecone come database per indicizzare i vettori e i relativi metadata.

Pinecone

Pinecone [22] è un vector-database cloud, e ultimamente anche locale, progettato per migliorare le applicazioni AI attraverso la memorizzazione a lungo termine tramite vector embeddings, un tipo di dato che rappresenta le informazioni semanticamente. Questi vector embeddings consentono alle applicazioni AI o agli LLM di acquisire conoscenza in modo da eseguire task complessi. Rispetto ai database scalari tradizionali che non riescono a stare dietro alla complessità e alla grandezza dei dati di documenti più lunghi e elaborati, Pinecone combina le caratteristiche principali dei database tradizionali con performance ottimizzate degli index vector, supportando operazioni CRUD e query veloci su miliardi di vettori, restituendo così risultati a bassa latenza e precisi, per migliorare ancora di più le performance si possono filtrare le query con namespaces (ad esempio contenuto o titolo) o metadata.

Con il seguente codice vengono inseriti all'interno del Database di Pinecone i vettori dei chunk dei documenti:

```

1  def insert_or_fetch_embeddings(index_name, chunks):
2      import pinecone
3      from langchain.vectorstores import Pinecone
4      from langchain.embeddings.openai import OpenAIEmbeddings
5
6      embeddings = OpenAIEmbeddings()
7
8      pinecone.init(api_key=os.environ.get('PINECONE_API_KEY'), environment=
os.environ.get('PINECONE_ENV'))
9
10     if index_name in pinecone.list_indexes():
11         print(f'Index {index_name} esiste. Loading embeddings ... ', end='')
12     )
13         vector_store = Pinecone.from_existing_index(index_name, embeddings)
14         print('Ok')
15     else:
16         print(f'Creazione di index {index_name} e embeddings ...', end='')
17         pinecone.create_index(index_name, dimension=1536, metric='cosine')
18         vector_store = Pinecone.from_documents(chunks, embeddings,
19 index_name=index_name)

```

```

18     print('Ok')
19
20     return vector_store

```

7.6.5 Query Retrieval

Dopo aver caricato i vector data su Pinecone, contenuti i chunk dei documenti dei Bandi, si può usare il seguente codice per testare la ricerca dei documenti:

```

1     results = vector_store.similarity_search(query)

```

Dove results sarà una lista di k Document che avranno similarità semantica rispetto alla query.

La RetrievalQA è una tecnica comune utilizzata dai chatbot per arricchire le risposte con dati al di fuori dei dati di addestramento del modello di chat utilizzando il vector store di Pinecone come retriever. Per implementare la retrieval, è possibile creare un retriever utilizzando il codice seguente come descritto nella documentazione di Langchain.

```

1 from langchain.chains import RetrievalQA
2 from langchain.chat_models import ChatOpenAI
3
4 llm = ChatOpenAI(model='gpt-3.5-turbo', temperature = 0.7)
5 retriever = vector_store.as_retriever(search_type='similarity',
6     search_kwargs={'k':3})
7 chain = RetrievalQA.from_chain_type(llm = llm, chain_type='stuff',
8     retriever = retriever)

```

Dando in input una query la chain che sarebbe una catena di azioni o di prompt, darà in output dall'LLM la risposta.

```

1 query="Di cosa si parla nel documento Scheda Efficientamento Energetico
2     Edifici Pubblici.pdf"
3 answer = chain.run(query)

```

Per migliorare la ricerca per indirizzare il retriever sui giusti chunk più velocemente possiamo inserire un metadato.

```

1 answer1 = chain.run(query1, metadata = {'label': 'Compilazione della
2     domanda'})

```

Una risposta esempio sarà: answer1 = "Per partecipare all'Avviso MicroPrestito della Regione Puglia, devi presentare la domanda di agevolazione tramite la procedura online "PIA Medie Imprese" disponibile sul sito www.sistema.puglia.it. Dovrai compilare il modulo di istanza di accesso utilizzando la modulistica fornita e apporre la firma digitale. L'istanza di accesso deve descrivere le caratteristiche tecniche ed economiche del progetto integrato e includere la compilazione telematica richiesta dall'Avviso e prevista dalla piattaforma telematica. Saranno effettuate delle verifiche da parte di Puglia Sviluppo e successivamente la Regione adotterà un provvedimento di ammissione o inammissibilità della proposta. Sarà comunicato l'esito dell'esame e, in caso di ammissibilità, saranno indicati i termini per la presentazione della documentazione progettuale."

