KDD Challenge 2023

Antonio Renato Montefusco, Luigi Gallo

[a] *Università degli Studi di Salerno, Fisciano,*

Abstract

Riuscire a predirre la prossima ricerca di un utente in base alle ultime fatte è un task molto arduo, sarebbe molto utile però sia lato consumatore che dal lato dell'azienda per quanto riguarda la fidelizzazione. L'azienda otterrebbe sicuramente maggiori entrate, suggerendo all'utente cosa cercare lo invoglia a comprare altri oggetti correlati a quelli già cercati, questo lo spingerebbe ad acquistare di più. Lato utente invece velocizzerebbe la sua spesa facendogli risparmiare tempo per effettuare le varie ricerche.
Questo task rientra nell'ambito dell'*elaborazione del linguaggio naturale*, ambito di ricerca molto in voga in questo momento, basti pensare all'esplosione di *Chat-GPT* che ha fatto scalpore anche tra utenti poco informati sull'intelligenza artificiale. In questo lavoro è stato utilizzato lo *Multilingual Shopping Session Dataset* un grande dataset composto da sessioni utente divise per lingua. Questo dataset è stato rilasciato direttamente da *Amazon* per promuovere la ricerca nel settore dell'*NLP* legato alle ricerce utente.
Andremo a descrivere il dataset, il processo di pulizia dei dati e il modello utilizzato come soluzione al problema di generazione della prossima ricerca dell'utente.

1. Introduzione

Un task che attualmente è molto in voga nell'ambito del *Natural Language Processing* riguarda la profilazione di un utente in base alle sue ultime ricerche o prodotti visualizzati. Il task che verrà analizzato sarà la previsione di un prodotto data una sessione utente, dove con sessione utente si intende una serie di prodotti, in ordine cronologico, con cui l'utente ha interagito o visualizzato durante la sua navigazione all'interno dell'e-commerce. Si cercherà quindi di dare in output una stringa che rappresenti un oggetto con cui l'utente potrebbe interagire in base agli oggetti visualizzati precedentemente. Questa stringa non farà riferimento per forza ad un oggetto presente all'interno del dataset fornito sarà una previsione riguardante i prodotti già presenti, quelli appena inseriti, definiti come *cold-start product* perché sono prodotti nuovi o con poche interazioni. Questi sicuramente sono i più difficili da raccomandare dato che si avranno poche informazioni.
In particolare caso il nostro input sarà costituito dalla provienzienza dell'utente (UK,JP,DE,IT o ES) e da una sua sessione.
La sessione di un utente sarà costituita da una serie di prodotti con cui esso ha interagito trimite dei click sul popolarissimo sito di e-commerce Amazon.
L'output sarà invece costituito da un titolo di prodotto, senza che quest'ultimo sia necessariamente contenuto dei dati preesistenti.

Esempio di input:

locale	**example session**
UK	[product$_1$, *product*$_2$, *product*$_3$]
DE	[product$_4$, *product*$_5$]

Esempio di output:

next item title
”toilet paper tube”
”bottle of ink”

I titoli così generati sono potenzialmente una risorsa molto preziosa.
Questi possono potenzialmente migliorare numerosissime attività di profilazione, che per un sito di e-commerce sono molto importanti in quanto incentivano l'utente ad acquistare più prodotti. Di seguito verrà riportato un esempio preso dal dataset. L'utente ha visualizzato i seguenti titoli di prodotti su *amazon IT* secondo questo ordine:

- Beurer Ih 26 Aerosol Con Doccia Nasale E Accessori Completi Inclusi
- Pic Solution - AirFamily Evolution Aerosol
- Beurer Ih 26 Aerosol Con Doccia Nasale E Accessori Completi Inclusi
- Pic Solution Aircube Aerosol A Pistone
- Beurer Ih 26 Aerosol Con Doccia Nasale E Accessori Completi Inclusi
- Pic Solution - AirFamily Evolution Aerosol