Capitolo 8

Applicazione chatbot: BandiPugliaBot

Il chatbot in questione è stato sviluppato usando la tecnica RAG in combinazione con il modello fine-tuned di GPT-3.5-Turbo. Utilizzando la libreria di Pinecone e dunque i chunk tradotti in vector data e archiviati come descritto precedentemente, il chatbot dopo il prompt dell'utente è in grado di fare la retrieval di questi chunk per migliorare la risposta, tutto ciò utilizzando

8.1 Streamlit

Streamlit [24] è un framework open-source che consente agli sviluppatori di creare rapidamente applicazioni web per la data science e il machine learning con poche righe di codice Python. È progettato per semplificare e accelerare il processo di creazione di interfacce utente interattive, consentendo agli sviluppatori di trasformare script di analisi dati in app condivisibili. Streamlit si distingue per la sua facilità d'uso, efficienza e la capacità di integrare funzionalità avanzate senza la necessità di conoscenze approfondite in web development.

Il codice di Streamlit relativo al BandiPugliaBot è il seguente:

```

1 # Streamlit code
2 st.title('Chatbot Bandi in corso Sistema Puglia')
3 chatbt_instance = chatbt()
4
5 uploaded_file = st.file_uploader("File upload", type="pdf")
6 if uploaded_file:
7     chatbt_instance.load_pdf(uploaded_file)
8
9 # messaggio di benvenuto
10 with st.chat_message('assistant'):
11     st.write("Ciao, sono il tuo assistente personale personalizzato per
12             rispondere a domande relative ai bandi in corso della regione Puglia
13             presenti sul sito [link](https://www.sistema.puglia.it/)!")
14
15 # Initialize chat history
16 if "messages" not in st.session_state:
17     st.session_state.messages = []

```

```

16
17 # Display chat messages from history on app rerun
18 for message in st.session_state.messages:
19     with st.chat_message(message["role"]):
20         st.markdown(message["content"])
21
22 if user_input := st.chat_input("Inserisci la tua domanda:"):
23     with st.chat_message("user"):
24         st.markdown(user_input)
25
26     result = chatbt_instance.qa({"question": user_input, "chat_history":
chatbt_instance.chat_history})
27     chatbt_instance.chat_history.append([(user_input, result["answer"])]
28     chatbt_instance.qa = chatbt_instance.load_db("stuff", 4)
29     chatbt_instance.vector_store = chatbt_instance.load_vector_store()
30     chatbt_instance.answer = result['answer']
31
32     # Add user message to chat history
33     st.session_state.messages.append({"role": "user", "content": user_input
34 })
35
36 # Display assistant response in chat message container
37 with st.chat_message("assistant"):
38     message_placeholder = st.empty()
39     full_response = ""
40     assistant_response = chatbt_instance.answer
41     # Simulate stream of response with milliseconds delay
42     for chunk in assistant_response.split():
43         full_response += chunk + " "
44         time.sleep(0.05)
45         # Add a blinking cursor to simulate typing
46         message_placeholder.markdown(full_response + " ")
47     message_placeholder.markdown(full_response)
48 # Add assistant response to chat history
49 chatbt_instance.memory.save_context({"input": user_input}, {"output":
full_response})
50 st.session_state.messages.append({"role": "assistant", "content":
full_response})

```

SNIPPET 8.1: Codice Streamlit per Chatbot

Il seguente codice in maniera riassuntiva esegue:


- Inizializzazione e Titolo: La prima parte del codice inizializza l'applicazione Streamlit e imposta il titolo della pagina.
- Caricamento PDF: Viene offerta la possibilità di caricare un file PDF tramite un'interfaccia di upload. Se un file viene caricato, la funzione `load_pdf` della classe `chatbt` viene chiamata per elaborare il documento.
- Messaggio di Benvenuto: Viene visualizzato un messaggio di benvenuto nell'interfaccia chat dell'applicazione, introducendo l'assistente virtuale e il suo scopo.

- **Gestione della Cronologia delle Chat:** Il codice mantiene una cronologia delle interazioni utente all'interno della sessione corrente, utilizzando `st.session_state` per memorizzare i messaggi scambiati.
- **Input dell'Utente e Risposta del Chatbot:** L'utente può inserire domande tramite un campo di input chat. Le domande vengono poi elaborate dalla classe `chatbot`, che utilizza il meccanismo di conversazione e recupero informazioni per fornire una risposta pertinente basata sul contesto del PDF caricato e su altre fonti di dati integrate.
- **Visualizzazione Dinamica delle Risposte:** Le risposte del chatbot vengono visualizzate nella chat in modo dinamico, simulando la digitazione per rendere l'interazione più naturale e coinvolgente.
- **Memorizzazione e Aggiornamento della Cronologia:** Le interazioni vengono continuamente aggiornate e memorizzate, consentendo al sistema di mantenere un contesto delle conversazioni e di apprendere dalle interazioni passate per migliorare la qualità delle risposte future.


Una volta caricato il chatbot sul server di Streamlit è completamente accessibile attraverso il link <https://chatbot-bandi-puglia.streamlit.app/>, la sua interfaccia visibile dall'Utente è la seguente:

Chatbot Bandi in corso Sistema Puglia

File upload

 Drag and drop file here
Limit 200MB per file • PDF

Browse files

 Ciao, sono il tuo assistente personale personalizzato per rispondere a domande relative ai bandi in corso della regione Puglia presenti sul sito [link!](#)


Inserisci la tua domanda: 

FIGURA 8.1: Streamlit Interface per BandiPugliaBot

8.2 Classe chatbt

La classe chatbt è progettata per implementare un chatbot che gestisce conversazioni, interroga database e carica documenti PDF per l'analisi del testo:

```

1 class chatbt:
2     pdf_caricato = False
3
4     def __init__(self):
5         self.chat_history = []
6         self.memory = ConversationTokenBufferMemory(memory_key="
chat_history", return_messages=True, llm=OpenAI(), max_token_limit
=1200)
7         self.qa = self.load_db("stuff", 4)
8         self.pdf_caricato = True
9         self.vector_store = self.load_vector_store()
10
11     def load_db(self, chain_type, k):
12         vector_store = self.load_vector_store()
13         retriever = vector_store.as_retriever(search_type="similarity",
search_kwargs={"k": k})
14         # memory = ConversationBufferMemory(memory_key="chat_history",
return_messages=True)
15
16         qa = ConversationalRetrievalChain.from_llm(
17             llm=ChatOpenAI(model_name='ft:gpt-3.5-turbo-1106:links:gpt-3-5-
signorile:8S8SNEPI',
18                             temperature=0.6,
19                             max_tokens=1024,
20                             model_kwargs= {"frequency_penalty": 0.5}),
21             chain_type=chain_type,
22             retriever=retriever,
23             memory = self.memory,
24             verbose= True
25         )
26         return qa
27
28     def load_vector_store(self):
29         embeddings = OpenAIEmbeddings()
30         directory = 'Documenti/docs/chroma/'
31         # Utilizza Chroma solo se il PDF non è stato caricato
32         if self.pdf_caricato == True:
33             vector_store = Chroma(persist_directory= directory,
embedding_function = embeddings)
34         else:
35             indexname='embedding-bandi'
36             vector_store = PineconeStore(index_name = indexname, embedding
= embeddings)
37         return vector_store
38
39
40     def load_pdf(self, uploaded_file):

```



```
41     directory = 'Documenti/docs/chroma/'
42     temp_dir = tempfile.mkdtemp()
43     path = os.path.join(temp_dir, uploaded_file.name)
44     with open(path, "wb") as f:
45         f.write(uploaded_file.getvalue())
46     pdf_loader = PyPDFLoader(path)
47     pdf_text = pdf_loader.load()
48     text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000,
chunk_overlap=150)
49     docs = text_splitter.split_documents(pdf_text)
50     embeddings = OpenAIEmbeddings()
51     vector_store = Chroma.from_documents(docs, embeddings,
persist_directory= directory)
52
53     chatbt_instance.vector_store = vector_store
54     chatbt_instance.qa = chatbt_instance.load_db("stuff", 4)
55     self.pdf_caricato = True
56     return st.success("Documento PDF caricato con successo!"),
vector_store
```

SNIPPET 8.2: Classe chatbt

Un riassunto di cosa fa la seguente classe è:

- **Inizializzazione:** Al momento della creazione di un'istanza, la classe inizializza una cronologia di chat vuota, configura la memoria per tracciare il contesto delle conversazioni, carica un database di domande e risposte, segnala che un PDF è stato caricato (anche se il caricamento effettivo avviene in seguito), e prepara un nuovo vector embeddings per l'analisi semantica dei documenti caricati dall'utente .
- **Caricamento del Database (load_db):** Questo metodo carica un database di domande e risposte utilizzando un modello di retrieval conversazionale che integra ricerca semantica e capacità di memorizzazione delle conversazioni. Utilizza il modello di OpenAI Fine-Tuned per il processamento delle domande e delle risposte sui Bandi della Regione Puglia.
- **Caricamento dello Spazio di Memorizzazione Vettoriale (load_vector_store):** Prepara uno spazio di memorizzazione vettoriale per l'analisi semantica, utilizzando Chroma per i documenti già caricati o Pinecone per nuovi caricamenti, a seconda dello stato del flag pdf_caricato.
- **Caricamento di PDF (load_pdf):** Carica un documento PDF, estrae il testo, lo divide in segmenti gestibili, genera rappresentazioni vettoriali del testo e aggiorna lo spazio di memorizzazione vettoriale e il sistema di recupero conversazionale per riflettere il nuovo contenuto.

8.3 Questionario Valutazione Usabilità del Chatbot

Dopo aver analizzato i dati raccolti dai questionari sull'usabilità di un chatbot destinato a rispondere ai bandi della regione Puglia, si possono evidenziare diversi aspetti significativi che riflettono le valutazioni complessive degli utenti. Questi risultati si basano sulle risposte fornite da 10 partecipanti.

Innanzitutto, il punteggio medio complessivo del CUQ (Customer Usability Questionnaire) si attesta a 81.56, indicando un livello di usabilità generalmente alto. Questo punteggio riflette la percezione degli utenti riguardo l'efficacia, l'efficienza e la soddisfazione nell'utilizzo del chatbot.

Analizzando i punteggi medi per ciascuna domanda, emergono alcuni trend che meritano attenzione:

- Le domande legate alla facilità d'uso e all'intuitività dell'interfaccia (Q1, Q3, Q5) hanno ricevuto valutazioni elevate, con medie che variano da 4.1 a 4.4 su una scala da 1 a 5, suggerendo che gli utenti trovano il chatbot relativamente facile e piacevole da utilizzare.
- La domanda Q7, che potrebbe riferirsi alla capacità del chatbot di fornire risposte pertinenti e utili, ha ottenuto il punteggio medio più alto di 4.7, indicando un'alta soddisfazione degli utenti riguardo la qualità delle interazioni.
- Le domande Q4, Q12, Q14, e Q16, con medie che vanno da 1.3 a 1.5, potrebbero riguardare aspetti critici dell'esperienza utente, come la gestione degli errori o la complessità delle funzionalità. Questi punteggi bassi suggeriscono aree di miglioramento significative, dove gli utenti hanno riscontrato difficoltà o insoddisfazione.
- In generale, le domande relative alla soddisfazione dell'utente (Q9, Q11, Q13, Q15) hanno ricevuto valutazioni molto positive, con punteggi che variano da 4.3 a 4.8, confermando che, nonostante alcune criticità, l'esperienza complessiva con il chatbot è stata percepita positivamente.

Questi risultati suggeriscono che, mentre il chatbot presenta un'alta usabilità generale e soddisfa le esigenze di base degli utenti nella gestione delle richieste legate ai bandi della regione Puglia, ci sono aree specifiche che richiedono attenzione e miglioramento. In particolare, aspetti legati alla gestione degli errori, alla comprensione delle istruzioni complesse e alla navigazione nell'interfaccia possono essere ottimizzati per elevare ulteriormente l'esperienza utente.

L'adozione di miglioramenti mirati in base a questi feedback può contribuire significativamente a ottimizzare il fine-tuning del chatbot, garantendo che le interazioni siano non solo funzionali ma anche piacevoli e intuitive per l'utente finale. Questo processo di iterazione continua, basato sulla raccolta e sull'analisi di feedback specifici, è cruciale per lo sviluppo di soluzioni di chatbot efficaci e apprezzati dagli utenti.

Conclusioni

Il viaggio intrapreso in questa tesi ha esplorato le frontiere dell'intelligenza artificiale attraverso il fine-tuning di ChatGPT, con un focus specifico sui bandi della Regione Puglia. Attraverso un'esplorazione meticolosa dei Large Language Models (LLMs), il fine-tuning completo, il fine-tuning efficiente in termini di parametri, il Prompt-Engineering, e la Retrieval Augmented Generation (RAG), questo lavoro ha cercato di ottimizzare la risposta di ChatGPT in un ambito tanto specifico quanto complesso.