Si potrebbe quindi suggerire all'utente qualcosa di correlato, presente all'interno dell'e-commerce, che lo porterebbe ad ac-

quistare altri prodotti, quindi a dargli altri input per le sue ricerche. Tra i prodotti presenti in catalogo si potrebbe suggerire l'oggetto con titolo: *Laica NE2014 Apparecchio a Pistone per Aerosolterapia*

2. Lavori correlati

La raccomandazione dell'elemento successivo è un problema di ricerca importante e ben studiato. I primi lavori di ricerca hanno proposto le catene di Markov per modellare le relazioni di ordine basso tra gli elementi per la raccomandazione dell'elemento successivo [3]. Con il progresso dei modelli neurali, le reti neurali profonde sono state applicate alla modellazione di modelli sequenziali che portano a una migliore accuratezza delle raccomandazioni [1]. Ricerche recenti hanno anche esplorato l'uso dell'aumento dei dati e dell'apprendimento contrastivo per migliorare le rappresentazioni degli utenti e degli elementi, migliorando così ulteriormente le prestazioni delle raccomandazioni [21]. Una soluzione che si è dimostrata essere particolarmente valida in questi casi è l'uso di un *Large Language Model*.
L'ambito dei *Large Language Model* è un'area di ricerca molto attiva, infatti il modello utilizzato da noi, *Llama*, è stato rilasciato nel febbraio di quest'anno. In generale questi modelli si basano su architetture avanzate di deep learning che permettono di generare testi in linguaggio naturale coerenti e comprensibili.
Prima dei *Large Language Model* si utilizzavano approcci statistici per generare testo, questi però avevano bisogno di una serie di regole e risorse progettate manualmente, però spesso le regole andavano a scontrarsi con l'ambiguità del linguaggio naturale.
I nuovi modelli di linguaggio utilizzano *Reti Neurali Ricorrenti*[15] o i *Transformers*[17]. Purtroppo però, per via dell'elevato sforzo computazionale e dell'enorme memoria necessaria non si sono diffusi su larga scala.
Oggigiorno però i *Large Language Model* hanno avuto un riscatto, già col l'esplosione di *GPT*[8] rilasciato da *OpenAI* nel 2018, questo modello era estremamente efficace nel generare testi coerenti senza avere problemi di ambiguità, è stato addestrato su un'elevato quantitativo di dati proveniente da internet, questo ha permesso a GPT di essere un modello multilingua.
Nel tempo sono stati sviluppati altri modelli come *T5* di Google Research che ha introdotto un approccio unificato per diverse attività di linguaggio naturale, *BERT* che ha introdotto il preaddestramento bidirezionale.
Tra i precedenti sforzi nella raccomandazione basata su LLM, Zhang et al. [20] ha proposto di utilizzare GPT-2 [9] o BERT [2] come raccomandazione principale, facendo la previsione del prossimo film sulla base di cinque film precedentemente guardati dall'utente target. Tuttavia, l'enorme spazio di raccomandazione e la modellazione inadeguata delle preferenze dell'utente lo rendono poco performante e parecchio esoso in termini di risorse.
La nostra soluzione sarà basasta su Llama, di Meta, uno dei migliori *Large Language Model* in circolazione, questo ha rivoluzionato il mondo dei Large Language Model, fornendo un modello leggero e performante più di altri modelli che utilizzano più del doppio dei suoi parametri.

3. Dataset

Il dataset proposto dalla competizione è il *Multilingual Shopping Session Dataset*[10], questi dati vengono divisi in due macro-dataset: il primo contenente delle sessioni utente anonime, per semplicità successivamente ci si riferirà a questa porzione con il nome di *Sessions*; il secondo contenente informazioni riguardo ai singoli prodotti, verrà indicato con il nome di *Products*.

Il dataset *Sessions* è formato da tre attributi:

- *Prev_items*, formato da una lista di *idProdotto* che fa riferimento al dataset *Products*, questi *idProdotto* indicano i prodotti cliccati dall'utente in quella sessione;
- *Next_item*, indica l'id del prodotto successivo da indovinare, questo attributo è presente solo nel dataset di training;
- *Locale*, indica la lingua della sessione utente.