L'applicazione pratica di questi concetti attraverso il BandiPugliaBot ha dimostrato che è possibile migliorare significativamente l'accessibilità e l'efficacia delle informazioni sui bandi regionali per gli utenti. La valutazione dell'usabilità, condotta con un questionario specifico, ha fornito feedback prezioso che sottolinea non solo l'efficacia dell'approccio adottato ma anche le aree di potenziale miglioramento.

Implicazioni e Riflessioni

La capacità di ChatGPT di fornire risposte pertinenti e contestualmente appropriate ai quesiti sui bandi della Regione Puglia apre nuove vie per l'interazione tra cittadini e amministrazioni pubbliche. Tuttavia, come evidenziato nella sezione "Sfide e Considerazioni Future", questo lavoro non è esente da sfide. La gestione della privacy, l'allineamento dei modelli ai valori umani, la necessità di risorse considerevoli per l'addestramento, e il rischio di generazione di informazioni false o fuorvianti rappresentano questioni cruciali che richiedono un'attenzione continua.

Sviluppi Futuri

Uno sviluppo futuro di quest'applicazione sarebbe quello di automatizzare ulteriormente il processo di Fine-Tuning per adattare dinamicamente ChatGPT alle mutevoli esigenze informative e alle nuove pubblicazioni di bandi. Implementando algoritmi di apprendimento automatico che monitorano in tempo reale le tendenze e le domande degli utenti, si potrebbe sviluppare un sistema capace di auto-aggiornarsi, riducendo così la necessità di interventi manuali nel processo di addestramento e nel mantenimento del modello. Questo porterebbe a un miglioramento della precisione delle risposte fornite, una maggiore personalizzazione dell'esperienza utente, e una riduzione dei tempi di risposta. L'automazione del fine-tuning potrebbe essere estesa anche ad altre aree o funzioni richieste dall'utente, trasformando ChatGPT in uno strumento ancora più versatile e adattabile.

Un altro sviluppo potenziale riguarda l'analisi completa del sito web per l'addestramento del chatbot su ogni sezione specifica, mirando a creare un assistente specializzato

per il suddetto sito. Questo addestramento sarebbe personalizzato e si aggiornerebbe automaticamente ad ogni rinnovamento del sito web e dei documenti in esso contenuti. In aggiunta, il chatbot sarebbe progettato per apprendere e rispondere in modo ottimale alle domande più frequenti poste dagli utenti.

In conclusione, questo lavoro di tesi rappresenta un passo importante verso la comprensione e l'applicazione pratica dei modelli di linguaggio avanzati. La strada da percorrere è ancora lunga e ricca di sfide, ma le possibilità e le potenzialità che si aprono sono immense e promettenti. La collaborazione interdisciplinare tra ricercatori, sviluppatori, e policy makers sarà fondamentale per navigare il futuro degli LLMs in modo etico ed efficace, assicurando che la tecnologia continui a servire l'umanità nel modo più positivo possibile.

Ringraziamenti

In questi lunghi che mi hanno portato alla realizzazione di questa tesi e dunque alla conclusione della triennale, ci sono state molte persone che hanno avuto un ruolo fondamentale nel mio percorso. Primo fra tutti, desidero esprimere la mia più profonda gratitudine ai miei genitori. Il loro incondizionato sostegno, la fiducia che hanno riposto in me e i sacrifici che hanno fatto lungo tutta la mia carriera universitaria sono stati il faro che ha illuminato il mio cammino. Senza il loro amore, la loro pazienza e il loro incoraggiamento, non sarei la persona che sono oggi.

Vorrei inoltre ringraziare i miei colleghi e amici, la mia rete di supporto quotidiano. Grazie per tutte le discussioni stimolanti, per i momenti di condivisione e per le risate che abbiamo condiviso. La vostra presenza ha reso questo viaggio meno arduo e decisamente più gioioso.

Un ringraziamento speciale a Ester che mi ha supportato e ha reso possibile la mia laurea concretamente, svolgendo progetti di coppia e fornendomi il materiale per comprendere più facilmente qualsiasi esame, rispondendo a qualsiasi mia domanda e dubbio.