Il dataset *Products* è formato da dieci attributi:

- *Id*, identificativo alfanumerico di un prodotto, questo non è univoco ma lo è per la stessa lingua, quindi assieme formano la chiave primaria del dataset.
- *Locale*, indica la lingua del prodotto (esempio: "IT");
- *Title*, titolo del prodotto (esempio: "Apple iPhone 14 Pro Max 512 Nero siderale");
- *Brand*, brand del prodotto (esempio: "Apple");
- *Price*, prezzo del prodotto (esempio: 1400.00);
- *Color*, colore del prodotto (esempio: "Nero siderale");
- *Size*, grandezza del prodotto (esempio: 512)
- *Model*, modello del prodotto (esempio: "MQAF3QL")
- *Material*, materiale del prodotto (esempio: "Alluminio")
- *Author*, autore del prodotto (esempio:"J.K. Rowling")
- *Desc*, descrizione accurata del prodotto presente nella pagina amazon (esempio:"A16 Bionic evoluto chip smartphone Reti cellulari 5G ultrarapide")

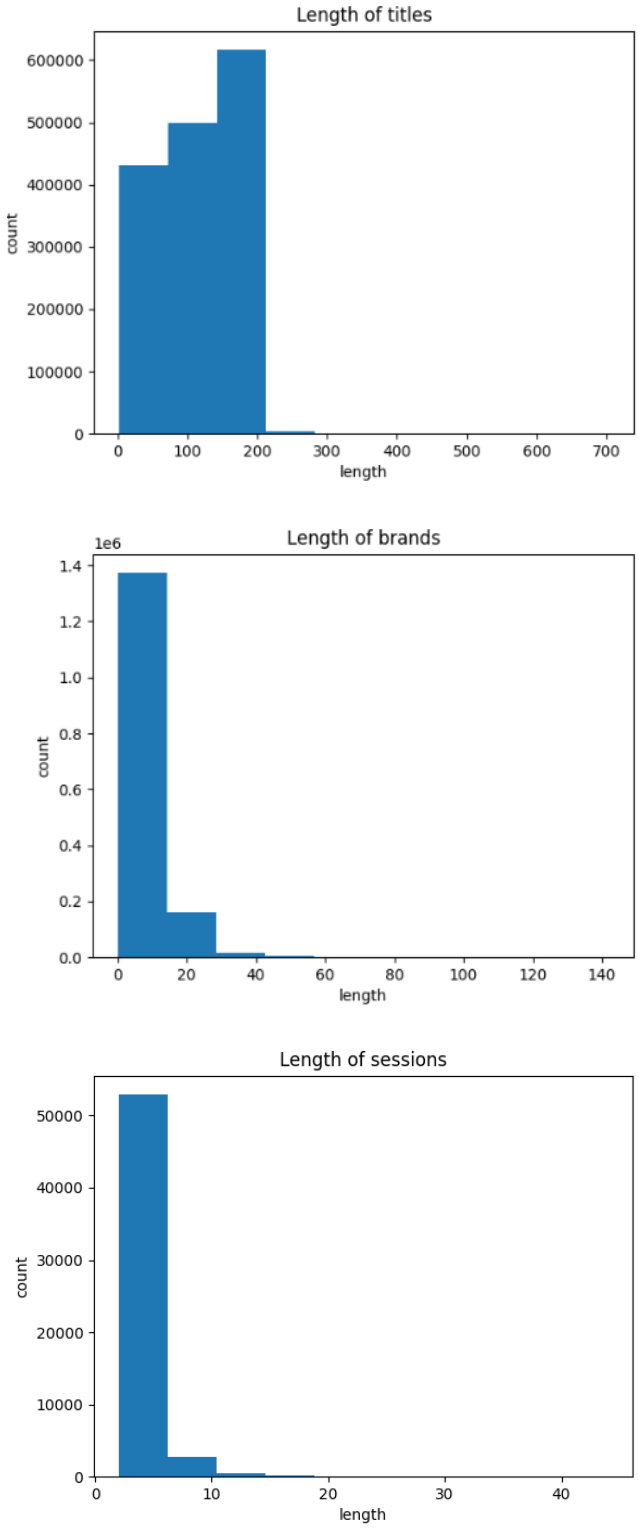

Figura 1: Istogrammi per attributo e lunghezza delle sessioni

Come si può intuire il dataset è multilingua, più precisamente presenta sessioni e prodotti in sei lingue: Tedesco, Giapponese, Inglese, Spagnolo, Francese, Italiano. Il dataset non presenta un'equa distribuzione tra prodotti sessioni e lingue, a seguire la Tabella 1 riassuntiva dei dati contati per lingua.

Si noti che l'attributo *title* tende spesso a accogliere informazioni, spesso contenute anche in altri attribuiti, considerate rilevanti per l'elemento in questione (*title* contiene *brand* nel 54.85% dei casi).
Considerando la raccolta dati a disposizione ed il task assegnato l'uso di un language model risulta essere una scelta particolarmente valida per giungere ad una soluzione efficace.

4. Occorrenza desiderata

Un esempio di occorrenza desiderata, considerando sessioni di tipo *"IT"*, potrebbe essere formata nel seguente modo:

Input	
Session	