Un ringraziamento anche a Tommaso, Donatello che mi ha aiutato nel comprendere gli esercizi di analisi e tutti amici e colleghi.

Un ringraziamento speciale va anche a Lucrezia. Grazie per aver creduto in me anche quando dubitavo di me stesso, per avermi supportato nei momenti di stress e per aver celebrato con me ogni piccolo successo.

Grazie a tutti voi per aver camminato al mio fianco in questa importante fase della mia vita. Le vostre parole di incoraggiamento, i vostri consigli e la vostra fiducia in me sono stati i pilastri su cui ho costruito il mio successo.

Bibliografia

- [1] Colin Raffel et al. «Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer». In: *Journal of Machine Learning Research*. 2020.
- [2] Jason Wei et al. «Chain of Thought Prompting Elicits Reasoning in Large Language Models». In: *arXiv preprint arXiv:2201.11903* (2022).
- [3] Tom B. Brown et al. «Language Models are Few-Shot Learners». In: *arXiv preprint arXiv:2005.14165* (2020).
- [4] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].
- [5] Steven Y. Feng et al. *A Survey of Data Augmentation Approaches for NLP*. 2021. arXiv: 2105.03075 [cs.CL].
- [6] Zihao Fu et al. *On the Effectiveness of Parameter-Efficient Fine-Tuning*. 2022. arXiv: 2211.15583 [cs.CL].
- [7] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL].
- [8] <https://neo4j.com/developer-blog/fine-tuning-retrieval-augmented-generation/>. «Supervised fine-tuning flow.» In: *Neo4J* (2023).
- [9] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL].
- [10] Jie Huang e Kevin Chen-Chuan Chang. *Towards Reasoning in Large Language Models: A Survey*. 2023. arXiv: 2212.10403 [cs.CL].
- [11] YuHe Ke et al. *Development and Testing of Retrieval Augmented Generation in Large Language Models – A Case Study Report*. 2024. arXiv: 2402.01733 [cs.CL].
- [12] Dawid J. Kopiczko, Tijmen Blankevoort e Yuki M. Asano. *VeRA: Vector-based Random Matrix Adaptation*. 2024. arXiv: 2310.11454 [cs.CL].
- [13] Langchain. *Getting Started - Introduction*. 2024. URL: https://python.langchain.com/docs/get_started/introduction (visitato il 23/02/2024).
- [14] LangChain. *RecursiveCharacterTextSplitter Function Documentation*. 2024. URL: <https://www.langchain.com/docs/recursivecharactertextsplitters>.

- [15] Patrick Lewis et al. «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks». In: *Advances in Neural Information Processing Systems*. A cura di H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.
- [16] Vladislav Lialin, Vijeta Deshpande e Anna Rumshisky. *Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning*. 2023. arXiv: 2303.15647 [cs.CL].
- [17] Yuning Mao et al. *Generation-Augmented Retrieval for Open-domain Question Answering*. 2021. arXiv: 2009.08553 [cs.CL].
- [18] Humza Naveed et al. *A Comprehensive Overview of Large Language Models*. 2024. arXiv: 2307.06435 [cs.CL].
- [19] OpenAI. «Fine-Tuning». In: <https://platform.openai.com/docs/guides/fine-tuning> (2023).
- [20] OpenAI. *GPT-3.5-turbo: How many tokens are in my text string?* 2022. URL: <https://platform.openai.com/docs/guides/turbo/how-many-tokens-are-in-my-text-string>.
- [21] OpenAI. *Use Cases of Embeddings*. <https://platform.openai.com/docs/guides/embeddings/use-cases>. Accessed: 2023-02-21. 2023.
- [22] Pinecone Systems Inc. *Overview of Pinecone: A Vector Database for Machine Learning Applications*. <https://docs.pinecone.io/docs/overview>. Accessed: 2024-02-26. 2023.
- [23] Alec Radford e Karthik Narasimhan. «Improving Language Understanding by Generative Pre-Training». In: 2018. URL: <https://api.semanticscholar.org/CorpusID:49313245>.
- [24] Streamlit. *Streamlit Documentation*. <https://docs.streamlit.io/>. Accessed: 2024-02-26. 2023.
- [25] Ashish Vaswani et al. «Attention is all you need». In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.
- [26] Wayne Xin Zhao et al. *A Survey of Large Language Models*. 2023. arXiv: 2303.18223 [cs.CL].