id	**title**
B0BDK9WHRF	Apple iPhone 14 Pro Max 512 Nero siderale
B0B8J6WTPS	JETech Privacy Pellicola Protettiva Copertura Totale iPhone 14 Pro Max

Output
next item title
"Apple Nuovo AirPods Max - Nero"

Quindi *Amazon* potrà avvalersi di un efficace sistema di consigliati che porterà un duplice beneficio:

- Lato consumer, le ricerche degli articoli risulteranno facilitate, in quanto non limitate più dalle informazioni "statiche" contente nel database.
- Lato seller, come diretta conseguenza dei consigli dinamici, si registrerà un aumento delle vendite.

5. Preprocessing

Quando si parla di problemi di natural language processing è molto importante fare attenzione ai dati su cui si va a lavorare, infatti nel dataset *Produts*, per la colonna descrizione in particolare, sono presenti diversi caratteri che possono essere problematici per il modello, come ad esempio la presenza di emoji

Tabella 1: Statistiche del DataFrame

	Statistic	price	id	locale	title	brand	color	size	model	material	author	desc
0	mean	1806107.12	10.00	2.00	117.26	9.02	8.05	11.75	8.98	10.39	11.91	153.66
1	std	8302930.14	0.00	0.00	57.70	5.67	6.88	7.53	6.09	13.34	5.86	117.47
2	min	0.00	10.00	2.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
3	25%	11.19	10.00	2.00	66.00	6.00	4.00	6.00	6.00	5.00	8.00	52.00
4	50%	23.28	10.00	2.00	119.00	7.00	6.00	11.00	8.00	7.00	12.00	135.00
5	75%	799.00	10.00	2.00	171.00	11.00	10.00	17.00	11.00	10.00	15.00	226.00
6	max	40000000.07	10.00	2.00	704.00	142.00	986.00	402.00	929.00	1848.00	300.00	1000.00

o di tag HTML. È stata quindi effettuata una pulizia del dataset *Produts* per ottenere una migliore generalizzazione, questa pulizia è stata fatta secondo i seguenti punti:

- Rimozione del rumore
  - Rimozione di emoji
  - Rimozione di tag HTML
  - Rimozione della punteggiatura
  - Rimozione degli URLs Lower Casing
  - Lemmatization

La rimozione del rumore consiste nell'eliminare caratteri o porzioni di testo ininfluenti per la comprensione semantica di un testo. Come ad esempio, ci siamo accorti che nella colonna *"desc"* erano presenti diverse emoji, queste non sono molto utili per comprendere il prodotto, tramite la libreria *demoji*, che permette di sostituire le emoji con stringhe di testo, in questo caso sono state sostituite con stringhe vuote.
Sfruttando la libreria BeautifulSoup, libreria utilizzata principalmente per effettuare *web scraping*, sono stati rilevati ed eliminati gli URLs e i tag HTML utilizzati per l'impaginazione dei testi. In generale, quando si deve fare comparazione di testi si cerca di ignorare le parole ininfluenti per concentrarsi solo su quelle di effettiva rilevanza. Di fatto sono state eliminate le stopwords, parole che non aggiungono significato semantico al testo(ad esempio gli articoli in italiano). Queste parole sono state eliminate tramite la libreria *stopwords* che contiene stopword secondo l'*ISO 639-1* di diverse lingue. Inoltre è stato impostato tutto il testo in lower case in modo tale da non far differenze tra maiuscolo e minuiscolo nella tokenizzazione.
In ogni lingua, per motivi grammaticali, si utilizzano diverse versioni della stessa parola, ad esempio i verbi coniugati o dei sinonimi per evitare ripetizioni all'interno di un testo. Ad esempio le parole *mangio, mangi, mangia* viene trasformata nel verbo *mangiare*, anche i sinonimi vengono unificati, ad esempio *felice, allegro, contento* vengono raggruppate nella parola *felice*. In generale, la lemmetizzazione si basa sull'utilizzo del vocabolario della lingua in questione per tornare la forma base della parola. Questo è stato fatto tramite la libreria *c*he supporta le lingue del dataset quindi *Inglese, Tedesco, Spagnolo, Giapponese, Italiano e Francese.*

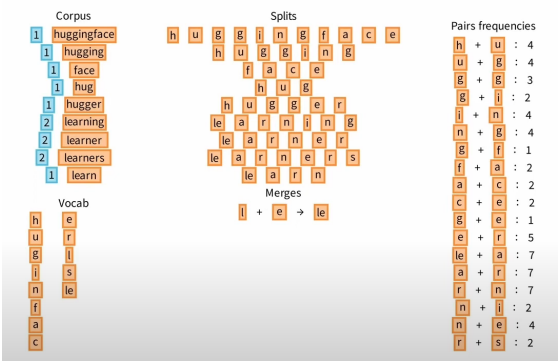

Figura 2: Prima iterazione di tokenizzazione tramite BPE

5.1. Tokenization

Il modello scelto per la challenge è *LLAMA*, un language model proprietario di *Meta* disponibile in diverse versioni da 7B, 13B, 33B e 65B di parametri. Si utilizzerà quindi il tokenizer nativo del modello, esso utilizza un adattamento dell'algoritmo *Byte-Pair Encoding(BPE)*[13] implementato tramite la libreria Sentence-Piece[7]. L'algoritmo Byte-Pair Encoding è diventato molto popolare nell'ambito del machine learning e dell'NLP, in pratica suddivide le parole in sotto-token più piccoli in base alla frequenza all'interno del testo, questo permette di generare una tokenizzazione compatta e propensa alla generalizzazione. In figura 2 vi è un esempio della tokenizzazione tramite BPE sul testo *"This is the Hugging Face Course. This chapter is about tokenization. This section shows several tokenizer algorithms."* preso dal sito *Hugging Face*[6], nel quale viene spiegato l'algoritmo di tokenizzazione, dove ogni parola viene presa singolarmente per poi essere spezzata in singole lettere che formano dei token, questi formeranno un primo vocabolario, si passa poi alle coppie di token dove, ad ogni iterazione, viene aggiunta al vocabolario la coppia di token con frequenza più alta. Si arriverà alla fine dell'algoritmo come nella figura 3. In totale LLAMA utilizza *1,4 trilioni* di token.

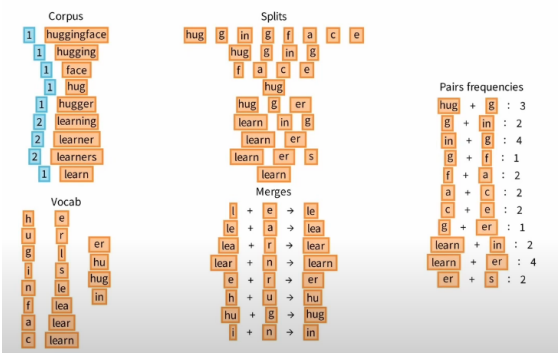

Figura 3: Fine algoritmo di tokenizzazione tramite BPE

6. Metriche di valutazione

La metrica di valutazione per questo task è la *bilingual evaluation understudy* (BLEU).
BLEU è una metrica ampiamente utilizzata per valutare i sistemi di elaborazione del linguaggio naturale (NLP) che producono testo, in particolare quelli che riguardano la traduzione automatica (MT) e la generazione di linguaggio naturale (NLG)[12].
BLEU viene valutata su un gruppo di n-gram:

$$\text{BLEU}=\text{BP}\cdot exp(\sum n = 1Nwnlogpn)$$

- *BP* è la penalità di brevità.
- *N* è la lunghezza massima di n-grammi utilizzata per calcolare i punteggi di precisione.
- *wn* è il peso assegnato a ciascun punteggio di precisione.
- *exp* è la funzione esponenziale.
- *pn* è il punteggio di precisione per ogni n-gram.

Il punteggio di precisione *pn* è il rapporto tra il numero di n-gram che compaiono in uno qualsiasi dei riferimenti, e il numero totale di n-gram nel candidato. Matematicamente, *pn* è calcolato come segue:

$$\text{pn}=\sum \in Cmin(countnC(s), maxr \in Rcountnr(s)) \sum \in CcountnC(s)$$

- *countnC(s)* è il conteggio di n-gram s nel candidato C.
- *countnr(s)* è il conteggio di n-gram s nel riferimento r appartente ad R, dove R è un insieme di riferimenti.

La penalità di brevità (*BP*) è un fattore di correzione che penalizza la generazione candidata troppo corta rispetto al riferimento. La penalità di brevità è calcolata come segue:

$$\text{BP=1,if Lc <Lrexp(1LrLc),ifLc<=Lr}$$

- *Lc* è la lunghezza della generazione.
- *Lr* è la lunghezza del riferimento più corto.

In generale, il punteggio BLEU varia da 0 a 1 (punteggi più alti che indicano una generazione migliore).
Per questo task è impostato *N=4* (ovvero BLEU-4) con *wn=1/N*.

7. Architettura

Llama[16] è un modello pre-addestrato sviluppato da Meta. Si basa su un architettura Transformer[17] per eseguire task di NLP, l'architettura è stata modificata per migliorarne le performance e i tempi di esecuzione. Esistono diverse versioni di Llama, da 7B a 65B, già dalla versione a 13B performa molto meglio di GPT-3(che ha 175B), inoltre la versione a 65B tiene testa ai migliori language model in circolazione come Chinchilla-70B e PaLM-540B.
Queste migliorie sono dovute a diverse modifiche che verranno discusse in questo capitolo, per queste modifiche si sono ispirati a "Chinchilla scaling laws"[4], la quale indica che ci deve essere almeno un rapporto di 20 a 1 tra il numero di token e il numero di iperparametri per avere un training ottimale, infatti *DeepMind* utilizzando un modello a 70B (Chinchilla-70B) è riuscita ad ottenere performanche migliori di un modello che utilizzava quattro volte quei iperparametri (Gopher-280B) ma utilizzava un dataset ristretto. Tra le varie modifiche che sono state fatte c'è stata l'idea di normalizzare l'input, ad differenza della maggior parte dei modelli in circolazione che invece normalizzano l'output, la normalizzazione è stata effettuata tramite la funzione *RMSNorm normalizing*[19].
Hanno anche cambiato la funzione di attivazione, di solito si utilizza una *ReLU*, ma è stato deciso di utilizzarne una variante, la funzione *SwiGLU* che ha permesso di migliorare di gran lunga le performance[14].
Hanno poi deciso di sostituire i positional embeddings con i *Rotary Positional Embeddings*, una tecnica utilizzata nel campo dell'apprendimento automatico per rappresentare la posizione relativa di un oggetto in un contesto multidimensionale. Come optimizer hanno utilizzato il classico *AdamW optimizer* con parametri *beta1=0.9* e *beta2=0.95*, utilizzando un learning rate schedule con learnging rate pari a 10%. Come self-attention è stata utilizzata una versione efficiente del *multi-head attention* per migliorare la velocità del modello, tale implementazione è presente nella libreria *xformers*[11], quest'implementazione

LLaMA	Model hyper parameters					
Number of parameters	dimension	n heads	n layers	Learn rate	Batch size	n tokens
7B	4096	32	32	3.0E-04	4M	1T
13B	5120	40	40	3.0E-04	4M	1T
33B	6656	52	60	1.5.E-04	4M	1.4T
65B	8192	64	80	1.5.E-04	4M	1.4T

Figura 4: Iperparametri del modello

permette di ridurre la memoria in uso e i tempi di computazione.
Sono stati poi inseriti dei checkpoint per ridurre i calcoli fatti durante il *backward pass*, in pratica si salvano le attivazioni più costose per poi riutilizzarle. Questo vuol dire che hanno reimplementato la funzione di backward invece che utilizzare quella di *Pytorch*.
Hanno implementato anche un sistema di per lavorare in parallelo su più GPU attraverso la rete, questo gli ha permesso di effettuare l'addestramento per la versione *Llama-65B* utilizzando *2049 Nvidia A100 con 80GB di RAM l'una*, il dataset conteneva *1.4T di token* e ci sono voluti approssimativamente 21 giorni.

8. Esperimento

Le migliori architetture per i task di NLP fanno uso di architetture Transformer [18]. Tra le varie architetture presenti in letteratura emerge LLaMA, preferibile per il nostro scopo in quanto molto meno esoso in termini di risorse rispetto ad altri modelli. LLaMA è un language model autoregressivo basato su un'architettura a Transformer. Il modello è disponibile in diverse dimensioni:

Requisiti LLaMA 8 bit	
Numero iperparametri del modello	**VRAM**
7B	24 GB
13B	32 GB
33B	64 GB
65B	128 GB

Alla luce dei requisiti emersi, nel nostro caso, risulta opportuno optare per il peso *7B*.

Per ottimizzare ulteriormente la velocità di training il modello è stato integrato con una tecnica chiamata LoRA(Low-Rank Adaptation of Large Language Models) [5]. LoRA riduce il numero di parametri di training imparando da coppie di matrici scoposte per rango mentre mantiene invariati i pesi originali. Ciò permette non solo di ridurre notevolmente i requisiti di archiviazione dei Large Language Models come LLama ma permette anche un task-switch più efficiente, il tutto senza introdurre latenza.
Per essere dato in input al modello, il dataset, una volta superata la fase di preprocessing, è stato converito da CSV a JSON. Il file JSON così generato sarà un dizionario contente i seguenti campi:

- *Instruction*, stringa che descrive l'attività che il modello dovrebbe seguire.
- *Input*, stringa che raprpesenta contesto facoltativo oppure l'input dell'attività.
- *Output*, stringa che rappresenta la risposta all'istruzione.

Inoltre, al fine di effetturare degli step di training più rapidi,il dataset è stato partizionato in quattro parti distribuite equamente. Il modello verrà addestrato usando una singola partizione per volta. Alla fine di ogni fase di training verrà generato un checkpoint da cui il modello potrà eventualmente ripartire.
Un'ulteriore d'ottimizzazione applicata è stato quello di impostare la variabile:

$$torch\_type = torch.float16$$

In questo il modello sarà costretto ad usare float a 16 bit invece che a 32 bit. Questa scelta consiste in un *tradeoff*, da una parte il nostro modello occuperà la metà dello spazio in VRAM ma dall'altra otterremo delle prestazioni peggiori.
Un ulteriore *tradeoff* per consentire l'esecuzione del modello su schede video meno potenti è sfruttare la *quantizzazione 4 bit*.

Requisiti LLaMA 4 bit	
Numero iperparametri del modello	**VRAM**
7B	12 GB
13B	16 GB
33B	32 GB
65B	64 GB

Spesso, nonostate le ottimizzazioni applicate, può capitare di non avere a disposzione una GPU adeguata quindi in questi casi si hanno a disposizione due scelte.
O si procedere usando la CPU e la memoria RAM al posto della GPU oppure affidarsi a piattaforme esterne come *Google Colab*.
Solitamente l'uso della CPU non è una strade perseguibile in quanto richiede una quantità troppo elevata di tempo.
Nel nostro caso abbiamo utilizzato uno *Xeon W 18-core*, che per fare training sulla prima metà del dataset (circa 1.8 Gb), ha impiegato più di 3 settimane.
Alla luce di queste considerazioni la scelta migliore risulta quella di utilizzare piattaforme esterne come *Google Colab* dove, sfruttando le opportune ottimizzazioni sopracitate possiamo eseguire il training di LLaMA 7B 4bit mediante l'acquisto di un piano adeguato alla mole di lavoro che stimiamo di dover compiere.

9. Conclusioni e sviluppi futuri

In questo lavoro abbiamo analizzato ed elaborato il dataset fornito nel corso della KDD'23 Cup in modo tale da renderlo maggiormente adatto alla corretta risoluzione del problema che ci è stato proposto. La nostra analisi ha anche sottolineato le principali questioni che dovrebbero essere affrontate nella ricerca futura in questo settore. In particolare, per quanto riguarda le differenze linguistiche, è stato necessario applicare un opportuno processo di tokenizzazione. Quest'ultimo è risultato fondamentale per poter uniformare tutti i dati al fine di facilitare l'estrazione semantica del contesto e allo stesso tempo essere conformi agli standad esistenti adottati solitamente nei task di NLP. A causa della mancanza di strumenti adeguati non abbiamo portato a termine l'addestramento del modello, tuttavia proprio grazie a questo imprevisto abbiamo approfondito, offrendo diverse soluzioni, una delle problematiche più comuni per chi lavora in questo settore, ossia la mancanza di hardware adeguato.
In futuro, sarà necessario applicare ulteriori tecniche per migliorare l'integrazione di dati tra le varie lingue. Un esempio potrebbe essere sfruttare le conoscenze linguistiche per applicare delle tecniche di "pruning" in un contesto semantico. In oltre sarà opportuno, in presenza di strumentazione adeguata, realizzare un effettivo modello in maniera da effettuare test raffinando l'attuale processo di fine-tuning.
Infine pensiamo che il modello proposto, per la sua potenziale dinamicità nelle risposte generate, possa essere considerato come un buon punto di partenza per migliorare i motori di ricerca basati sulla semantica garantendo all'utente una migliore user experience.

Riferimenti bibliografici

[1] Jianxin Chang, Chen Gao, Yu Zheng, Yiqun Hui, Yanan Niu, Yang Song, Depeng Jin, and Yong Li. Sequential recommendation with graph neural networks. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 378–387, 2021.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 191–200. IEEE, 2016.

[4] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[5] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[6] HuggingFace. Byte-pair encoding tokenization, 2022.

[7] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[8] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. 2018.

[9] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[10] Chandan K Reddy, Lluís Màrquez, Fran Valero, Nikhil Rao, Hugo Zaragoza, Sambaran Bandyopadhyay, Arnab Biswas, Anlu Xing, and Karthik Subbian. Shopping queries dataset: A large-scale esci benchmark for improving product search. *arXiv preprint arXiv:2206.06588*, 2022.

[11] Meta Research. xformers, 2021.

[12] Federica Russo. Corso di laurea magistrale in interpretazione (classe lm-94).

[13] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[14] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[15] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[16] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[19] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

[20] Yuhui Zhang, Hao Ding, Zeren Shui, Yifei Ma, James Zou, Anoop Deoras, and Hao Wang. Language models as recommender systems: Evaluations and limitations. 2021.

[21] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 1893–1902, 2020.